Aplicaciones de Machine Learning como herramienta de apoyo a la planificación del operativo de SAMUR-PC

Machine Learning applications as a support tool for SAMUR-PC operational planning

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA DEL SOFTWARE
CURSO 2019–2020

Manuel Monforte Escobar
Raúl Vindel Loeches

Director

Pedro Antonio González Calero
Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Madrid, Junio de 2020

Agradecimientos

Manuel Monforte Escobar

A Fernando Monforte Escobar, médico voluntario de SAMUR - Protección Civil, por servir de enlace entre SAMUR - Protección Civil y nosotros durante todo el desarrollo del proyecto.

A la Dirección General de Emergencias y Protección Civil del Ayuntamiento de Madrid, y especialmente a la Subdirección General de SAMUR - Protección Civil por su interés en el proyecto y apoyo.

A Fernando Rosa Velardo, Vicedecano de Ordenación Académica por ayudarnos con las dudas de carácter legal en relación al proyecto con SAMUR-PC.

A Pedro Antonio González Calero por su labor como profesor, guiándonos en el desarrollo durante el proyecto.

A todos aquellos profesores, compañeros del trabajo, compañeros de clase que me ayudaron y me apoyaron durante la realización del proyecto.

Raúl Vindel Loeches

A Fernando Monforte Escobar, médico voluntario de SAMUR - Protección Civil, por servir de enlace entre SAMUR - Protección Civil y nosotros durante todo el desarrollo del proyecto.

A la Dirección General de Emergencias y Protección Civil del Ayuntamiento de Madrid, y especialmente a la Subdirección General de SAMUR - Protección Civil por su interés en el proyecto y apoyo.

A Fernando Rosa Velardo, Vicedecano de Ordenación Académica por ayudarnos con las dudas de carácter legal en relación al proyecto con SAMUR-PC.

A Pedro Antonio González Calero por su labor como profesor, guiándonos en el desarrollo durante el proyecto.

A todos aquellos profesores, compañeros del trabajo, compañeros de clase que me ayudaron y me apoyaron durante la realización del proyecto.

Índice General

Índice de Figuras

Índice de Tablas

Abstract

Nowadays the speed with which an emergency can be dealt with plays a major role in the life of the victim. The need to have emergency services available and prepared for quick and efficient action has been and will always be necessary. In this paper we propose to explore a set of machine learning techniques based on the data provided by SAMUR-PC in relation to the warnings that take place in the city of Madrid between the years 2009-2019, to try to predict the number of warnings that will occur in certain circumstances.

Keywords: Machine learning, classification, linear regression, logistic regression, deep learning, neural network, CNN networks, LSTM networks, SAMUR-PC

Resumen

Hoy en día la rapidez con la que se pueda atender una emergencia juega un papel trascendental en la vida de la víctima. La necesidad de tener disponibles servicios de emergencias y preparados para una rápida y eficiente actuación ha sido y siempre será necesaria. En este trabajo se propone explorar un conjunto de técnicas de aprendizaje automático basadas en los datos concedidos por SAMUR-PC en relación a las activaciones que tienen lugar en la ciudad de Madrid entre los años 2009-2019 para intentar predecir el número de avisos que tendrá lugar en unas circunstancias determinadas.

Palabras clave: Aprendizaje automático, clasificación, regresión lineal, regresión logística, aprendizaje profundo, red neuronal, redes CNN, redes LSTM, SAMUR-PC

Capítulo 1

Introducción

1.1. Motivación

La actividad diaria de un servicio de emergencias requiere una correcta planificación del tipo, el número de recursos operativos y la cobertura de las bases, con el objetivo de ser lo más eficientes posibles. Algunas de las principales consecuencias de esta eficiencia, son una mejora en los tiempos de respuesta, así como una correcta distribución de la carga de trabajo en los profesionales e incluso, ayuda a reducir el gasto al no movilizar los diferentes dispositivos sin necesidad. En todos los ámbitos, surge la necesidad de recopilar información para poder afrontar circunstancias futuras. SAMUR - Protección Civil tiene una plataforma tecnológica en la cual se recogen todos las activaciones que recibe el servicio de emergencias, guardándolos en una Base de Datos.

1.2. Contexto

SAMUR - Protección Civil es un servicio de emergencias de reconocido prestigio internacional que actúa en la ciudad de Madrid.

SAMUR-PC surge como necesidad de crear un servicio de emergencias en la ciudad de Madrid. Partiendo del antiguo parque de ambulancias, se diseñan nuevos modelos y se actualizan las comunicaciones mejorando de forma paralela los uniformes, la selección y la formación del personal. En diciembre de 1992, tras demostrarse la utilidad del servicio se decide conceder la categoría de Trasporte Sanitario.

El objetivo de este servicio es resolver de forma rápida y eficaz las emergencias sanitarias que se produzcan en la vía pública dentro de la ciudad de Madrid. Cambiando así la perspectiva de "traslado" por "atención al paciente en el lugar del suceso y posterior traslado". En 1995, SAMUR se convirtió en SAMUR-Protección Civil, adquiriendo más responsabilidades.

El presente Trabajo Fin de Grado surge de la posibilidad de mejorar la planificación del operativo diario de SAMUR - Protección Civil a partir de las activaciones almacenadas en la Base de Datos.

1.3. Objeto de la Investigación

Como se ha comentado anteriormente, la planificación del operativo diario de un servicio de emergencias adquiere una gran importancia.

A partir de la Base de Datos de los avisos producidos entre 2009-2019, se estudiará la posibilidad de aplicar técnicas de aprendizaje automático (Machine Learning) para ayudar a conocer que días y en qué zonas de Madrid ocurrirán el mayor número de avisos y que tipo de recurso asistencial necesitarán para su resolución. Como complemento al trabajo de investigación se creara un prototipo de aplicación web para mostrar a SAMUR-PC la posibilidad de utilizar las técnicas empleadas para predecir el número de avisos en unas determinadas circunstancias.

1.4. Plan de Trabajo

El trabajo esta formado por el propio proyecto de investigación relacionado con aplicar técnicas de Aprendizaje Automático a los datos proporcionadas por SAMUR - Protección Civil. Para mostrar los resultados de las técnicas de aprendizajes seleccionadas se ha creado un prototipo de aplicación web para SAMUR-PC que se entregará en local, quedando el despliegue de la misma en sus manos, siendo el eje central del Trabajo, la aplicación de técnicas de Machine Learning sobre los datos disponibles.

A continuación se resumen el plan de trabajo con sus respectivas fases:

1. **Propuesta e Investigación del proyecto**: En abril de 2019, Fernando Monforte nos transmitió la posibilidad de explotar los datos operativos de SAMUR-PC y desarrollar una investigación que sirviera de ayuda a la planificación de los recursos asistenciales del operativo diario. La idea nos pareció interesante y una vez que tuvimos conocimiento básico de los tipos de datos que se registraban, comenzamos a trabajar sobre ella, llegando a la conclusión que podría ser útil la aplicación de técnicas de Machine Learning.

   Una vez que tuvimos claro las posibilidades que planteaba la aplicación de técnicas de Machine Learning, Fernando Monforte concertó una reunión entre nosotros y el equipo directivo de SAMUR-PC, para presentar el proyecto de investigación, sirviendo de intermediario entre nosotros y el equipo directivo de SAMUR-PC durante todo el proceso.

   En la reunión mantenida con el equipo directivo de SAMUR-PC y con Fernando Monforte, transmitimos las posibilidades que planteaba la explotación de datos mediante machine learning, y desde SAMUR-PC nos transmitieron las áreas en las que estaban más interesados, así como los detalles de posibles predicciones que serían de interés como herramienta de apoyo a la planificación operativa diaria del servicio, insistiendo en la necesidad de contar con algún parámetro que sirviera de indicador de la precisión de la predicción.

   En este contexto, y teniendo en cuenta el ámbito de conocimiento del proyecto de investigación, decidimos contactar con Pedro Antonio González Calero, profesor de la asignatura Aprendizaje Automático y Big Data de la Facultad de Informática, quien finalmente accedió a ser el tutor del proyecto.

   Durante los sucesivos meses mantuvimos varias reuniones con el equipo directivo de SAMUR-PC, para aclarar y resolver aspectos legales del proyecto de investigación,

ya que el mismo implicaba el uso de datos confidenciales de la Subdirección General de SAMUR-Protección Civil, y por tanto, requería la autorización oportuna de la Subdirección de SAMUR-Protección Civil y de la Dirección General de Emergencias y Protección Civil del Ayuntamiento de Madrid.

De forma paralela, mientras se completaban los trámites legales necesarios y hasta poder disponer de las bases de datos que se utilizarán para el desarrollo del proyecto, los alumnos investigaron sobre las principales tecnologías y frameworks para desarrollar el prototipo de aplicación web y aplicar técnicas de aprendizaje automático.

Para ello, se investigó sobre el lenguaje de programación *Python* y sus librerías *Numpy, Pandas, Keras* y *Scikit Learn.*

2. **Desarrollo de la aplicación web**: Una vez que la fase de definición del proyecto estuvo suficientemente avanzada y se adquirieron todos los conocimientos necesarios, se comenzó la segunda fase: el desarrollo del prototipo web. Durante los tres primeros meses del curso académico (octubre-diciembre), se desarrolló un prototipo para mostrar a SAMUR-PC. Tras la validación y sugerencias de SAMUR-PC se procedió a realizar el prototipo final en el mes siguiente cambiando las tecnologías utilizadas en el original. Este cambio se explicará en el correspondiente apartado con mas detalle.

3. **Aplicaciones de Aprendizaje Automático**: Esta fase se llevó a cabo en un periodo aproximado de seis meses (enero - junio).

   A) Exploración de los datos: Esta primera fase se llevó a cabo en un periodo aproximado de un mes. Comenzó con la realización de la ETL, proceso en el cual se aplicaron trasformaciones a los datos para prepararlos para su posterior representación en gráficas. Algunas trasformaciones a destacar son las siguientes:

   - Eliminar espacios en las columnas de tipo String
   - Corregir Numeración de Bases: En SAMUR-PC, existen un total de 23 Bases, numeradas de la 1 a la 23. Aquellos avisos que no son finalmente realizados se le asignan Bases atípicas con números como por ejemplo 1000, 5000. Esta trasformación consistió en trasformar todas estas Bases a -1, para poder agrupar todos los "avisos de unidades especiales o riesgo previsible" en una misma Base a la cual hacen referencia.

   Tras esta limpieza de datos se comenzó a generar gráficas y tablas para lograr una mayor comprensión de los datos para la posterior aplicación de las técnicas de aprendizaje automático.

   B) Aprendizaje automático: Esta segunda fase se llevo a cabo durante los cuatro meses siguientes a la fase anterior. Se aplicaron diferentes técnicas de Machine Learning sobre los datos con el objetivo de responder preguntas solicitadas por SAMUR-PC. Además en esta fase, para realizar una de las correspondientes fases de creación de modelos predictivos, fue necesario anonimizar los datos de SAMUR-PC para poder aplicar técnicas de aprendizaje en sistemas cloud.

4. **Resultados**: Finalmente, en la fase de resultados se procedió al análisis de todos los resultados que se habían obtenido con las numerosas pruebas realizadas y a la

extracción de conclusiones a partir de dichos resultados, incluyendo la integración de los modelos creados al prototipo web diseñado durante la segunda fase del proyecto.

1.5. Estructura del Trabajo

El resto del trabajo está organizado en 8 capítulos y 4 anexos con la estructura que se comenta a continuación:

El Capítulo 2 introduce algunos conceptos generales para conocer en qué consiste el Machine Learning. Se describe el origen del mismo y las principales técnicas utilizadas hoy en día.

El Capítulo 3, eje central del proyecto. Describe la fase de exploración de los datos. Incluye la realización de procesos ETL (Extract, Transform and Load), representación de los datos mediante gráficas para una mayor comprensión de los mismos y la posterior aplicación de técnicas de aprendizaje automático.

El Capítulo 4 presenta el prototipo realizado a SAMUR-PC con la explicación de la arquitectura de la aplicación, así como las tecnologías utilizadas.

El Capítulo 5 muestra las principales conclusiones de este trabajo y las líneas futuras de investigación.

Los Capítulos 6 y 7 son las traducciones a inglés de la Introducción y de las Conclusiones.

El Capítulo 8 contiene las contribuciones individuales de los alumnos.

1.6. Lenguajes y Tecnologías utilizadas

Durante el proyecto se ha utilizado en todo momento Python en su versión 3.7 como lenguaje principal, sobre diferentes entornos de trabajo. En cuanto a la creación de la aplicación web, se empezó utilizando PHP pero pasó a realizarse con Javascript, NodeJs y ExpressJs.

1. **Python3**

   Python es un lenguaje de programación interpretado administrado por Python Software Foundation. Su uso en aprendizaje automático está muy extendido ya que cuenta con librerías que ayudan al tratamiento (Pandas, Scikit-Learn) y visualización (Matplotlib, Seaborn) de los datos, además de otras que proporcionan algoritmos de predicción (Scikit-Learn & Tensorflow).

2. **PHP**

   Se trata de un lenguaje de programación de propósito general de código del lado del servidor. Es usado junto con HTML para desarrollar web de contenido dinámico.

3. **Javascript**

   Es un lenguaje ligero e interpretado, más conocido como el lenguaje de script para páginas web, pero también usado en muchos entornos sin navegador. Se utiliza tanto en FrontEnd para crear páginas web dinámicas como en Backend (Node.js) para implementar el lado del servidor.

4. **Jupyter Notebook**

   Jupyter Notebook es una aplicación web basada en "notebooks" formados por fragmentos de código agrupados en celdas y texto en formato Markdown, lo que permite documentar fácilmente el código.

5. **Google Cloud**

   Se trata del espacio virtual de Google en el cual se pueden realizar una serie de tareas que antes requirían hardware o software y que ahora utilizan la nube como forma de acceso. En este caso se usó para realizar entrenamientos de Machine Learning a mayor velocidad que entrenamientos de forma local.

6. **Google Collaboratory**

   Google Collaboratory, también llamado Colab, permite ejecutar y programar Python en tu navegador, accediendo de forma gratuita a GPUs, obteniendo así mayor velocidad de ejecución.

7. **NodeJs**

   Es un entorno de ejecución de JavaScript. Donde Node.js realmente brilla es en la creación de aplicaciones de red rápidas, ya que es capaz de manejar una gran cantidad de conexiones simultáneas con un alto nivel de rendimiento, lo que equivale a una alta escalabilidad.

8. **ExpressJs**

   Express.js es un framework para Node.js que sirve para ayudarnos a crear aplicaciones web en menos tiempo ya que nos proporciona funcionalidades como el enrutamiento, opciones para gestionar sesiones y cookies...

9. **PowerBI**

   Power BI es un servicio de análisis empresarial de Microsoft, su objetivo es proporcionar visualizaciones interactivas y capacidades de inteligencia empresarial con una interfaz lo suficientemente simple como para que los usuarios finales creen sus propios informes y paneles.

10. **Latex**

    Latex es un sistema de composición de textos, orientado a la creación de documentos de investigación escritos que presenten una alta calidad tipográfica. Se seleccionó como herramienta para redactar la presente memoria debido al carácter investigativo del proyecto.

10. **Docker**

    Docker es una tecnología de creación de contenedores que permite la creación y el uso de contenedores de Linux. Puede usar los contenedores como máquinas virtuales extremadamente livianas y modulares. Las principales ventajas son simplicidad y configuraciones más rápidas, no es necesario instalar en el equipo todas las tecnologías que utiliza un proyecto, con tener el proyecto en un contenedor de Docker, basta con descargar dicho contenedor, además, Docker logra reducir la implementación a segundos; esto se debe al hecho de que crea un contenedor para cada proceso y no arranca un sistema operativo.

11. **Trello**

    Trello es un software de administración de proyectos con interfaz web (también cuenta con aplicación para dispositivos móviles). Permite organizar tableros con distintas listas y tareas, asignar tareas a distintas personas, establecer alarmas o fechas junto con las tareas, etc. A continuación adjuntamos una captura del panel de tareas utilizado en el proyecto:

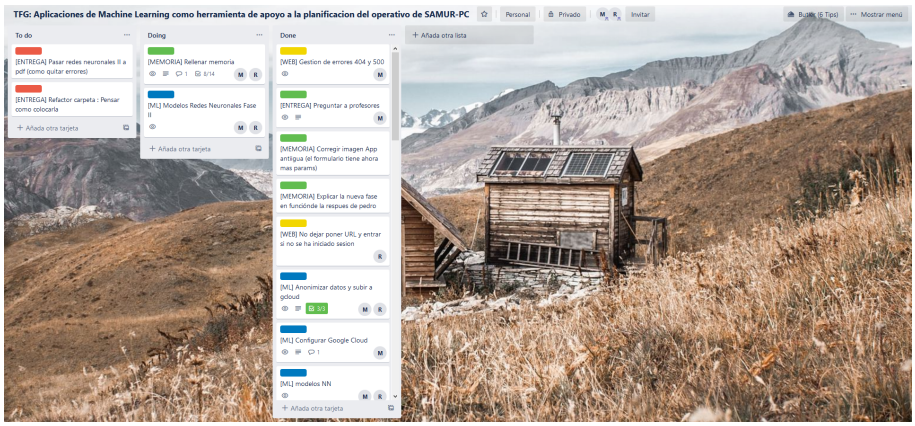


Figura 1.1: Panel de tareas

1.7. Repositorio

Este proyecto se encuentra publicado en un repositorio de GitHub, dado que el repositorio no puede estar en público por motivos de confidencialidad, se ha creado una carpeta en Google Drive con el contenido del mismo . El repositorio se divide en tres partes:

- **App:** Directorio que contiene el prototipo web final realizado durante el proyecto. En el se encuentra la configuración de Docker para ejecutar el prototipo de forma sencilla.
- **Prototipo:** Directorio que contiene el código desarrollado como la primera versión del prototipo web mostrada a SAMUR - Protección Civil
- **Documentación:** Directorio que contiene todos los archivos de documentación realizados durante el proyecto, incluye:
  - Manual de usuario para ejecutar aplicación web
  - Documentos en pdf de los jupyter notebooks creados durante la realización del proyecto.
- **demo.ekv:** Contiene un vídeo mostrando la página desarrollada en caso de no ser posible realizar su puesta en marcha.

1.8. Asignaturas de la carrera aplicadas

Durante la carrera se han cursado distintas asignaturas que han ayudado a adquirir conocimientos transversales con los cuales hemos podido realizar este proyecto.

1. **Aprendizaje automático & Big Data (AA)**

   Es la asignatura a través de la cual se han adquirido los conocimientos necesarios para realizar las actividades relacionadas con Machine Learning y manejo de gran volúmenes de datos.

2. **Aplicaciones Web (AW)**

   Gracias a la cual se han podido aplicar conocimientos de desarrollo web basados en HTML, PHP, NodeJs y ExpressJs para realizar el prototipo web que soporta los modelos de Aprendizaje Automático.

3. **Bases de Datos (BD)**

   Asignatura que nos ha permitido conocer conceptos como Bases de Datos relacionales utilizadas hoy en día a menudo en el mundo del desarrollo de Software. Nos ha permitido gestionar la autenticación de usuarios en el prototipo creado para SAMUR - Protección Civil.

4. **Tecnología de la programación (TP)**

   Asignatura en la cual hemos podido aprender el paradigma de programación orientado a objetos, que unida a Ingeniería del Software, Modelado del Software y Aplicaciones Web, nos han permitido desarrollar el prototipo para SAMUR-PC.

5. **Ingeniería del Software y Modelado de Software (IS & MS)**

   En nuestra opinión dos de las asignaturas más importantes de la carrera que nos han ayudado a comprender la importancia de los patrones de diseño en el desarrollo de Software y aplicarlos al prototipo utilizando patrones como Controller o DAO (Data Access Object).

6. **Prácticas en empresa**

   Gracias a esta asignatura optativa hemos podido desarrollar nuestra experiencia laboral y conocer más herramientas utilizadas en el mundo del Big Data, particularizando en el proyecto, además conocimos y aprendimos la creación de informes interactivos con PowerBI.

7. **Estadística Aplicada (EA)**

   Con esta asignatura hemos adquirido conocimientos básicos sobre estadística que nos han ayudado a entender de mejor forma las técnicas de aprendizaje automático explicadas en esta memoria.

8. **Gestión de Proyectos Software (GPS)**

   En esta asignatura hemos podido comprender la gestión de los proyectos en el mundo del software y aplicarlo al trabajo, creando un panel de tareas, siguiendo modelos de tableros característicos de metodologías ágiles, en los cuales las tareas tienen tres estados (To Do, Doing, Done).

9. **Desarrollo de Sistemas Interactivos (DSI)**

   Asignatura que ha ayudado a la hora de desarrollar la aplicación web de forma interactiva y de una forma más atractiva y usable de cara al usuario.

Capítulo 2

Estado del Arte

El objetivo de este capítulo es explicar diferentes conceptos generales relacionados con el origen del Machine Learning y las principales técnicas utilizadas en él.

2.1. Historia del Machine Learning

2.1.1. Origen

Su origen data de 1943, cuando el neurólogo Warren McCulloch y el matemático Walter Pitts crearon el primer modelo de una red neuronal usando un circuito eléctrico a partir de conocimientos suyos sobre neuronas y su funcionamiento.

Sin embargo, no fue hasta 1952 cuando se vió el primer programa que era capaz de aprender por sí mismo. Era un programa que jugaba a las damas creado por Arthur Samuel.

Después, en 1958, Frank Rosenblatt diseñó la primera red neuronal artificial, cuyo objetivo era reconocer patrones y formas. En 1959, Bernard Widrow y Marcian Hoff crearon dos modelos, uno que era capaz de detectar patrones binarios y predecir cuál sería el siguiente bit; y otro que era capaz de eliminar el eco en las líneas telefónicas.

A partir de ahí, el ámbito del Machine Learning no progresó mucho, en gran parte debido a la popularidad de la arquitectura Von Neumann, la cual, al ser más simple, hizo que se escribieran más programas basados en ella.

2.1.2. Evolución

En los años 80 y 90 volvió a crecer el interés sobre el machine learning. Entre otras cosas, esto se debe al interés de Japón y Estados Unidos. Primero Japón anunció que se estaba centrando en redes neuronales más avanzadas, y para no quedarse atrás, Estados Unidos investigó más en profundidad sobre el tema.

En 1986 se dió un paso importante, y es el hecho de que las redes neuronales usaran propagación hacia atrás. Esto permitía usar varias capas en una red neuronal, creando lo que conocemos como 'slow learners' (aprendiz o alumno lento), es decir, que está aprendiendo durante un largo periodo de tiempo.

Lo más destacado del final de los años 90 fue cuando, en 1997, el IBM computer Deep Blue venció al campeón del mundo de ajedrez y cuando en 1998, tras un estudio sobre reconocimiento de dígitos, se consiguió detectar códigos postales escritos a mano.

Ya en el siglo XXI, las empresas se han dado cuenta de que el Machine Learning aumentará el potencial de cálculo. Algunos proyectos actuales que usan Machine Learning son:

- **Google Brain:** Era una red neuronal centrada en detectar patrones en fotos y vídeos. Después se utilizó para detectar objetos en los vídeos de Youtube.
- **Amazon Machine Learning Platform:** Es parte de Amazon Web Services, y actúa sobre varios de sus sistemas internos, como por ejemplo recomendaciones en las búsquedas de los usuarios.
- **DeepMind:** Compañia comprada por google que puede jugar videojuegos con la misma fluidez que los humanos.

2.1.3. ¿Qué es?

Es una rama de la IA (Inteligencia Artificial). La IA es la disciplina cuyo principal objetivo es dotar a una máquina de la inteligencia similar a un ser humano.

El Machine Learning, es en concreto, la aplicación práctica de modelos matemáticos para dotar a un ordenador de IA, que aprende de forma automática a partir de los datos históricos.

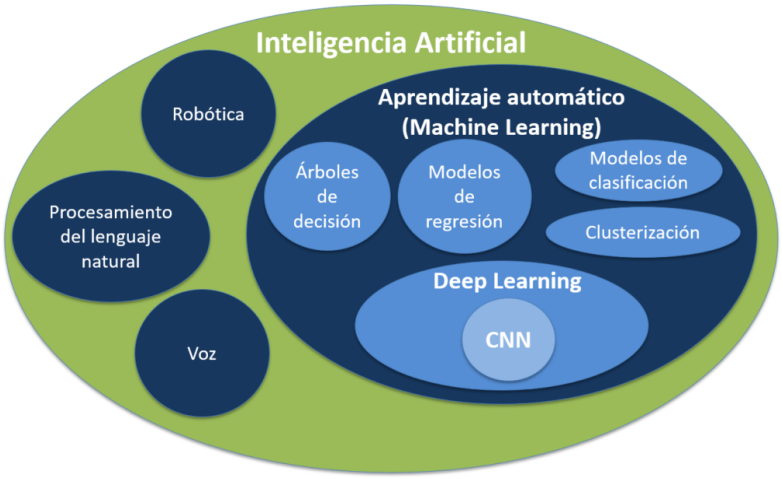


Figura 2.1: Machine Learning e IA

2.2. Aplicaciones del Machine Learning

Dentro de las técnicas de Aprendizaje automático, existen dos tipos de técnicas claramente diferenciadas:

- **Técnicas supervisadas:** son aquellas técnicas en las cuales tenemos la variable objetivo o que queremos predecir dentro de los datos de entrenamiento. Si la variable objetivo se trata de una variable continua la técnica a aplicar se denomina regresión. Si por el contrario, se trata de una variable categórica, la técnica a aplicar será una técnica de clasificación.

- **Técnicas no supervisadas** son aquellas técnicas en las cuales la variable objetivo no se encuentra dentro de los datos de entrenamiento. Si la variable objetivo se trata de una variable continua la técnica a aplicar será reducción de dimensionalidad, por el contrario, si se trata de una variable categórica, la técnica a aplicar será Clustering (agrupamiento).

2.2.1. Regresión Lineal

Como hemos comentado anteriormente, se trata de una técnica de aprendizaje supervisada, en la cual, la variable objetivo es una variable continua.

Una de las primeras regresiones realizadas fue en 1886 por Francias Galton [1]. Recogió datos sobre la altura de padres e hijos de diferentes familias, colocando en el eje de ordenadas la altura de 900 padres de familia y en el eje de abscisas 900 hijos. Observó la existencia de una relación lineal entre la altura de padres y de hijos. A continuación se muestra una gráfica realizada por Francis Gilton, y a su derecha una gráfica más actualizada del mismo estudio:

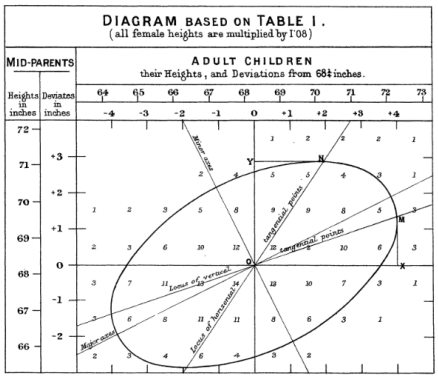


Figura 2.2: Regresión Francis Gilton

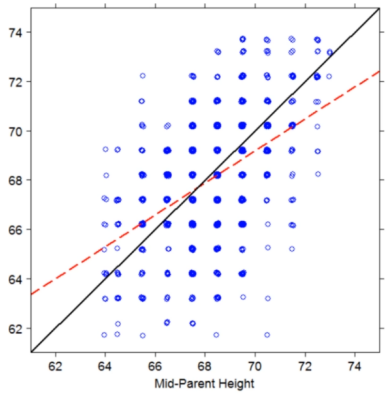


Figura 2.3: Regresión F.Gilton Actualizada

Francias Galton observó que la altura de los hijos era un poco menor que la de sus padres, regresando a la media de la población, de ahí el término "regression" en inglés.

En Regresión lineal, el objetivo es generar un modelo de la siguiente forma:

$$h_\Theta(x) = \Theta_0 + \Theta_1 x_1 + ... + \Theta_n x_n$$

Donde la variable objetivo se consigue como la combinación lineal de variables independientes, y donde x_i es la entrada i-ésima, y Θ_i valores para los cuales la función de coste es mínima.

Para que el modelo sea lo más preciso posible, se busca minimizar al máximo la función de coste. En Regresión Lineal, la función de coste es de la forma:

$$J(\Theta_0, \Theta_1, ..., \Theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\Theta(x^{(i)}) - y^{(i)})$$

Para minimizar este coste se realiza el algoritmo de "Descenso de Gradiente". Este empieza con valores de Θ arbitrarios, los cuales se irán modificando siguiendo el siguiente algoritmo:

$$\Theta_j := \Theta_j - \alpha \frac{\partial}{\partial \Theta_j} J(\Theta_0, ..., \Theta_n)$$

Para evaluar el modelo de regresión existen diferentes medidas frecuentemente utilizadas:

- **Error absoluto medio:** Es la media de las diferencias entre la variable objetivo y las precisiones sin el signo. El error se puede interpretar en unidades de la variable objetivo, particularizando en este proyecto, será el nº de avisos.

$$\text{MAE} = \frac{1}{n} \sum_{t=1}^{n} |e_t|$$

- **Error Cuadrático Medio:** Es menos robusto ya que al elevar los errores al cuadrado, suele penalizar los errores más grandes, por lo que se realizará la raíz del resultado.

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{t=1}^{n} e_t^2}$$

- **Coeficiente de determinación:** mide la porción de la varianza de la variable objetivo que se puede explicar por el modelo. El máximo valor que puede alcanzar es uno, pudiendo tener valores negativos. Un problema importante es que no nos indica si el modelo explica las predicciones debido a que está sobre-ajustado, por ello se utilza el coeficiente de determinación ajustado.

Nuestros modelos de regresión lineal se medirán por el RMSE y MAE para determinar entre que valores pueden oscilar el número de avisos.

2.2.2. Regresión Logística

Se trata de un tipo de regresión utilizada para predecir el resultado de una variable categórica, es decir, una variable que solo puede tomar ciertos valores predefinidos en función de las variables independientes. Por el hecho de predecir una variable categórica, no es una regresión utilizada en nuestro estudio.

El modelo objetivo proporcionará un valor entre 0 y 1, en cuyo caso, se predecirá como 1 si $h_\Theta \geqslant 0,5$, o se predecirá como 0 si $h_\Theta < 0,5$ donde:

$$h_\Theta(x) = \frac{1}{1+e^{-\Theta^T x}}$$

Al igual que en Regresión Lineal, se tiene un coste el cual se quiere minimizar y un algoritmo de Descenso de Gradiente para minimizarlo. La función de coste viene dada por la función:

$$J(\Theta) = \frac{-1}{m}[\sum_{i=1}^{m} y^{(i)} log(h_\theta(x^{(i)})) + (1 - y^{(i)}) log(1 - h_\theta(x^{(i)}))]$$

El algoritmo de Descenso de Gradiente viene dado por la función:

$$\Theta_j := \Theta_j - \alpha \sum_{i=1}^{m} (h_\Theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

2.2.3. Árboles de Clasificación

Es un técnica de aprendizaje automático bastante común utilizada para la toma de decisiones. Para generar árboles de decisión se utilizan diversos algoritmos inductivos, es decir que aprenden patrones a bases de ejemplos. El más utilizado es el algoritmo de Hunt, que se caracteriza por ser un algoritmo recursivo, oportunista (en cada paso toma la decisión mas favorable sin pensar en pasos anteriores o futuros) y mínimo local (llega a una solución óptima pero no garantiza que sea óptima global). El algoritmo es el siguiente, para un nodo T que tiene un conjunto de elementos D(t):

1. Si todos los elementos D(t) pertenecen a la misma clase, el nodo T es una hoja de esa clase

2. Si no lo son, se aplica un criterio de partición dividiendo D(t) en subgrupos. Para cada salida del criterio se crea un nodo hijo de T y los elementos D(t) se reparten entre los nodos hijos.

3. Se repite el proceso para cada nodo hijo.

A continuación se muestra un ejemplo de un árbol de decisión generado, este en concreto, en función de las características de la muestra como la longitud y anchura del pétalo, es capaz de predecir que tipo de flor iris se trata:

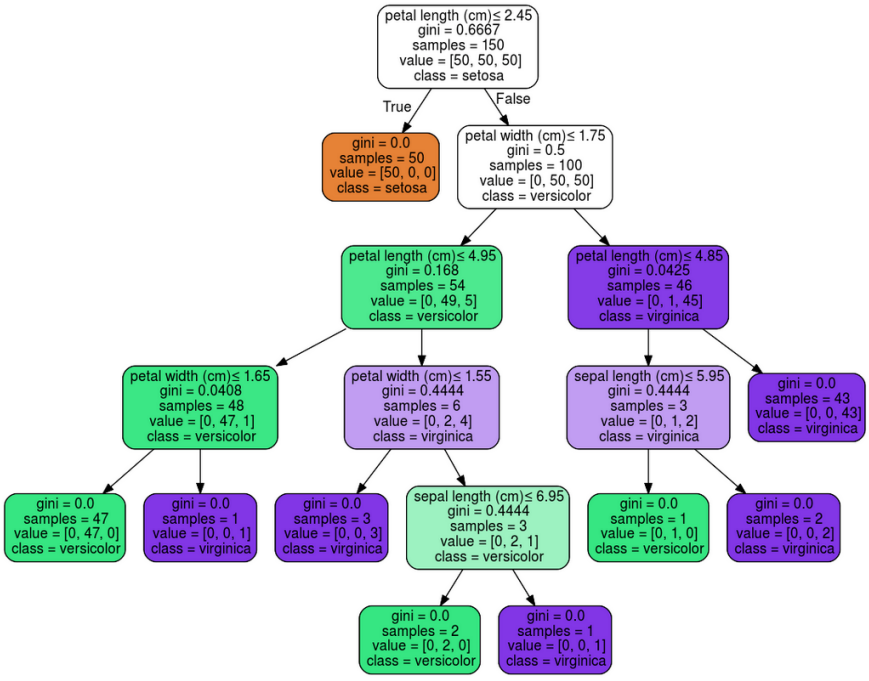


Figura 2.4: Árbol de decisión

2.2.4. Agrupamiento o Clustering

Es una técnica de aprendizaje automático no supervisada que busca como output una variable categórica (una clase o etiqueta), al ser no supervisada, no tenemos la variable objetivo en nuestros datos de entrenamiento. Los principales usos de los algoritmos de clustering son:

- Segmentar un dataset para comprender sus características intrínsecas y encontrar grupos de elementos similares.
- Detectar elementos anómalos, es decir, elementos que no pertenecen a ninguna clase
- Simplificar datasets al agrupar conjuntos de variables con valores similares

El algoritmo más utilizado es el K-Medias (K-Means). Para explicar el algoritmo vamos a explicarlo con un ejemplo. Suponemos que somos un supermercado y queremos conocer grupos de clientes para diseñar campañas de marketing. Tras realizar un muestreo obtenemos datos como la edad del cliente y el valor medio del carrito de la compra. A continuación observamos una gráfica de estos datos:

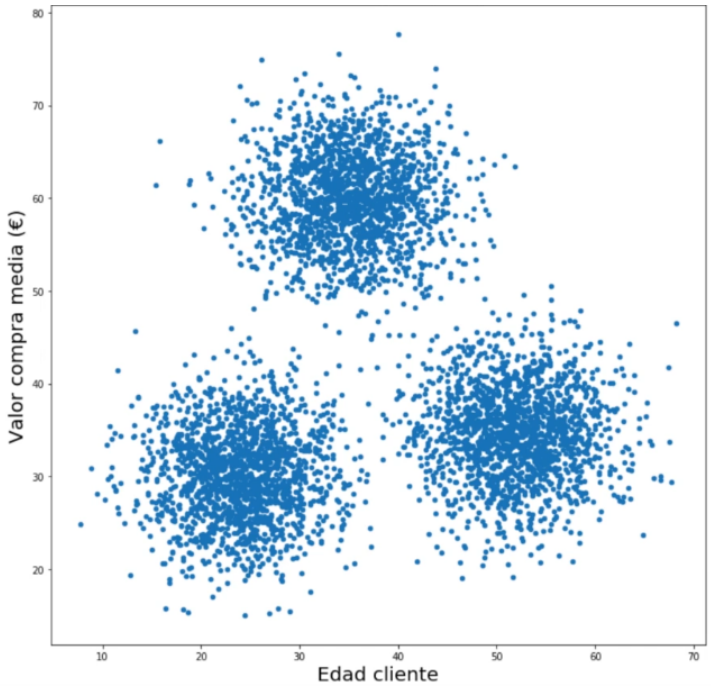


Figura 2.5: Ejemplo K-Means

Los pasos a realizar por el algoritmo son los siguientes:

1. **Paso nº1:** Elegir k, dónde k es un hiper parámetro que nos indica el número de clusters o grupos que queremos obtener, en este caso 3.
2. **Paso nº2:** Asignamos k-centroides al azar, dónde los centroides son los puntos que están en el centro de cada clúster o grupo.
3. **Paso nº3:** Asignamos cada observación al clúster más cercano. Para ello nos guiamos por la distancia euclidiana.

4. **Paso nº4:** Calculamos la inercia. Al tratarse de un algoritmo recursivo debemos especificar una condición de parada para que no siga de forma indefinida, esta condición de parada es minimizar la inercia (se calcula como el sumatorio al cuadrado de las distancias de cada observación al centroide más cercano (C)), de esta forma estaremos colocando los centroides poco a poco en el lugar correcto.

$$I = \sum_{i=0}^{n} min(||x_j - \mu_i||^2), \mu_j$$

5. **Paso nº5:** Calculamos los nuevos centroides. Tras esto volvemos al paso nº3, y realizamos este proceso recursivo hasta que conseguimos la condición de parada.

A continuación se muestras capturas para visualizar el proceso:

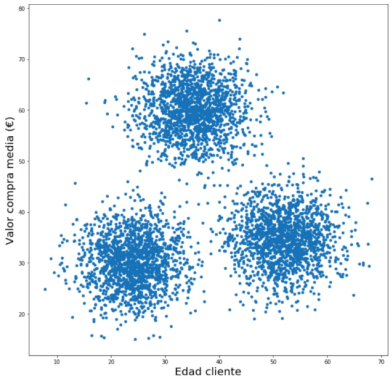


Figura 2.6: K-Means Paso nº1

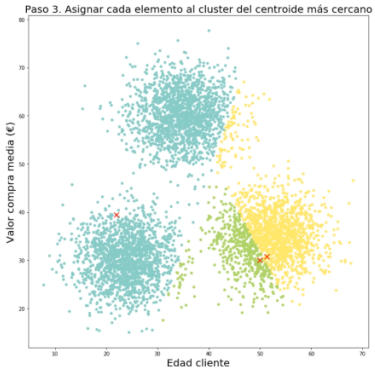


Figura 2.7: K-Means Paso nº3

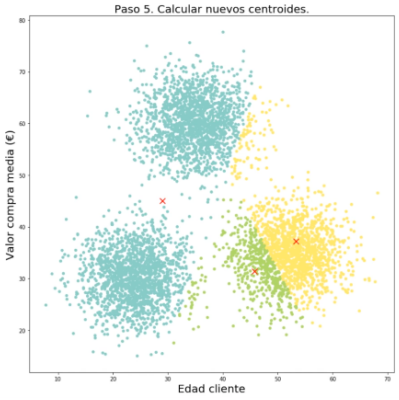


Figura 2.8: K-Means Paso nº5

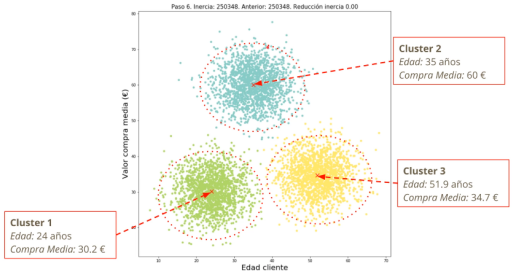


Figura 2.9: K-Means Conclusión

2.2.5. Máquinas de vector soporte (SVM)

Se trata de un tipo de problemas de clasificación de distintas clases, en el que la idea principal es encontrar un plano que separe estas. Puede tratarse de un problema bidimensional o n-dimensional, en cuyo caso se hacen transformaciones para acabar representando los datos y el plano de forma bidimensional.

En concreto se busca un hiperplano de máximo margen (MMH), el cual es un plano n-dimensional que separa ambas categorías de la forma más óptima, es decir, encontrando el plano con mayor margen entre los puntos más cercanos de cada clase. De este modo, el margen está determinado por la distancia entre los puntos más cercanos de las distintas clases, y estos puntos son los que componen los "Vectores Soporte"(vectores que soportan o apoyan la decisión de escoger ese plano y margen).

Un hiperplano se define matemáticamente de la siguiente manera:

$$\omega^T x + b = 0$$

Donde ω^T es el conjunto de pesos del plano (un elemento por dimensión), x es la entrada (las variables independientes) y b es el sesgo o "bias", el cual representa la distancia al origen.

Se puede dar el caso, como se aprecia en la siguiente imagen, en el que haya datos de nuestras clases que se salgan de su comportamiento normal, es decir, que dificulten la separación por un plano o incluso estén en el lado contrario a todos los demás componentes de su clase.

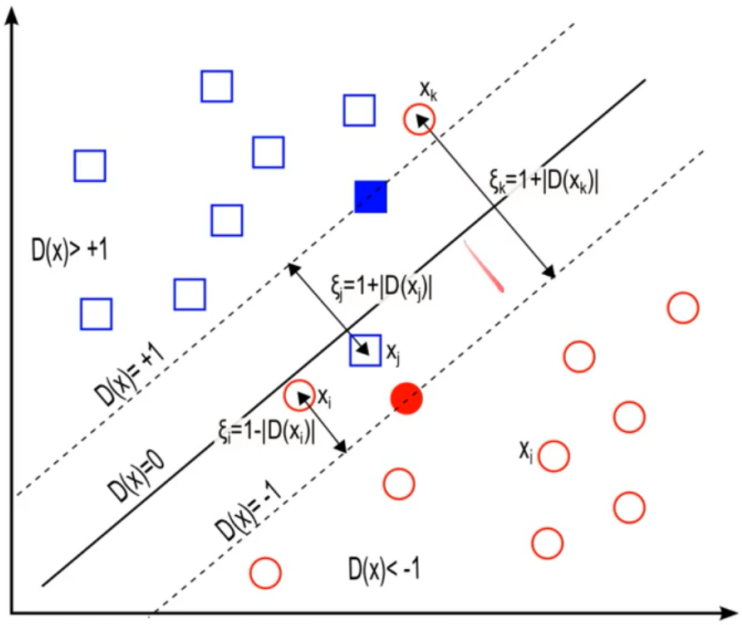


Figura 2.10: Ejemplo Variables de Holgura

En estos casos se utilizan las denominadas Variables de Holgura, las cuales permiten, a costa de clasificar algún componente incorrectamente, tener un mejor margen.

2.2.6. Redes Neuronales

Las redes neuronales artificiales (ANN) son redes neuronales multicapa totalmente conectadas. Consisten en una capa de entrada, múltiples capas ocultas y una capa de salida. Cada nodo de una capa está conectado a todos los demás nodos de la siguiente capa. Hacemos la red más profunda aumentando el número de capas ocultas.

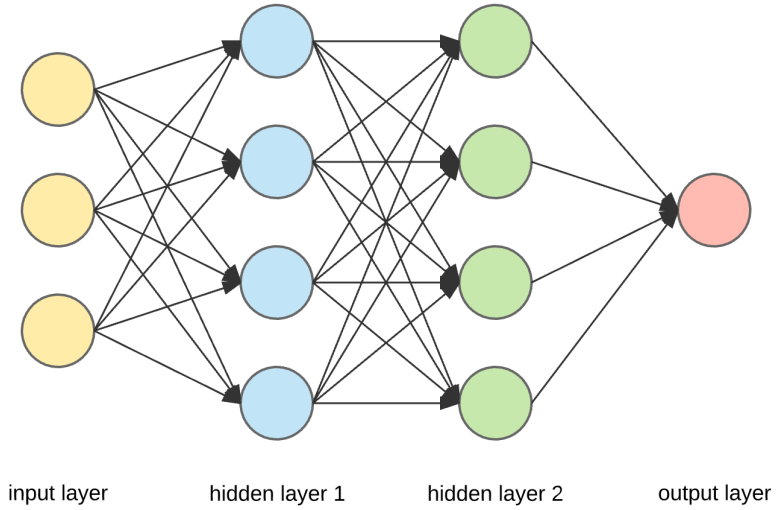


Figura 2.11: Ejemplo ANN

Si nos fijamos en los nodos de las capas ocultas o de la capa final encontramos esta estructura:

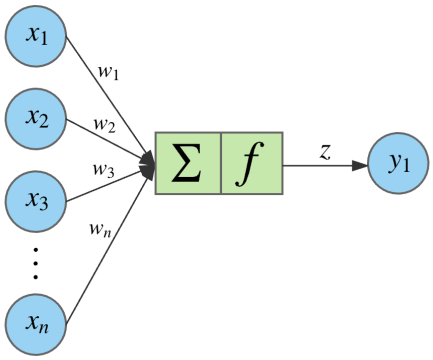


Figura 2.12: Nodo ANN

Un nodo dado toma la suma ponderada de sus entradas y la pasa a través de una función de activación no lineal. Esta es la salida del nodo, que luego se convierte en la entrada de otro nodo en la siguiente capa. La señal fluye de izquierda a derecha, y la salida final se calcula realizando este procedimiento para todos los nodos. Entrenar esta red neuronal profunda significa aprender los pesos asociados a todos los bordes.

La ecuación para un nodo dado es la siguiente: La suma ponderada de sus entradas pasó a través de una función de activación no lineal. Se puede representar como un producto de punto vectorial, donde n es el número de entradas para el nodo:

$$z = f(x \cdot w) = f\left(\sum_{i=1}^{n} x_i w_i\right)$$

$$x \in d_{1\times n},\ w \in d_{n\times 1},\ z \in d_{1\times 1}$$

Figura 2.13: Función Nodo ANN

Hasta ahora hemos descrito el camino hacia adelante, es decir, dada una entrada y pesos cómo se calcula la salida. Después de completar el entrenamiento, sólo hacemos el camino hacia adelante para hacer las predicciones. Pero primero necesitamos entrenar nuestro modelo para aprender los pesos, y el procedimiento de entrenamiento funciona de la siguiente manera:

1. Inicializar al azar los pesos de todos los nodos.

2. Para cada ejemplo de entrenamiento, hacer el camino hacia adelante una vez usando los pesos actuales, y calcular la salida de cada nodo yendo de izquierda a derecha. El resultado final es el valor del último nodo.

3. Comparar el resultado final con el objetivo real de los datos de entrenamiento y medir el error utilizando una función de pérdida (para nuestro estudio, utilizaremos a función MSE con el objetivo de obtener la menor pérdida consecuentemente en el RMSE final).

4. Realizar un camino hacia atrás de derecha a izquierda y propagar el error a cada nodo individual usando la retro-propagación. Calcular la contribución de cada peso

al error y ajustar los pesos en consecuencia utilizando el descenso de gradiente. Tras ello, propagar los gradientes de error hacia atrás a partir de la última capa. La retro-propagación es la clave que hay detrás del aprendizaje profundo (Deep Learning)

La propagación hacia detrás, mejora el modelo en función del error de predicción usando un método de optimización. Esta formado por tres pasos:

1. Cálculo del error en la predicción: Hay que definir una función de error, para problemas de regresión, se utiliza generalmente el error cuadrático, para problemas de clasificación binaria se utiliza la pérdida logarítmica.

   - Regresión: $J(\theta, x^i, y^i) = \frac{1}{2}(h_\theta(x^i) - y^i)^2$
   - Clasificación: $\delta_1^2 = -(y^i log(h_\theta(x^i) + (1 - y^i) log(1 - h_\theta(x^i)$

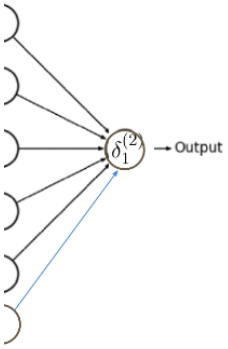


Figura 2.14: Cálculo del error

2. Propagamos el error a la capa oculta, para ello utilizamos parte del método del descenso de gradiente:

$$\begin{aligned} \delta_1^1 &= w_1 * \delta_1^2 * J'(a_{11}) \\ \delta_2^1 &= w_2 * \delta_1^2 * J'(a_{12}) \\ &\dots \\ \delta_n^1 &= w_n * \delta_1^2 * J'(a_{1n}) \end{aligned}$$

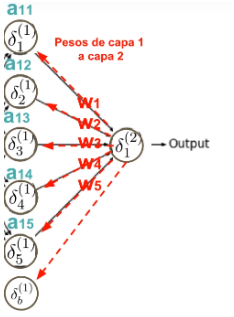


Figura 2.15: Propagación error capa oculta

3. Propagar el error a la capa de entrada:

$$\delta_1^0 = (w_{11} * \delta_1^1 + ... + w_{n1} * \delta_n^1) * J'(x_{11})$$
$$\delta_2^0 = (w_{12} * \delta_1^1 + ... + w_{n2} * \delta_n^1) * J'(x_{12})$$
$$...$$
$$\delta_m^0 = (w_{1m} * \delta_1^1 + ... + w_{nm} * \delta_n^1) * J'(x_{1m})$$

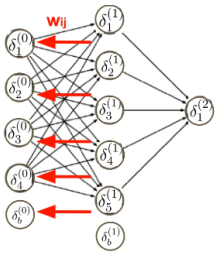


Figura 2.16: Propagación error capa entrada

A continuación se describen las principales redes ANN:

- **Convolutional neural network (CNN):**

  Este tipo de redes neuronales está formado por neuronas de una forma muy similar a las neuronas de la corteza visual primario de un cerebro. Su fuerte son los casos con datos distribuidos de forma continua, por lo que son el modelo más utilizado en cuanto a problemas relacionados con imágenes/vídeos y son muy utilizadas en el campo de la visión artificial. También son útiles para sistemas de recomendación.

  La principal ventaja de este tipo de modelos es su capacidad de detectar automáticamente las características importantes sin necesidad de la supervisión de un humano, de hecho, actualmente realizan clasificación de imágenes mejor que los humanos. La principal desventaja es capturar relaciones espaciales.

  En cuanto a su arquitectura, están formadas por varias capas de filtros convolucionales de una o varias dimensiones. En estos casos, la convolución (operación matemática que mezcla dos conjuntos de datos) es aplicada a los datos de entrada.

  A la hora de entrenar este tipo de modelos, se entrenan de la misma forma que los modelos ANN, es decir, mediante propagación hacia atrás con descenso de gradiente.

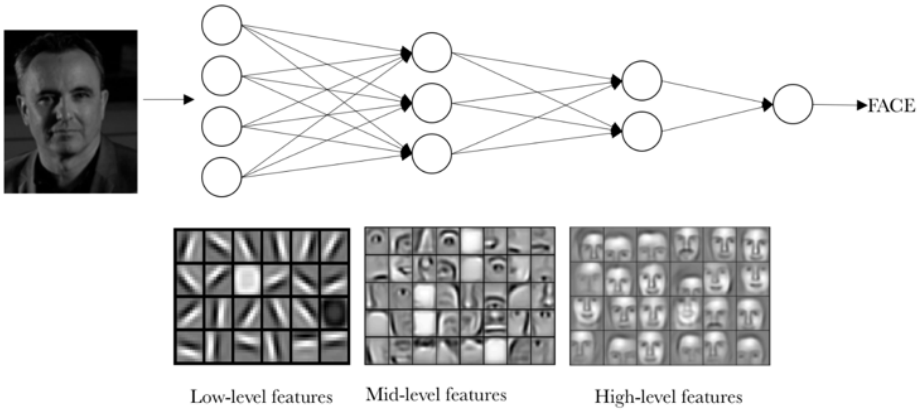


Figura 2.17: Ejemplo evolución CNN

- **Long short-term memory (LSTM):** Como hemos visto en las CNN, existen problemas de capturar relaciones espaciales. Además, las capas Densas tratan cada input de manera independiente, cuando en los problemas del mundo real, los datos suelen estar generalmente relacionados de forma secuencial. Existen un tipo de capas que forman las denominadas Redes Neuronales Recurrentes (RNN) que buscan resolver estos problemas.

  En las RNN, el output de una iteración se considera como input en la siguiente iteración. Se retroalimentan de lo que han producido en el pasado. Una desventaja de utilizar capas densas de forma recurrente es que los gradientes tienden a desaparecer (convertir los pesos en 0), para evitar esto, hay versiones mejoradas de las capas recurrentes como son las capas LSTM.

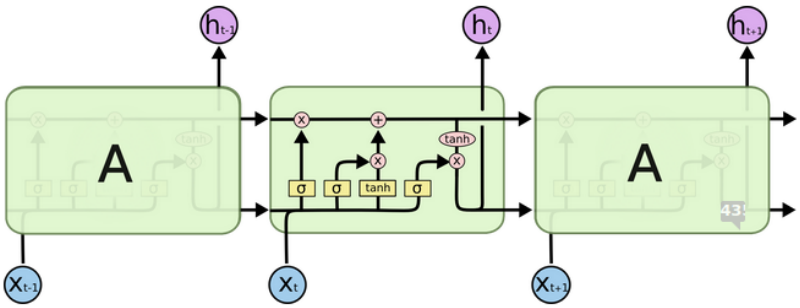


Figura 2.18: Estructura Celda LSTM

Las capas LSTM tienen estado y capacidad para modificar dicho estado, quitando información que ya no es interesante y añadiendo aquella que sí lo es. Para conseguir esta modificación del estado se ayuda de las denominadas puertas (gates). Estas puertas se componen de una capa propia con su función de activación y un producto escalar.

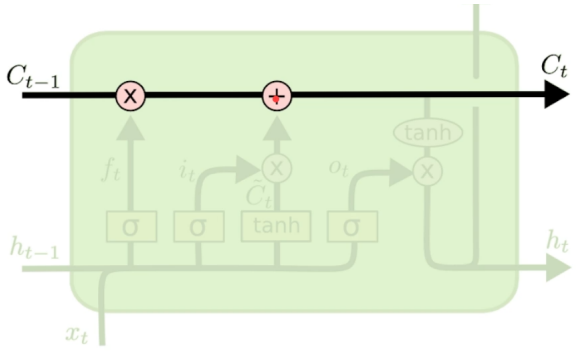
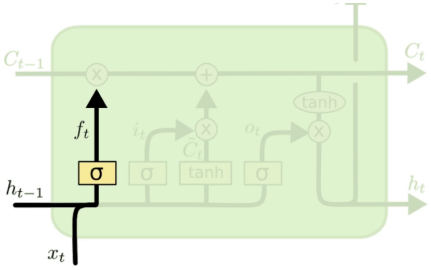


Figura 2.19: Estado Celda LSTM

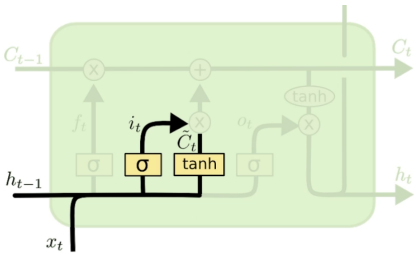


Puerta de olvido (forget gate)

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figura 2.20: Capa olvido celda LSTMM

En primer lugar se aplica la capa de olvido, que se encarga de borrar la información que ya no interesa .



Puerta de entrada (input gate)

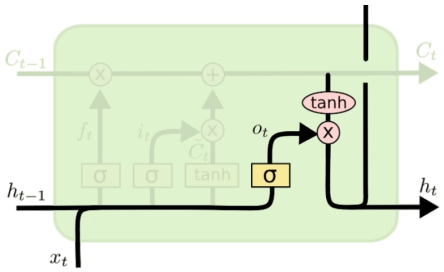
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figura 2.21: Capa entrada celda LSTM

En segundo lugar, la puerta de entrada decide que información se va añadir nueva. Para ello encontramos una capa de activación sigmoide que decide qué valores hay que actualizar y por otro lado una función de activación tangente hiperbólica que decide cómo se modifican esos valores. Finalmente se combinan las dos anteriores y se suman al estado actual.

Con la capa de olvido y de entrada ya tenemos el estado actualizado. Con ese estado nuevo, la puerta de salida produce la activación y obtenemos la salida.



Puerta de salida (output gate)

$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(\dot{C}_t\right)$$

Figura 2.22: Capa salida celda LSTM

Capítulo 3

Aprendizaje Automático sobre los datos

En este apartado explicaremos en profundidad las características de cada una de las partes de las que se compone el trabajo que hemos realizado: el modelo propiamente dicho y la colección de datos (o dataset).

3.1. Fase de Exploración

3.1.1. Procesos de Trasformación de los datos

En este apartado se resumen las principales trasformaciones aplicadas a los datos de SAMUR-PC, con el objetivo de preparar los datos para sus posteriores etapas de visualización y aplicación de técnicas de aprendizaje automático:

- **Unir hojas de archivos xlsx SAMUR-PC:** Los datos recogidos comprenden los años 2009-2019 (octubre). Cada año se encuentra en un fichero formato xlsx. Dentro de ese fichero hay tres hojas cada una con unos 50.000 avisos. El primer proceso de trasformación consiste en colocar las tres hojas de 50.000 avisos en una única de 150.000.
- **Crear un único xlsx:** El objetivo de este proceso de trasformación es unir todos los años en un mismo fichero xlsx, el cual utilizará powerBI para crear el informe que permitirá visualizar los datos históricos que se utilizará en el prototipo para mostrar a SAMUR-PC la posibilidad de predecir pero también de visualizar datos del pasado.
- **Eliminar espacios columna DIAS SEMANA:** Este proceso consiste en eliminar de la columna "DIAS SEMANA"los espacios, ya que en la BD se reservan 9 espacios, VARCHAR(9), por tanto el lunes tiene valor "Lunes ", "Martes ", etc...
- **Crear nueva columna para corregir bases:** Este proceso consiste en crear una nueva columna. SAMUR-PC tiene 24 bases (1-23), algunos avisos están asignados a bases atípicas como 5000, esto sirve para marcar aquellos avisos de unidades especiales o de riesgo previsible, por lo que se procede a crear una columna nueva que contendrá el número de la base, en caso de ser valor atípico, a todos se le asignará un -1.

3.1.2. Gráficas

En esta fase se realizan gráficas relativas a las preguntas que desea resolver SAMUR-PC entorno a los datos. Las predicciones que desea conocer son:

3.1.2.1. Número de avisos para una fecha determinada

Para explorar esta posible opción se han generado los siguientes gráficos.

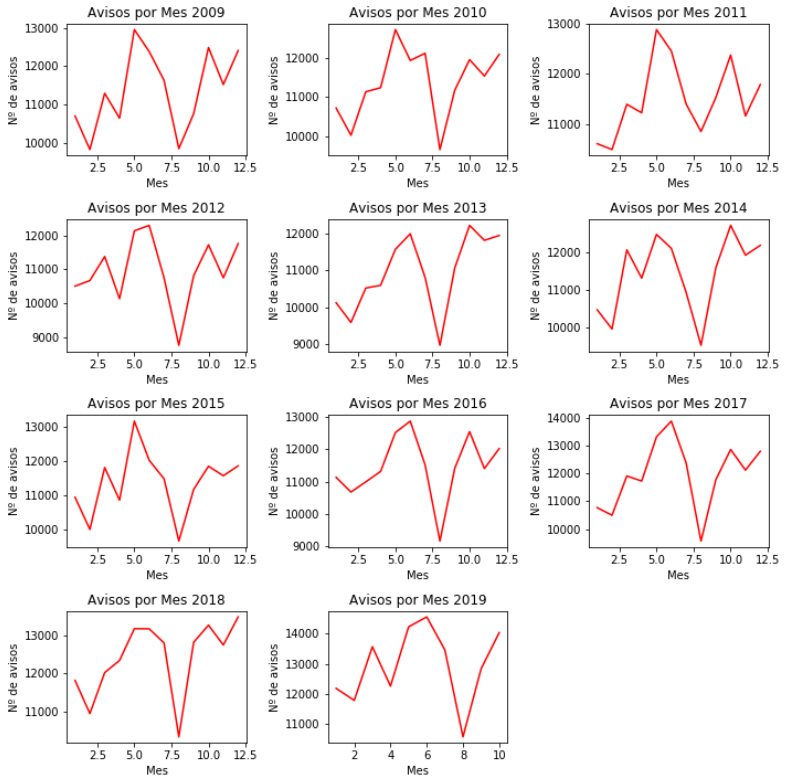


Figura 3.1: Nº de Avisos por año y meses

Como podemos comprobar en las gráficas anteriores, los meses siguen el siguiente patrón. El mes con el menor número de avisos durante todos los años, es el mes de agosto, mientras que los meses en los que siempre ocurren el mayor número de avisos es mayo, junio y octubre. A continuación se muestra una gráfica donde se acumulan todos los avisos por meses entre los diferentes años, de esta forma podemos conocer en que meses se acumulan en mayor medida los avisos de SAMUR-PC.

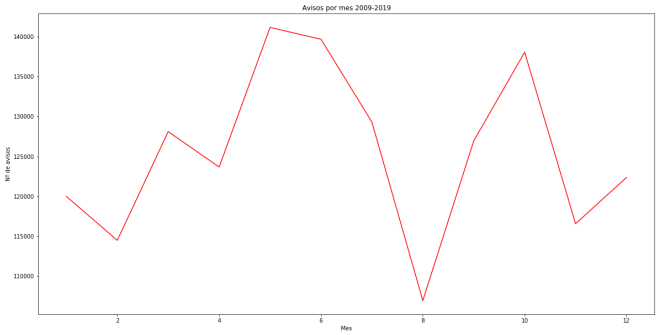


Figura 3.2: Nº de Avisos por meses

3.1.2.2. Número de avisos por dispositivo

Para explorar esta posible opción se han generado los siguientes gráficos:

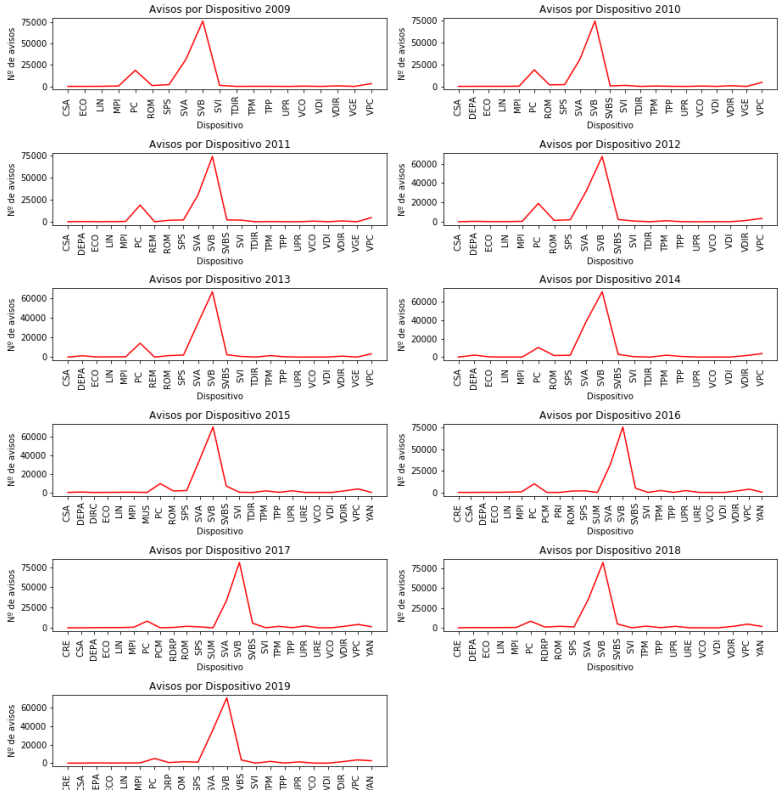


Figura 3.3: Nº de Avisos por año y dispositivo utilizado

Como podemos comprobar la mayor parte de los avisos se destinan las siguientes unidades:

- **SVB:** Soporte Vital Básico
- **SVA:** Soporte Vital Avanzado

También destaca, en menor mediadas unidades de Protección Civil (PC).

Si mostramos está gráfica agrupando todos los años 2009-2019, podemos observar lo comentado anteriormente con mayor claridad:

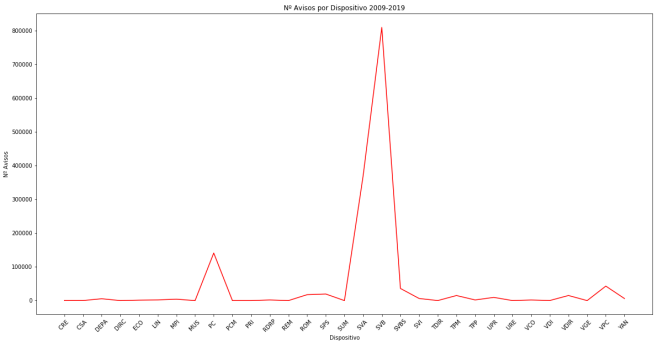


Figura 3.4: Nº de Avisos por dispositivo utilizado

3.1.2.3. Número de avisos por distrito

Para explorar esta posible opción se han generado los siguientes gráficos.

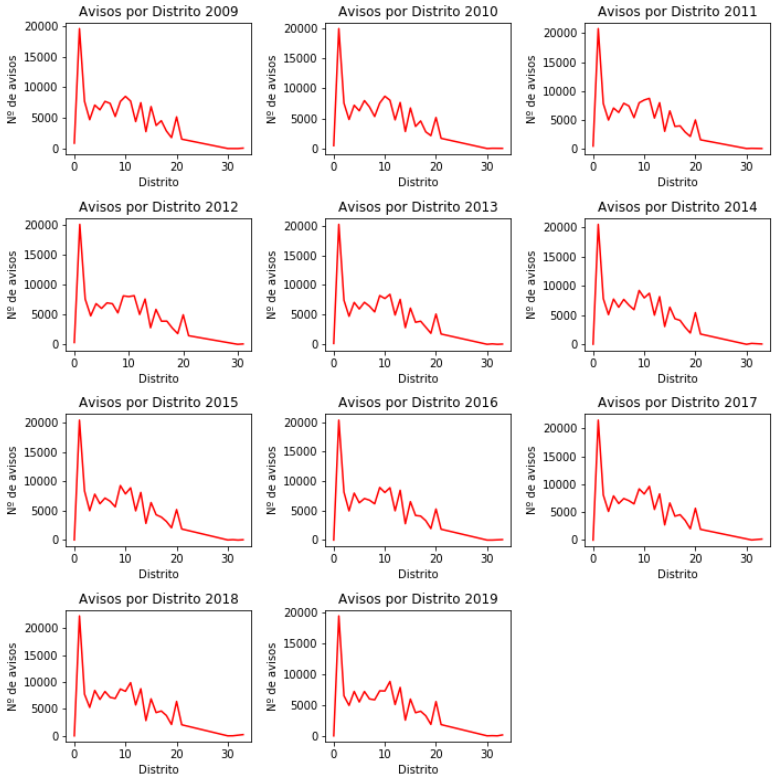


Figura 3.5: Nº de Avisos por distrito y año

Como podemos comprobar la mayor parte de los avisos se concentran en el distrito número 1, que corresponde al distrito del centro de Madrid. Como en los ejemplos anteriores agrupamos los datos para mostrar una gráfica que acumula todos los años, anotando en el eje de abscisas el distrito correspondiente.

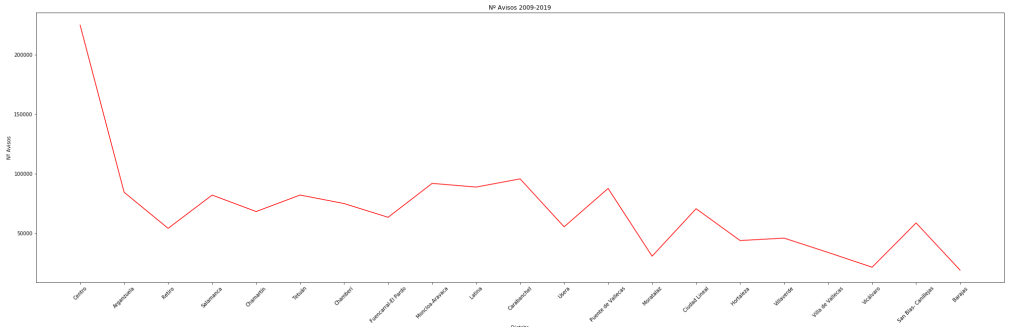


Figura 3.6: Nº de Avisos por distritos

3.1.2.4. Relación entre los avisos los fines de semana y los días de diario

Dado que es difícil determinar si un día del mes, por ejemplo, el 10 es un día en el que siempre hay muchos o pocos avisos, consideramos más importantes clasificarlos por avisos entre los diferentes días de la semana, así como diferenciar avisos de Lunes-Jueves y los avisos del fin de Semana (Viernes-Domingo). A continuación se adjuntan diferentes gráficas visualizando los datos.

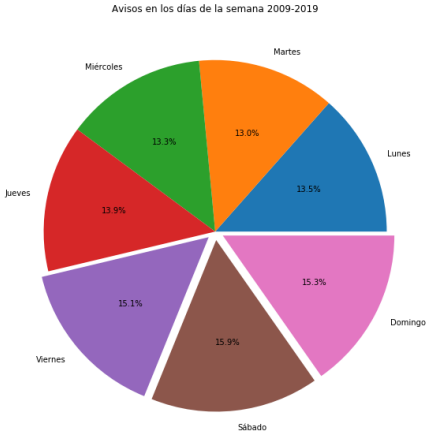


Figura 3.7: Porcentaje de avisos diarios

Podemos comprobar que el número de avisos no cambia en gran medida entre los diferentes días de la semana y también entre los días del fin de semana. Sin embargo entre los días de diario y del fin de semana observamos un aumento de los avisos como se puede comprobar en la siguientes gráficas:

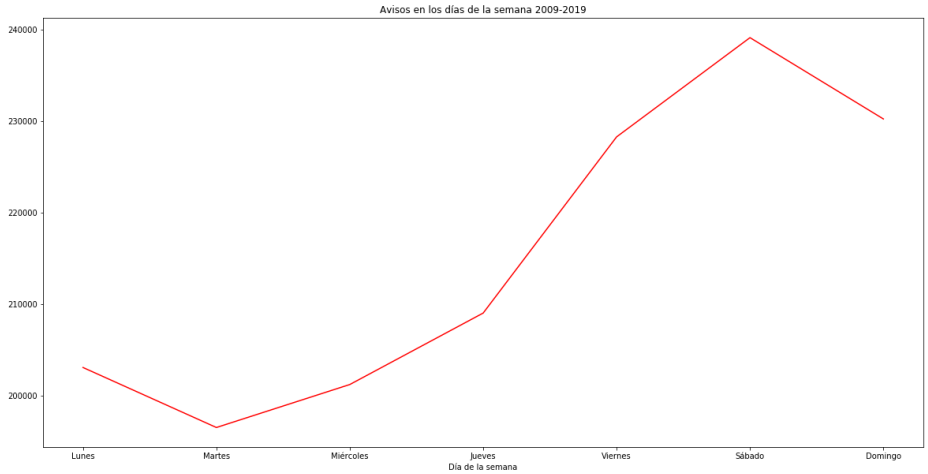


Figura 3.8: Nº de Avisos por día de la semana

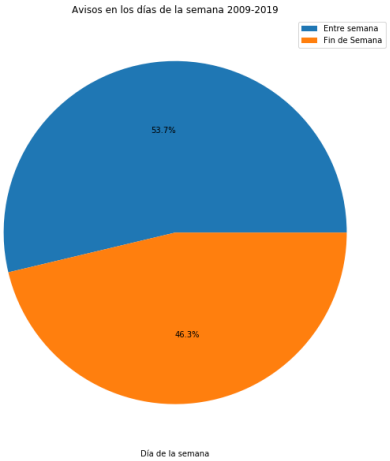


Figura 3.9: Comparativo avisos Diario vs Fin de Semana

3.1.2.5. Número de avisos por base

Para explorar esta posible opción se han generado los siguientes gráficos. El primero de ellos muestra en número de avisos por cada base, debemos tener en cuenta que la Base nº -1 significa avisos no realizados.

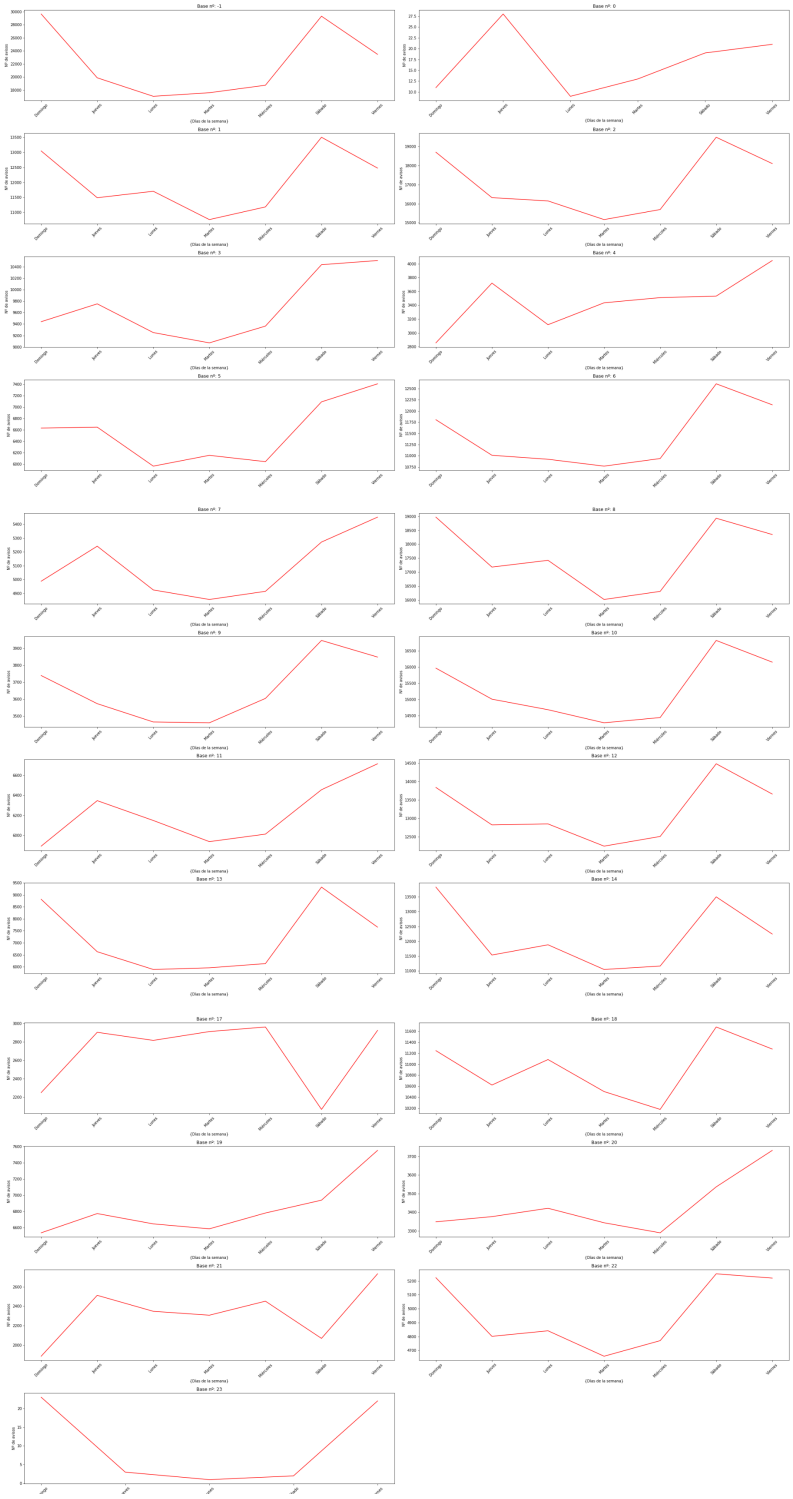


Figura 3.10: Nº de Avisos por base en los días de la semana

3.1.2.6. Número de avisos por turno

En este apatado se muestra una gráfica tras agrupar los datos por día de la semana y turno, obteniendo en nº de avisos de SAMUR-PC.

Como podemos observar en la siguiente gráfica los turnos de mañana y tarde a lo largo

de toda la semana no varían demasiado. El turno de noche es el que mayor diferencia presenta, observando un aumento notable de los avisos durante las noches del Viernes, Sábado y Domingo.

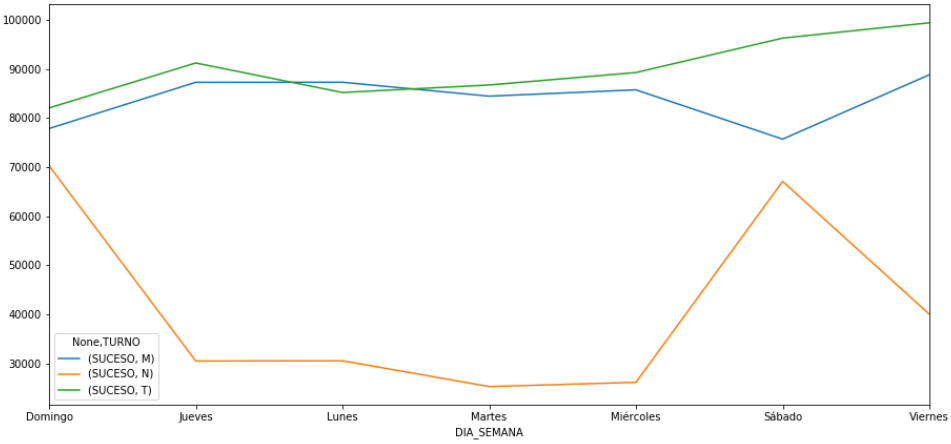


Figura 3.11: Nº de Avisos por turno y día de la semana

3.2. Técnicas de aprendizaje empleadas

3.2.1. Regresión Lineal

En este aparatado se aplicará técnica de Regresión Lineal sobre los datos de SAMUR-PC teniendo el nº de Avisos como variable objetivo . Se utilizarán de los 10 años, nueve de ellos de entrenamiento, y se utilizará el año 2018 para predecir y comprobar los resultados obtenidos. De todas formas, por si el año 2018, no fuera representativo también se aplicará validación cruzada a los modelos realizados, es decir, se escogerán porciones de los datos para entrenamiento y test calculando las métricas (RMSE y MAE) para estos datos, después de ello la media de las métricas obtenidas de las particiones será la métrica final.

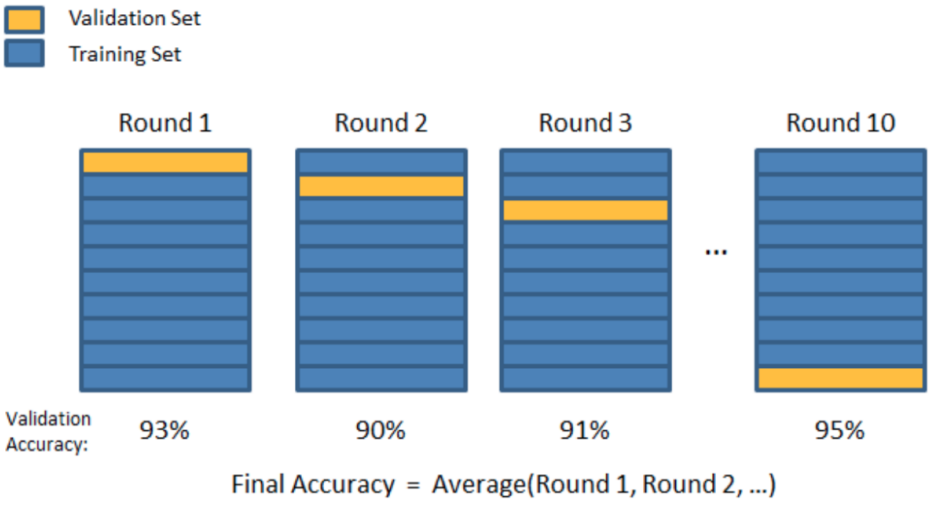


Figura 3.12: Validación cruzada

En el anexo se encuentra un fichero denominado Regresión Lineal.ipynb", donde se encuentra el código para crear el modelo, entrenarlo y ejemplos de predicción de datos. Los modelos realizados y los correspondientes valores de las métricas son las siguientes:

ID Modelo	**Variable Objetivo**	**Variables Entrenamiento**	**RMSE**	**MAE**
1	Nº de Avisos	Nº de Mes - Nº de día del mes	63.68	49.55
2	Nº de Avisos	Nº de Mes - Nº de día del mes Día de la semana	54.42	44.37
3	Nº de Avisos	Nº de Mes - Nº de día del mes Día de la semana Base operativa	10.29	8.47
4	Nº de Avisos	Nº de Mes - Nº de día del mes Día de la semana Distrito	8.59	6.60
5	Nº de Avisos	Nº de Mes - Nº de día del mes Día de la semana Turno	41.03	33.52
6	Nº de Avisos	Nº de Mes - Nº de día del mes Día de la semana Base Turno	4.18	3.36
7	Nº de Avisos	Nº de Mes - Nº de día del mes Día de la semana Base Dispositivo	5.03	3.98
8	Nº de Avisos	Nº de Mes - Nº de día del mes Día de la semana Distrito Dispositivo	3.48	2.59
9	Nº de Avisos	Nº de Mes - Nº de día del mes Día de la semana Dispositivo	40.70	28.97
10	Nº de Avisos	Nº de Mes - Nº de día del mes Día de la semana Base Turno Dispositivo	2.22	1.76

ID Modelo	Variable Objetivo	Variables Entrenamiento	RMSE	MAE
11	Nº de Avisos	Nº de Mes - Nº de día del mes Día de la semana Distrito Turno Dispositivo	1.67	1.26
12	Nº de Avisos	Nº de Mes - Nº de día del mes Día de la semana Dispositivo Turno	16.31	11.85
13	Nº de Avisos	Nº de Mes - Nº de día del mes Día de la semana Distrito Turno	4.03	3.11

Tabla 3.1: Resultados Regresiones Lineales

Tras todos los modelos realizados, hemos observado que los resultados RMSE y MAE son demasiado grandes por lo que se aplicarán Redes neuronales para intentar obtener mejores resultados.

3.2.2. Redes neuronales

En este aparatado se aplicarán Redes neuronales sobre los datos de SAMUR-PC teniendo el nº de Avisos como variable objetivo . Se utilizarán de los 10 años, nueve de ellos de entrenamiento, y se utilizará el año 2018 para predecir y comprobar los resultados obtenidos.

Al comienzo de esta fase se estudiaron los diferentes tipos de redes neuronales (CNN y LSTM) explicadas en el apartado 2.3 de este trabajo y estudiados en los artículos [2] [3] [4] [5] [6]. Se obtuvieron mejores resultados para redes LSTM que para redes CNN. A continuación mostramos un ejemplo de ellos, tratando de predecir el número de avisos para una fecha determinada. En las gráficas en el eje Y se encuentra los valores de la métrica RMSE y en las X el tamaño de las diferentes capas probadas.

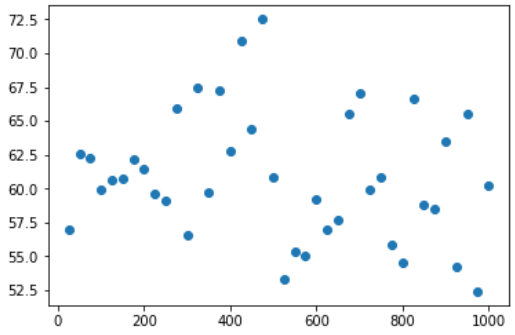


Figura 3.13: RMSE utilizando CNN

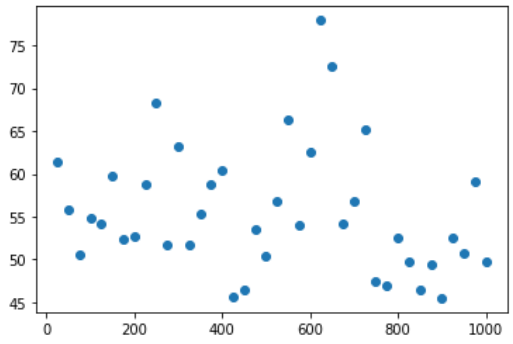


Figura 3.14: RMSE utilizando LSTM

Como podemos observar la segunda gráfica adquiere un mínino menor que la primera, esto se debe a las virtudes propias de la red LSTM como se ha comentado en el apartado 2.2.4. Por ello los modelos de redes neuronales se implementarán realizando redes neuronales LSTM. Para seleccionar la función de activación y la función de pérdida se ha seguido el siguiente artículo [7], seleccionando finalmente la función de activación 'relu' y la función de pérdida 'mse' para minimizar el RMSE final.

En el anexo se encuentra un fichero denominado "Redes neuronales.ipynb", donde se encuentra el código para crear el modelo, entrenarlo y ejemplos de predicción de datos.

Para esta primera fase en Redes neuronales, se crea una red neuronal formada por la capa de entrada, la capa de salida y dos capas ocultas. Para determinar el tamaño de las capas ocultas se iteran entre unos valores determinados, por cada par de valores se crean dos capas ocultas y obtenemos el RMSE. Para esta primera fase de redes neuronales, se ha intentado buscar una mejoría mínima entorno 20 % respecto a las regresiones lineales realizadas.

El principal problema que ha tenido lugar en esta fase ha sido el siguiente, al desarrollar las pruebas en un entorno local, y dado el gran volumen de datos (entorno a 1.5 millones de registros o activaciones de SAMUR-PC durante los años 2009-2019) no se han podido estudiar el comportamiento en redes más grandes ni la presencia de sobreajuste utilizando conjuntos de entrenamiento y validación.

Los modelos realizados y los correspondientes valores de las métricas son las siguientes:

ID Modelo	Variables Entrenamiento	RMSE RL	RMSE NN	Mejora (%)
1	Nº de Mes - Nº de día del mes	63.68	51.58	19 %
2	Nº de Mes - Nº de día del mes Día de la semana	54.42	43.95	19.23 %
3	Nº de Mes - Nº de día del mes Día de la semana Base operativa	10.29	8.4	18.4 %
4	Nº de Mes - Nº de día del mes Día de la semana Distrito	8.59	6.26	27.12 %
5	Nº de Mes - Nº de día del mes Día de la semana Turno	41.03	26.41	35.63 %
6	Nº de Mes - Nº de día del mes Día de la semana Base Turno	4.18	3.71	11.24 %
7	Nº de Mes - Nº de día del mes Día de la semana Base Dispositivo	5.03	4.52	10.13 %
8	Nº de Mes - Nº de día del mes Día de la semana Distrito Dispositivo	3.48	2.37	31.9 %
9	Nº de Mes - Nº de día del mes Día de la semana Dispositivo	40.70	15.98	60.8 %
10	Nº de Mes - Nº de día del mes Día de la semana Base Turno Dispositivo	2.22	2.07	6.8 %

ID Modelo	Variables Entrenamiento	RMSE RL	RMSE NN	Mejora (%)
11	Nº de Mes - Nº de día del mes Día de la semana Distrito Turno Dispositivo	1.67	1.48	11.4 %
12	Nº de Mes - Nº de día del mes Día de la semana Dispositivo Turno	16.31	6.64	59.3 %
13	Nº de Mes - Nº de día del mes Día de la semana Distrito Turno	4.03	3.33	17.4 %

Tabla 3.2: Resultados Redes Neuronales Dos Capas

Como hemos visto en la tabla anterior se ha conseguido mejorar todos los modelos, pero no en todos ellos se ha conseguido una mejora notable. El principal factor ha sido limitar el número de las capas que se han comprobado, ya que con los equipos que contaban los estudiantes, entrenar redes neuronales de mayor tamaño suponía un coste de tiempo inviable. Por ello, se investigó la posibilidad de utilizar sistemas cloud para entrenar los modelos, de esta forma aunque el coste en tiempo fuera mayor, la capacidad de procesamiento aumentaba en gran forma y permitiría entrenar redes neuronales de mayor tamaño.

Se realizaron pruebas con la CPU de Google [8], creando los modelos en Google Colaboratory, [9] [10]. Previo a la creación de los modelos se tuvieron que anonimizar todas las activaciones de SAMUR-PC, como se ha comentado anteriormente, todo proceso de transformación de datos se encuentra detallado en el fichero Utils SAMUR-PC.ipynb localizado en el anexo del trabajo. Previamente a la realización de modelos se investigó sobre qué utilizar como acelerador de hardware, para ello se comprobó cuanto tiempo tardaba en entrenar nuestro primer modelo de red neuronal de dos capas ocultas y un número de nodos aleatorios en cada una de ellas, el resultado fue este:

Elegimos Hardware

```
[ ]  def gpu():
       with tf.device('/device:GPU:0'):
         model = lstm_model_three_layers(50,50,50,2)
         model.fit(X_train, Y_train, epochs=150, verbose=0)
     def cpu():
       with tf.device('/cpu:0'):
         model = lstm_model_three_layers(50,50,50,2)
         model.fit(X_train, Y_train, epochs=150, verbose=0)

     print('GPU (s):')
     gpu_time = timeit.timeit('gpu()', number=1, setup="from __main__ import gpu")
     print(gpu_time)

     print('CPU (s):')
     cpu_time = timeit.timeit('cpu()', number=1, setup="from __main__ import cpu")
     print(cpu_time)
```

```
GPU (s):
274.1648142459999
CPU (s):
95.2441840800002
```

Figura 3.15: CPU vs GPU.png

Como podemos comprobar y aunque parezca sorprendente, tarda considerablemente menos tiempo con CPU que con GPU (graphics processing unit), esto se debe a que la GPU es muy recomendada para redes neuronales convulacionales, es decir, cuando queremos encontrar características en imágenes, sin embargo para redes neuronales LSTM, que se caracterizan por buscar características en datos de forma secuencial, es mejor utilizar CPU.

Esta segunda fase de redes neuronales se desarrolló de la siguiente forma para cada uno de los modelos:

1. **Buscar el nº de capas ocultas:** Para ello dividimos el conjunto de datos, en dos subconjuntos. Por un lado el conjunto de entrenamiento y por otro lado el conjunto de validación. Tras ello creamos una curva de validación, es decir, creamos redes neuronales con diferentes nº de capas evaluando el coste, es decir, la función de pérdida, en nuestro caso, el error cuadrático medio. Dibujamos una gráfica representado ambos costes, se seleccionará aquel punto en el que coste del conjunto de validación es mínimo. Por ejemplo para el modelo nº2 obtenemos la siguiente gráfica:

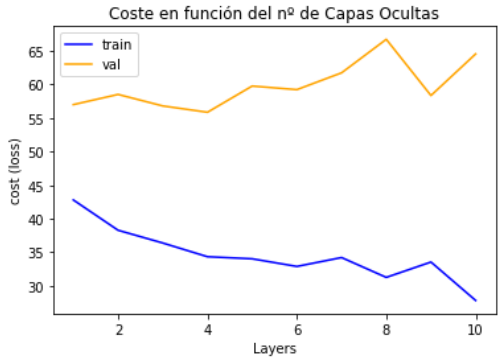


Figura 3.16: Costes conjunto entrenamiento y validación

2. **Buscar tamaño de las capas**: Tras seleccionar el nº de capas, debemos seleccionar el nº de nodos de cada capa, para ello se generaran nuevos modelos, en cada modelo se generaran el tamaño de las capas de forma aleatoria, buscando encontrar una mejora. Además sobre el tamaño aleatorio de estas capas creadas se buscarán diferentes configuraciones del parámetro de regularización (Dropout) para evitar el sobre-ajuste a los datos (Overfitting) y así que nuestro modelo no memorice los datos y sea correcto para generalizar con nuevos conjuntos de datos. A continuación se muestra un ejemplo de los modelos generados para encontrar el primer modelo final:

```
Fit model with [34] sizes and Dropout = 0.1
Test RMSE = 54.045301978148586
Fit model with [34] sizes and Dropout = 0.2
Test RMSE = 56.229978619617455
Fit model with [34] sizes and Dropout = 0.3
Test RMSE = 59.752848791475955
Fit model with [67] sizes and Dropout = 0.1
Test RMSE = 49.02043761816737
Fit model with [67] sizes and Dropout = 0.2
Test RMSE = 51.5275129835329
Fit model with [67] sizes and Dropout = 0.3
Test RMSE = 57.13862461410709
Fit model with [53] sizes and Dropout = 0.1
Test RMSE = 58.40781029963219
Fit model with [53] sizes and Dropout = 0.2
Test RMSE = 47.20334139084302
Fit model with [53] sizes and Dropout = 0.3
Test RMSE = 55.824934288424714
Fit model with [73] sizes and Dropout = 0.1
Test RMSE = 51.652820953152265
Fit model with [73] sizes and Dropout = 0.2
Test RMSE = 48.448419014146964
Fit model with [73] sizes and Dropout = 0.3
Test RMSE = 45.192252688904496
Fit model with [40] sizes and Dropout = 0.1
Test RMSE = 53.75480920103496
Fit model with [40] sizes and Dropout = 0.2
Test RMSE = 53.19972852728811
Fit model with [40] sizes and Dropout = 0.3
Test RMSE = 53.46532553490279
Fit model with [93] sizes and Dropout = 0.1
Test RMSE = 51.06215963600691
Fit model with [93] sizes and Dropout = 0.2
Test RMSE = 47.9948799471379
Fit model with [93] sizes and Dropout = 0.3
Test RMSE = 51.41987380256345
Mejor RMSE: 45.192252688904496
```

Figura 3.17: Modelos creados aleatoriamente

3. **Selección de modelo:** Para seleccionar el modelo, se escogerá aquel que minimice el RMSE de entre todos los modelos creados en el paso anterior.

Los resultados de la Fase II de redes neuronales se pueden ver a continuación:

ID Modelo	Variables Entrenamiento	RMSE RL	RMSE NN Cloud	Mejora (%)
1	Nº de Mes - Nº de día del mes	63.68	57.20	11 %
2	Nº de Mes - Nº de día del mes Día de la semana	54.42	45.19	17 %
3	Nº de Mes - Nº de día del mes Día de la semana Base operativa	10.29	8.23	20 %
4	Nº de Mes - Nº de día del mes Día de la semana Distrito	8.59	6.52	24,1 %
5	Nº de Mes - Nº de día del mes Día de la semana Turno	41.03	23.92	42 %
6	Nº de Mes - Nº de día del mes Día de la semana Base Turno	4.18	4.16	1 %
7	Nº de Mes - Nº de día del mes Día de la semana Base Dispositivo	5.03	4.35	13.5 %
8	Nº de Mes - Nº de día del mes Día de la semana Distrito Dispositivo	3.48	2.41	30 %
9	Nº de Mes - Nº de día del mes Día de la semana Dispositivo	40.70	14.08	65.4 %
10	Nº de Mes - Nº de día del mes Día de la semana Base Turno Dispositivo	2.22	2.05	7.66 %

ID Modelo	Variables Entrenamiento	RMSE RL	RMSE NN Cloud	Mejora (%)
11	Nº de Mes - Nº de día del mes Día de la semana Distrito Turno Dispositivo	1.67	1.47	11.9 %
12	Nº de Mes - Nº de día del mes Día de la semana Dispositivo Turno	16.31	6.45	60.4 %
13	Nº de Mes - Nº de día del mes Día de la semana Distrito Turno	4.03	3.38	16.12 %

Tabla 3.3: Resultados Redes Neuronales - Sistemas Cloud

Capítulo 4

Desarrollo de la aplicación

En este capítulo se describen el prototipo inicial realizado a SAMUR-PC, y el prototipo final desarrollado listo para desplegar cuando ellos consideren.

4.1. Arquitectura de la aplicación

Dado que se trata de una aplicación web, la mejor arquitectura y la utilizada en la aplicación es un modelo cliente-servidor, donde el cliente inicia la comunicación enviando información al servidor y el servidor permanece conectado a la espera de información que llegue por un determinado puerto.

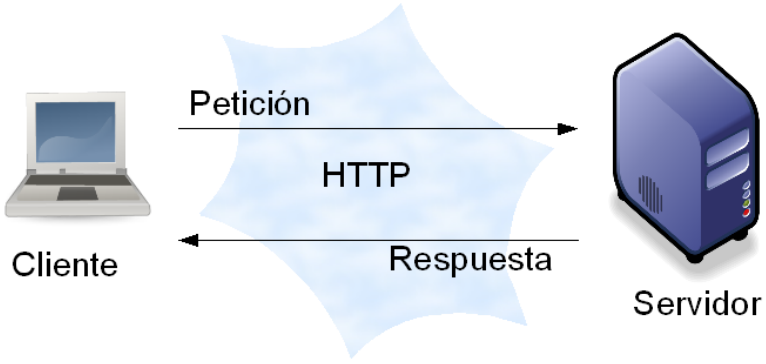


Figura 4.1: Arquitectura del prototipo

4.2. Prototipo Inicial

El prototipo se comenzó a desarrollar de forma paralela a la obtención de los datos procedentes de SAMUR-PC. Durante las primeras semanas de curso se buscó qué tecnologías utilizar. Por un lado se planteó crear una aplicación de escritorio utilizando Java con JavaSwing para aportar el componente gráfico a la misma. En cuanto a la arquitectura de esta aplicación había dos opciones, un MVC (modelo vista controlador) vista en la asignatura TP, o, una arquitectura multicapa vista en la asignatura (IS, MS).

Dado que queríamos que la aplicación pudiera ser utilizada en diferentes plataformas con comodidad, finalmente nos decantamos por una aplicación web basada en el modelo cliente-servidor.

La primera decisión que se tomó fue implementar el lado del servidor con PHP ya que uno de los integrantes del grupo lo conocía y había desarrollado alguna aplicación con él en otros cursos. Durante los dos primeros meses del curso se desarrolló la aplicación de forma continuada, los miembros del TFG mantuvieron reuniones cada dos semanas para evitar posibles desviaciones del proyecto y ser fieles a la planificación inicial.

En noviembre tuvimos una reunión con el director del TFG, el cual nos sugirió indagar en Canvas (tecnología de HTML5) para crear un mapa dinámico con las posibles localizaciones de avisos de SAMUR- Protección Civil dada una fecha. Tras varias semanas se investigó sobre Canvas y librerías de javascript como leaflet y se creó un prototipo de aplicación para mostrar a SAMUR-PC.

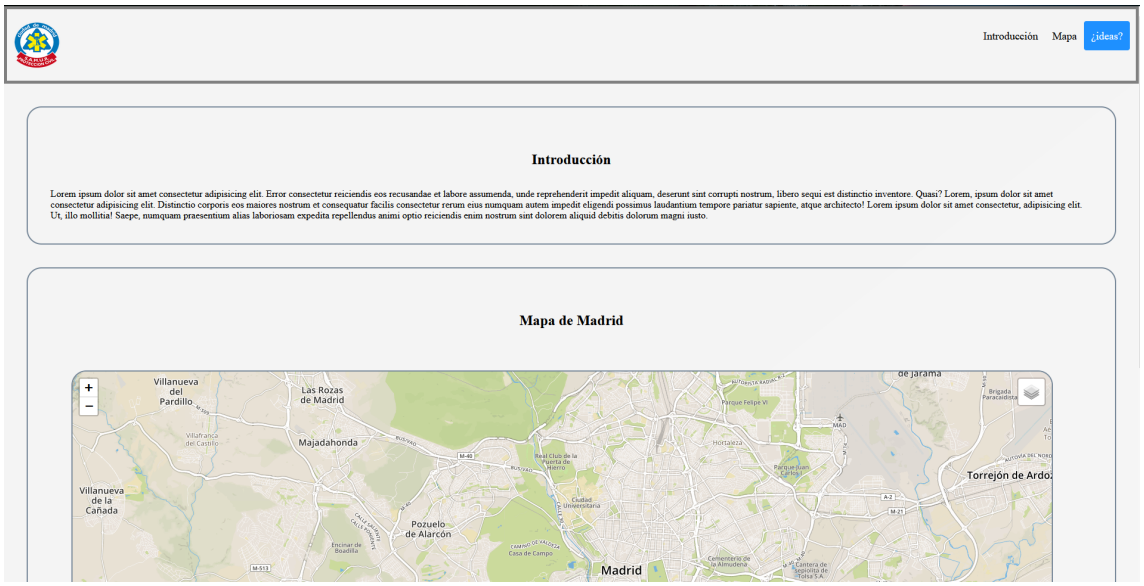


Figura 4.2: Captura prototipo aplicación

Tras la reunión con SAMUR-PC, presentamos el primer prototipo de la aplicación y quedaron satisfechos con él y nos hicieron algunas aportaciones que nos harían cambiar la idea de la aplicación. En vez de un mapa dinámico se nos ocurrió la posibilidad de generar un informe de manera dinámica con toda la información que fuera posible predecir con una alta precisión y permitir descargarla en formato pdf, con esto se aportaría mucha más información y además la geolocalización de los avisos no era buena idea dado que el aprendizaje automático se basa en datos históricos y dos avisos difícilmente podrán tener las mismas coordenadas, por lo que no llegaríamos a modelos predictivos fiables.

Uno de los integrantes de proyecto realizó las prácticas de empresa como ingeniero de datos, aprendió una de las herramientas de visualización (Microsoft Power BI) de la más punteras en el mercado, por lo que se decidió aplicarla al proyecto. A SAMUR-PC le encantó la idea ya que además de que la aplicación sirviera para realizar predicciones sobre los avisos, también contaría con una gráfica para visualizar los avisos en los últimos años por distrito, por código de aviso, tipo de dispositivo... con la que se podría interactuar.

Figura 4.3: Informe Power BI

4.3. Prototipo Final: Aplicación web

A finales de noviembre, uno de los integrantes aprendió en la asignatura AW (Aplicaciones web) implementar una aplicación mediante NodeJs y ExpressJs. Dado que la tecnología aprendida era más moderna y permitía una código más claro y sencillo, durante las vacaciones de Navidad se cambió la implementación de la aplicación a las tecnologías ya citadas. El resultado fue una aplicación con una arquitectura más sólida, la lógica centralizada y sin estar acoplada a la BD.

Por tanto a la vuelta de Navidad se consigue tener el prototipo de la aplicación web implementada. Este prototipo recibe consultas del cliente mediante formularios, el servidor llama a un script de python que genera las correspondientes predicciones y lo guarda en un archivo en csv, que después es mostrado al cliente en una página, este informe puede ser descargado en PDF.

Por último, dado que el despliegue queda en manos de SAMUR- Protección Civil, se ha dockerizado la aplicación, es decir, se ha creado un contenedor de Docker el el cual se encuentra la Base de datos para el acceso a la aplicación. Por otra parte se ha creado otro contenedor que contiene la aplicación de node.js y se han relacionado ambos. De esta forma, no es necesario instalar todas las tecnologías que utiliza el proyecto; solamente con descargar Docker, se podrá ejecutar la aplicación. Todo esto se encuentra detallado en un documento denominado ”Manual de Usuario”que se entregará como anexo al proyecto.

A continuación se adjunta unas imágenes de la apariencia del prototipo final a falta de desplegar cuando SAMUR-PC considere:

Figura 4.4: Portal de inicio del prototipo

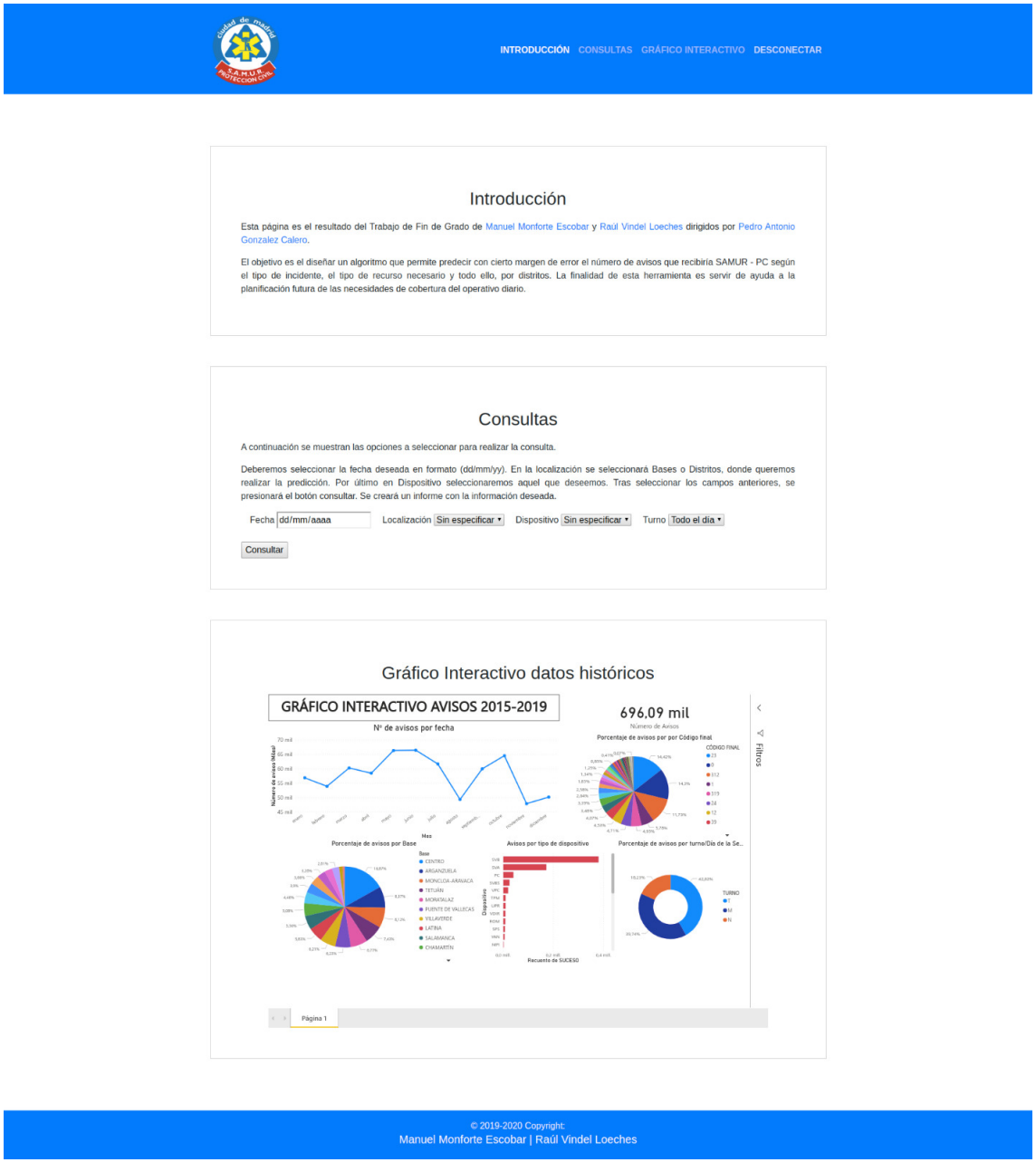


Figura 4.5: Página principal del prototipo

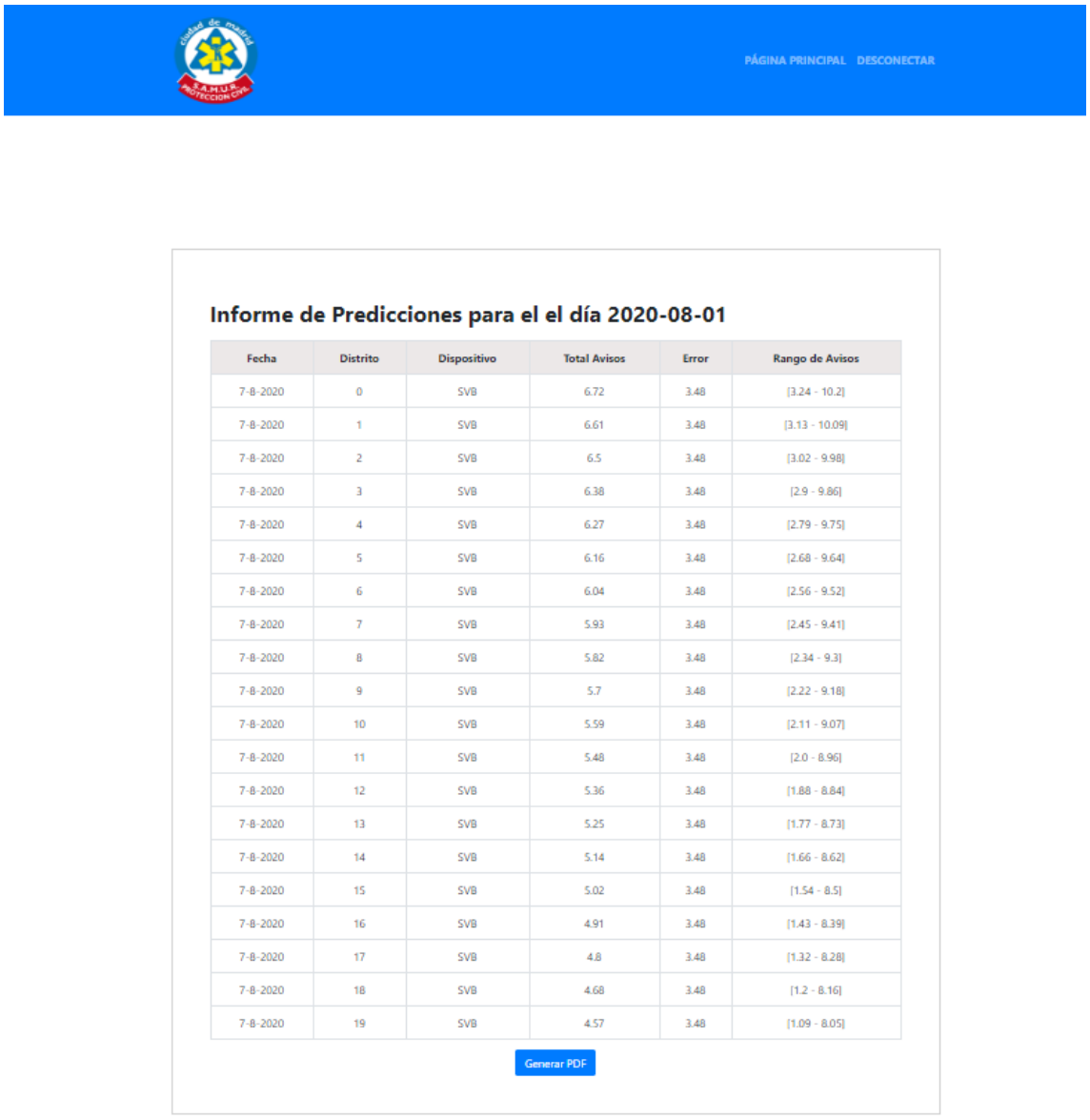


Figura 4.6: Captura para predicción del 2020-08-01

Capítulo 5

Conclusiones y Trabajo Futuro

5.1. Conclusiones

El objetivo de este trabajo ha sido la aplicación de técnicas de aprendizaje automático sobre los datos que nos ha proporcionado SAMUR - Protección Civil para apoyar la planificación del operativo diario. El estudio se ha centrado tras mantener reuniones con Fernando Monforte Escobar en predecir el número de activaciones en función del tipo de dispositivo, del distrito o de la base a la que se destina dicha activación para una fecha determinada.

Las técnicas utilizadas han sido Regresión Lineal y Redes neuronales, realizando una profunda investigación en ésta última para conocer cual es el mejor tipo de red neuronal a aplicar. A continuación mostramos una tabla con los resultados finales:

ID Modelo	Variables Entrenamiento	R. Lineal	RN Fase I	RN Fase II
1	Nº de Mes - Nº de día del mes	63.68	51.58	57.20
2	Nº de Mes - Nº de día del mes Día de la semana	54.42	43.95	45.19
3	Nº de Mes - Nº de día del mes Día de la semana Base operativa	10.29	8.4	8.23
4	Nº de Mes - Nº de día del mes Día de la semana Distrito	8.59	6.26	6.52
5	Nº de Mes - Nº de día del mes Día de la semana Turno	41.03	26.41	23.92
6	Nº de Mes - Nº de día del mes Día de la semana Base Turno	4.18	3.71	4.16
7	Nº de Mes - Nº de día del mes Día de la semana Base Dispositivo	5.03	4.52	4.35
8	Nº de Mes - Nº de día del mes Día de la semana Distrito Dispositivo	3.48	2.37	2.41
9	Nº de Mes - Nº de día del mes Día de la semana Dispositivo	40.70	15.98	14.08
10	Nº de Mes - Nº de día del mes Día de la semana Base Turno Dispositivo	2.22	2.07	2.05

ID Modelo	Variables Entrenamiento	R. Lineal	RN Fase I	RN Fase II
11	Nº de Mes - Nº de día del mes Día de la semana Distrito Turno Dispositivo	1.67	1.48	1.47
12	Nº de Mes - Nº de día del mes Día de la semana Dispositivo Turno	16.31	6.64	6.45
13	Nº de Mes - Nº de día del mes Día de la semana Distrito Turno	4.03	3.33	3.38

Tabla 5.1: Tabla conclusiones

Como podemos observar durante el proyecto se han ido mejorando de forma progresiva los modelos realizados. En todos ellos observamos que las redes neuronales obtienen una mejora frente a regresión lineal, esto explica el gran uso y la importancia que ha adquirido esta técnica en los últimos años.

Este proyecto nos ha permitido adquirir conocimientos del aprendizaje automático con mayor profundidad, explorando los diferentes tipos de redes neuronales existentes, así como adquirir conocimientos transversales que utilizaremos a lo largo de nuestra vida profesional como por ejemplo realizar reuniones con stakeholder y clientes para conocer los intereses de un producto y el proceso de realización del mismo.

Por último, destacamos que dado que el presente TFG se centra en un trabajo de investigación, también hemos querido desarrollar un prototipo aprovechando las conclusiones de este trabajo para apoyar la toma de decisiones de SAMUR - Protección Civil en lo relativo a la planificación del operativo diario, añadiendo la posibilidad de visualizar los avisos ocurridos durante los últimos años y las predicciones realizadas a partir de estos modelos para una fecha y circunstancias determinadas.

5.2. Trabajo Futuro

Como posibles trabajos futuros pueden señalarse los siguientes:

- **Añadir términos polinómicos:** Con el objetivo de intentar mejorar los modelos obtenidos a lo largo del proyecto, se pueden añadir términos polinómicos como variables independientes, es decir, crear nuevas variables independientes como la combinación lineal de una misma columna.
- **Explorar nuevas configuraciones de Redes Neuronales:** Dado que uno de los mayores inconvenientes ha sido el coste en tiempo de entrenamiento de los diferentes modelos utilizando redes neuronales, uno de los siguientes pasos a dar, sería la utilización de la CPU de Google Cloud de pago para poder explorar nuevas configuraciones y seguir mejorando los modelos.
- **Creación de curvas de aprendizaje**: Un paso a futuro, sería la creación en cada modelo de una curva de aprendizaje con el objetivo de conocer, si sería interesante obtener nuevos datos para intentar mejorar la precisión del modelo o ya se ha llegado a la máxima precisión con los datos disponibles.
- **Integración Base de Datos SAMUR-PC:** Otro de los pasos a seguir para mejorar el prototipo creado a SAMUR - Protección Civil sería la integración de la Base de Datos de las activaciones para poder retro-alimentar el informe de PowerBI de la página Web.
- **Retroalimentación de modelos SAMUR-PC:** Uno de los pasos más importantes y interesantes para continuar con el proyecto sería, tras la integración de la Base de datos, entrenar los modelos con los nuevos avisos que ocurren cada día para mantenerlos actualizados a lo largo del tiempo y así ajustarse a nuevas necesidades y patrones en los datos.
- **Creación de la app final:** Para mejorar la escalabilidad y flexibilidad de la aplicación, uno de los posibles trabajos futuros sería la migración de la app web a una PWA (Progressive Web App) de esta forma el personal de SAMUR-PC la tendría disponible en su teléfono móvil.
- **Reunión con el equipo informático de SAMUR-PC para ceder app y explicar el funcionamiento**

Capítulo 6

Introduction

6.1. Motivation

The daily activity of an emergency service requires correct planning of the type and number of operational resources and coverage of the bases, with the aim of being as efficient as possible. Some of the main consequences of this efficiency are an improvement in the response times, as well as a correct distribution of the workload in the professionals and even helps to reduce the expense by not mobilizing the different devices without necessity. In all areas, there is a need to collect information in order to be able to face future circumstances. SAMUR - Civil Protection has a technological platform in which all the activations received by the emergency service are collected and stored in a database.

6.2. Context

SAMUR - Civil Protection is an internationally recognised emergency service operating in the city of Madrid.

SAMUR-PC arises from the need to create an emergency service in the city of Madrid. Starting from the old ambulance park, new models are designed and communications are updated, improving at the same time the uniforms, selection and training of the personnel. In December 1992, after demonstrating the usefulness of the service, it was decided to grant the category of Health Transport.

The objective of this service is to quickly and efficiently resolve health emergencies that occur on public roads within the city of Madrid. Thus changing the perspective from 'transfer' to 'care at the place of the event and subsequent transfer'. In 1995, SAMUR became Samur-Protección Civil, acquiring more responsibilities.

The present End of Degree Work arises from the possibility of improving the planning of the daily operation of SAMUR - Civil Protection from the activations stored in the Database.

6.3. Purpose of the Investigation

As mentioned above, the planning of the daily operation of an emergency service takes on great importance.

Based on the database of notices produced between 2009-2019, the possibility of applying machine learning techniques will be studied to help find out which days and areas of Madrid will have the greatest number of notices and what type of assistance resource they will need to resolve them. As a complement to the research work, a prototype web application will be created to show SAMUR-PC the possibility of using the techniques used to predict the number of warnings in certain circumstances.

6.4. Workplan

The work is formed by the research project itself related to applying Machine Learning techniques to the data provided by SAMUR - Protección Civil. In order to show the results of the selected learning techniques, a prototype of a web application has been created for SAMUR-PC that will be delivered locally, leaving the deployment of the application in their hands. The central axis of the work is the application of Machine Learning techniques on the available data.

The work plan with its respective phases is summarized below:

1. **Proposal and Project Research**: In April 2019, Fernando Monforte gave us the possibility of exploiting SAMUR-PC's operational data and developing an investigation that would help in planning the assistance resources of the daily operation. We found the idea interesting and once we had basic knowledge of the types of data being recorded, we began to work on it, coming to the conclusion that the application of Machine Learning techniques could be useful.

   Once we were clear about the possibilities of applying Machine Learning techniques, Fernando Monforte arranged a meeting between us and the SAMUR-PC management team to present the research project, acting as an intermediary between us and the SAMUR-PC management team throughout the process.

   In the meeting held with the management team of SAMUR-PC and with Fernando Monforte, we transmitted the possibilities posed by the exploitation of data through machine learning, and from SAMUR-PC they transmitted to us the areas in which they were most interested, as well as the details of possible predictions that would be of interest as a support tool for the daily operational planning of the service, insisting on the need to have some parameter that would serve as an indicator of the accuracy of the prediction.

   In this context, and taking into account the scope of knowledge of the research project, we decided to contact Pedro Antonio González Calero, professor of the subject Machine Learning and Big Data in the Faculty of Computer Science, who finally agreed to be the tutor of the project.

   During the following months we held several meetings with the management team of SAMUR-PC, to clarify and resolve legal aspects of the research project, since it involved the use of confidential data from the Sub-directorate General of SAMUR-Civil Protection, and therefore required the appropriate authorisation from

the Sub-directorate General of SAMUR-Civil Protection and the Directorate General of Emergencies and Civil Protection of the Madrid City Council.

At the same time, while the necessary legal procedures were being completed and until the databases that will be used for the development of the project were available, the students researched the main technologies and frameworks for developing the prototype web application and applying Machine learning techniques.

To do this, research was done on the programming language *Python* and its libraries *Numpy, Pandas, Keras* and *Scikit Learn*.

2. **Web application development**: Once the project definition phase was sufficiently advanced and all the necessary knowledge was acquired, the second phase began: the development of the web prototype. During the first three months of the academic year (October-December), a prototype was developed to showcase SAMUR-PC. After validation and suggestions from SAMUR-PC, the final prototype was made in the following month, changing the technologies used in the original. This change will be explained in the corresponding section in more detail.

3. **Machine Learning Applications**: This phase was carried out over a period of approximately six months (January - June).

   A) Exploration of the data: This first phase was carried out over a period of approximately one month. It began with the execution of the ETL, a process in which transformations were applied to the data to prepare them for their subsequent representation in graphs. Some of the transformations to be highlighted are the following:

   - Delete spaces in String columns
   - Correct Base Numbering: In SAMUR-PC, there are a total of 23 Bases, numbered from 1 to 23. Those notices that are not finally carried out are assigned atypical Bases with numbers such as 1000, 5000. This transformation consisted in transforming all these Bases to -1, in order to group all the no warnings in the same Base to which they refer.

   After this data cleaning, graphs and tables were generated to achieve a better understanding of the data for the subsequent application of the Machine learning techniques.

   B) Machine learning: This second phase was carried out during the four months following the previous one. Different Machine Learning techniques were applied to the data in order to answer questions requested by SAMUR-PC. Furthermore, in this phase, in order to carry out one of the corresponding phases of creating predictive models, it was necessary to anonymise the SAMUR-PC data in order to be able to apply learning techniques in cloud systems.

4. **Results**: Finally, in the results phase, the analysis of all those obtained from the numerous tests carried out was carried out and conclusions were drawn from these results, including the integration of the models created into the web prototype designed during the second phase of the project.

6.5. Project structure

The rest of the work is organized in 8 chapters and 4 annexes with the structure described below:

The Chapter 2 introduces some general concepts to know what Machine Learning consists of. The origin of Machine Learning and the main techniques used today are described.

Chapter 3, the central axis of the project. Describes the data exploration phase. It includes the performance of ETL (Extract, Transform and Load) processes, the representation of the data by means of graphs for a better understanding of them and the subsequent application of Machine learning techniques.

Chapter 4 presents the prototype made to SAMUR-PC with the explanation of the architecture of the application, as well as the technologies used.

Chapter 5 shows the main conclusions of this work, the future lines of research and the publications derived from this work.

Chapters 6 and 7 are the English translations of the Introduction and Conclusions.

Chapter 8 contains the individual contributions of the students.

6.6. Languages and Technologies used

During the project, Python version 3.7 has been used at all times as the main language, on different work environments. As for the creation of the web application, it was started using PHP but it was changed to Javascript, NodeJs and ExpressJs.

1. **Python3**

   Python is an interpreted programming language managed by the Python Software Foundation. Its use in Machine learning is very extended since it has libraries that help the treatment (Pandas, Scikit-Learn) and visualization (Matplotlib, Seaborn) of the data, besides others that provide prediction algorithms (Scikit-Learn & Tensorflow).

2. **PHP**

   It is a general purpose server-side programming language. It is used in conjunction with HTML to develop dynamic web content.

3. **Javascript**

   It is a light, interpreted language, better known as the scripting language for web pages, but also used in many non-browser environments. It is used both in FrontEnd to create dynamic web pages and in Backend (Node.js) to implement the server side.

4. **Jupyter Notebook**

   Jupyter Notebook is a web application based on notebooks formed by code fragments grouped in cells and text in Markdown format, which allows easy documentation of the code.

5. **Google Cloud**

   This is Google's virtual space in which you can perform a series of tasks that previously required hardware or software and now use the cloud as a form of access. In this case it was used to carry out Machine Learning training at a higher speed than when training locally.

6. **Google Collaboratory**

   Google Collaboratory, also called Colab, allows you to run and program Python in your browser, accessing GPUs for free, for faster execution.

7. **NodeJs**

   It is a JavaScript execution environment. Where Node.js really shines is in the creation of fast network applications, as it is able to handle a large amount of simultaneous connections with a high level of performance, which equates to high scalability.

8. **ExpressJs**

   Express.js is a framework for Node.js that serves to help us create web applications in less time as it provides us with features such as routing, session management options and cookies...

9. **PowerBI**

   Power BI is a Microsoft business analytics service, aimed at providing interactive visualizations and business intelligence capabilities with an interface simple enough for end users to create their own reports and dashboards.

10. **Latex**

    Latex is a text composition system, oriented to the creation of written research documents with a high typographic quality. It was selected as a tool for writing this report due to the investigative nature of the project.

10. **Docker**

    Docker is a container creation technology that enables the creation and use of Linux containers. You can use the containers as extremely lightweight and modular virtual machines. The main advantages are simplicity and faster configurations, it is not necessary to install in the computer all the technologies that a project uses, with having the project in a Docker container, it is enough to download that container, in addition, Docker manages to reduce the implementation to seconds; this is due to the fact that it creates a container for each process and does not boot an operating system.

11. **Trello**

    Trello is a project management software with web interface (also has application for mobile devices). It allows you to organize dashboards with different lists and tasks, assign tasks to different people, set alarms or dates along with the tasks, etc. Below is a snapshot of the task panel used in the project:

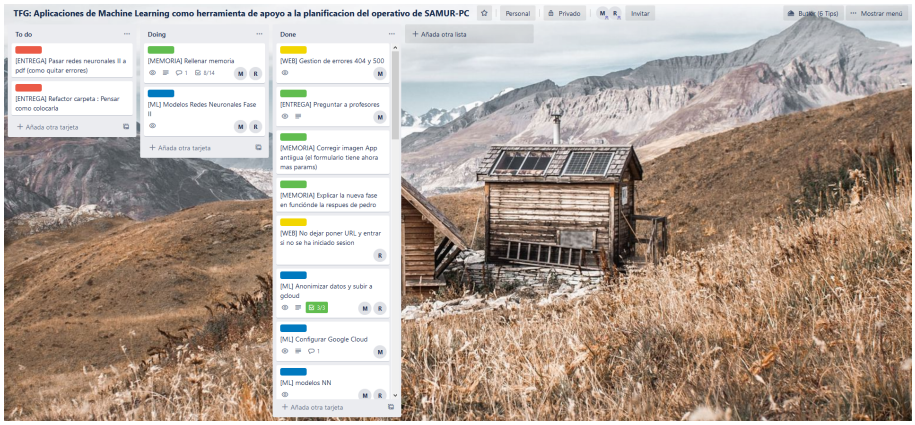


Figura 6.1: Task pane

6.7. Repository

This project is published in a GitHub repository, since the repository cannot be in public for confidentiality reasons, a folder has been created at Google Drive with the content of this project. The repository is divided into three parts:

- **App:** Directory containing the final web prototype made during the project. In it you will find the Docker configuration to execute the prototype in an easy way.
- **Prototipo:** Directory containing the code developed as the first version of the web prototype shown to SAMUR - Civil Protection
- **Documentación:** Directory containing all documentation files made during the project, including:
  - User manual for running web application
  - Pdf documents of the jupyter notebooks created during the project.
- **demo.ekv:** Contains a video showing the developed page in case it is not possible to start it up.

6.8. Applied degree subjects

During the course of the degree, different subjects have been studied that have helped to acquire transversal knowledge with which we have been able to carry out this project.

1. **Aprendizaje automático & Big Data (AA)**

   It is the subject through which the necessary knowledge has been acquired to carry out the activities related to Machine Learning and handling of large volumes of data.

2. **Aplicaciones Web (AW)**

   Thanks to which it has been possible to apply knowledge of web development based on HTML, PHP, NodeJs and ExpressJs to realize the web prototype that supports the models of Machine Learning.

3. **Bases de Datos (BD)**

   This subject has allowed us to know concepts such as relational databases often used today in the world of software development. It has allowed us to manage the authentication of users in the prototype created for SAMUR - Civil Protection.

4. **Tecnología de la programación (TP)**

   Subject in which we have been able to learn the paradigm of object-oriented programming, which together with Software Engineering, Software Modeling and Web Applications, have allowed us to develop the prototype for SAMUR-PC.

5. **Ingeniería del Software y Modelado de Software (IS & MS)**

   In our opinion two of the most important subjects of the career that have helped us understand the importance of design patterns in software development and apply them to the prototype using patterns such as Controller or DAO (Data Access Object).

6. **Prácticas en empresa**

   Thanks to this optional course, we have been able to develop our work experience and learn more about the tools used in the world of Big Data, particularly in the project. We also learned how to create interactive reports with PowerBI.

7. **Estadística Aplicada (EA)**

   With this course we have acquired basic knowledge about statistics that has helped us to better understand the Machine learning techniques explained in this report.

8. **Gestión de Proyectos Software (GPS)**

   In this subject we have been able to understand the management of projects in the world of software and apply it to work, creating a panel of tasks, following models of boards characteristic of agile methodologies, in which the tasks have three states (To Do, Doing , Done).

9. **Desarrollo de Sistemas Interactivos (DSI)**

   Subject that has helped in the development of the web application in an interactive way and in a more attractive and usable way for the user.

Capítulo 7

Conclusions and Future Work

7.1. Conclusions

The objective of this work has been the application of automatic learning techniques on the data provided by SAMUR - Civil Protection to support the planning of the daily operation. After meetings with Fernando Monforte Escobar, the study focused on predicting the number of activations according to the type of device, the district or the base to which the activation is destined for a given date.

The techniques used have been Linear Regression and Neural Networks, carrying out an in-depth research in the latter to find out which is the best type of neural network to apply. Below we show a table with the final results:

Model ID	Features	Lineal R.	RN Lap I	RN Lap II
1	Nº de Mes - Nº de día del mes	63.68	51.58	57.20
2	Nº de Mes - Nº de día del mes Día de la semana	54.42	43.95	45.19
3	Nº de Mes - Nº de día del mes Día de la semana Base operativa	10.29	8.4	8.23
4	Nº de Mes - Nº de día del mes Día de la semana Distrito	8.59	6.26	6.52
5	Nº de Mes - Nº de día del mes Día de la semana Turno	41.03	26.41	23.92
6	Nº de Mes - Nº de día del mes Día de la semana Base Turno	4.18	3.71	4.16
7	Nº de Mes - Nº de día del mes Día de la semana Base Dispositivo	5.03	4.52	4.35
8	Nº de Mes - Nº de día del mes Día de la semana Distrito Dispositivo	3.48	2.37	2.41
9	Nº de Mes - Nº de día del mes Día de la semana Dispositivo	40.70	15.98	14.08
10	Nº de Mes - Nº de día del mes Día de la semana Base Turno Dispositivo	2.22	2.07	2.05

Model ID	Features	R. Lineal	RN Lap I	RN Lap II
11	Nº de Mes - Nº de día del mes Día de la semana Distrito Turno Dispositivo	1.67	1.48	1.47
12	Nº de Mes - Nº de día del mes Día de la semana Dispositivo Turno	16.31	6.64	6.45
13	Nº de Mes - Nº de día del mes Día de la semana Distrito Turno	4.03	3.33	3.38

Cuadro 7.1: Summary Table

As we can see during the project, the models made have been progressively improved. In all of them we observe that the neuronal networks obtain an improvement against linear regression, this explains the great use and importance that this technique has acquired in the last years.

This project has allowed us to acquire knowledge of automatic learning in greater depth, exploring the different types of existing neural networks, as well as acquiring transversal knowledge that we will use throughout our professional lives, such as holding meetings with stakeholders and clients to find out about the interests of a product and the process of making it.

Finally, we would like to emphasize that since this TFG is focused on research work, we also wanted to develop a prototype using the conclusions of this work to support the decision making of SAMUR - Civil Protection regarding the planning of the daily operation, adding the possibility to visualize the warnings that have occurred during the last years and the predictions made from these models for a given date and circumstances.

7.2. Future Work

As possible future works following this investigation line, there are proposed the following ones:

- **Add polynomial terms:** In order to try to improve the models obtained throughout the project, you can add polynomial terms as independent variables, that is, create new independent variables as the linear combination of the same column.
- **Explore new Neural Network configurations:** Given that one of the biggest drawbacks has been the cost in training time of the different models using neural networks, one of the next steps to be taken, would be the use of the paid Google Cloud CPU to be able to explore new configurations and continue improving the models.
- **Creation of learning curves**: A step forward, would be the creation of a learning curve in each model with the aim of knowing, if it would be interesting to obtain new data to try to improve the accuracy of the model or it has already reached the maximum accuracy with the available data.
- **SAMUR-PC Database Integration:** Another step to improve the prototype created to SAMUR - Civil Protection would be the integration of the Database of the activations to be able to feed back the PowerBI report from the web page.
- **SAMUR-PC model feedback:** One of the most important and interesting steps to continue with the project would be, after the integration of the Database, to train the models with the new warnings that occur every day to keep them updated over time and thus adjust to new needs and patterns in the data.
- **Creation of the final app:** To improve the scalability and flexibility of the application, one of the possible future works would be the migration of the web app to a PWA (Progressive Web App) so that SAMUR-PC staff would have it available on their mobile phones.
- **Meeting with the SAMUR-PC computer team to give the app and explain how it works**

Capítulo 8

Aportaciones Individuales

Las contribuciones realizadas se describen a continuación siguiendo el índice de la memoria:

8.1. Manuel Monforte Escobar

Tras conocer, gracias a Fernando Monforte Escobar, alguna de las necesidades de SAMUR-PC, como la optimización en la planificación del operativo diario, se contactó con ellos para explicar y mostrar las ventajas que presenta la aplicación de Machine Learning al caso de uso.

Tras la reunión con SAMUR-PC, y conocer el interés y el deseo de que el proyecto siguiera adelante, se realizó una búsqueda del director del proyecto. Tras contactar con varios profesores, se acordó que Pedro Antonio González Calero, profesor de la Asignatura Aprendizaje Automático & Big Data, fuese el director del proyecto.

Celebración de reuniones con los directivos de SAMUR-PC para aclarar términos burocráticos relacionados con la licencia de uso y la confidencialidad de los datos utilizados para el entrenamiento de Modelos. En esta fase, se puso en contacto al Vicedecano de Ordenación Académica (Fernando Rosa Velardo) con SAMUR-PC, para resolver y aclarar cualquier duda al respecto.

Creación y presentación de un prototipo inicial a SAMUR-PC para recibir feedback. Para ello se investigó en tecnologías de Javascript como Leaflet para generar mapas de forma dinámica. Además se estudió el lenguaje PHP para la creación del BackEnd de la aplicación y HTML5 para la creación del FrontEnd de la aplicación.

Búsqueda de información y teoría sobre LateX.

Reuniones del equipo de trabajo. Por un lado los estudiantes del proyecto, mantuvieron reuniones semanales para evitar perder el objetivo y ceñirse a la planificación lo máximo posible. También se celebraron reuniones con el director del TFG, tras la crisis sanitaria que tuvo lugar en relación al COVID-19; se mantuvo el contacto de forma semanal con el director por correo electrónico para mostrar los avances del proyecto y recibir feedback y nuevos pasos a seguir.

Investigación sobre crear informes de forma dinámica con lenguaje Javascript y tecnologías Web para el prototipo final de SAMUR-PC.

Creación de informe PowerBI para la visualización de los datos históricos cedidos por SAMUR-PC. En esta apartado se incluye la investigación de la tecnología PowerBI así como el estudio para integrar el informe en la aplicación Web y el contacto con diferentes compañeros de trabajo de visualización. Entre ellos, Sergio Valero García para la realización del informe.

Creación del prototipo final (generador de informes) para SAMUR-PC, tras consultar con Fernando Monforte Escobar, stakeholder y representante de SAMUR-PC a lo largo del trabajo. Se realizaron pruebas para integrar scripts de python (lenguaje utilizado en las técnicas de Machine Learning) y Javscript(lenguaje utilizado en el BackEnd del prototipo).

Realización de procesos ETL para preparar los datos para la elaboración de gráficas. Como se ha comentado en apartados anteriores, incluye, creación de archivos Excel para unificar datos, cambiar tipo de datos en columnas, corregir datos atípicos y unificarlos...

Realización de gráficas y descripción de las mismas para mostrar patrones o tendencias en las activaciones de SAMUR-PC durante los años 2009-2019.

Realización de los modelos 01-07 de Regresión lineal introduciendo técnicas como validación cruzada por si el conjunto de test seleccionado no fuera representativo.

Realización de los modelos 01-07 de Redes neuronales, correspondientes a la primera fase, es decir, desarrollo en local. Estudiando previamente la librería utilizada, en este caso, Keras con el framework Tensorflow.

Estudio de herramientas Cloud para poder crear redes neuronales de mayor tamaño y con distintas configuraciones. Para ello se intentó configurar las máquinas de Google Cloud no gratuitas aprovechando el saldo ofrecido por la ucm a los estudiantes. Tras no poder configurar, se indagó y se seleccionó Google Colaboratory para realizar esta segunda fase de Redes neuronales.

Anonimización de datos de SAMUR-PC para poder entrenar los modelos en Google Cloud. En este apartado se incluye el estudio de la API de Google Drive (PyDrive) para poder utilizar desde los cuadernos de Colaboratory los datos anonimizados alojados en Google Drive.

Realización de modelos 01-06, 11, 13 de Redes Neuronales , correspondientes a la segunda fase, es decir, utilizando los procesadores de Google. Aplicando nuevos valores, como aumentando número de capas, parámetros de regularización y número de nodos en cada una de las capas.

Integración de modelos en el prototipo creado para SAMUR-PC. Para ello se estudió la forma de exportar los modelos creados, indagando en librerías como pytorch o pickle y poder importarlos mediante un script de python.

Investigación sobre tecnologías para despliegue rápido y sencillo como Docker, utilizada para facilitar al tribunal la puesta en ejecución del prototipo web, acompañado por un manual de usuario.

Creación de los contenedores Docker y creación del archivo Docker compose, para

relacionar la base de datos situada en un contenedor y la propia aplicación situado en otro.

Traducción de los apartados de Introducción y Conclusiones de la presente memoria.

Elaboración del apartado conclusiones y trabajo a futuro de la presente memoria

Las últimas semanas del proyecto fueron dedicadas a la corrección de la memoria desarrollada hasta el momento, asegurándose de el correcto cumplimiento de la normativa vigente en relación a los TFG.

Durante todo el desarrollo del TFG (desde la proposición de la idea a SAMUR-PC en Abril 2019, hasta la entrega del mismo en Junio 2020), se ha realizado una continúa investigación en relación a todos los frameworks, lenguajes, documentos sobre Machine Learning, visualización de datos... Es decir ha habido un proceso iterativo constante de Idea, Experimentación y Aprendizaje.

8.2. Raúl Vindel Loeches

Tras conocer, gracias a Fernando Monforte Escobar, alguna de las necesidades de SAMUR-PC, como la optimización en la planificación del operativo diario, se contactó con ellos para explicar y mostrar las ventajas que presenta la aplicación de Machine Learning al caso de uso.

Tras la reunión con SAMUR-PC, y conocer el interés y el deseo de que el proyecto siguiera adelante, se realizó una búsqueda del director del proyecto. Tras contactar con varios profesores, se acordó que Pedro Antonio González Calero, profesor de la Asignatura Aprendizaje Automático & Big Data, fuese el director del proyecto.

Celebración de reuniones con los directivos de SAMUR-PC para aclarar términos burocráticos relacionados con la licencia de uso y la confidencialidad de los datos utilizados para el entrenamiento de Modelos. En esta fase, se puso en contacto al Vicedecano de Ordenación Académica (Fernando Rosa Velardo) con SAMUR-PC, para resolver y aclarar cualquier duda al respecto.

Creación y presentación de un prototipo inicial a SAMUR-PC para recibir feedback. Para ello se investigó en tecnologías de Javascript como Leaflet para generar mapas de forma dinámica. Además se estudió el lenguaje PHP para la creación del BackEnd de la aplicación y HTML5 para la creación del FrontEnd de la aplicación.

Realización de login y formularios iniciales de forma segura gracias a través de los cuales se llegaría a visualizar el resultado de la ejecución del modelo correcto utilizando PHP y HTML5.

Búsqueda de información y teoría sobre LateX, así como la realización de un breve curso con la finalidad de realizar esta memoria de forma más óptima. Además de apuntarse a un curso posterior que ofreció un alumno en la Facultad de Informática de nuestra Universidad.

Reuniones del equipo de trabajo. Por un lado los estudiantes del proyecto, mantuvieron reuniones semanales para evitar perder el objetivo y ceñirse a la planificación lo máximo posible. También se celebraron reuniones con el director del TFG, tras la crisis sanitaria que tuvo lugar en relación al COVID-19; se mantuvo el contacto de forma semanal con el director por correo electrónico para mostrar los avances del proyecto y recibir feedback y nuevos pasos a seguir.

Investigación sobre crear informes de forma dinámica con lenguaje Javascript y tecnologías Web para el prototipo final de SAMUR-PC.

Realización de procesos ETL para preparar los datos para la elaboración de gráficas. Como se ha comentado en apartados anteriores, incluye, creación de archivos Excel para unificar datos, cambiar tipo de datos en columnas, corregir datos atípicos y unificarlos...

Realización de gráficas y descripción de las mismas para mostrar patrones o tendencias en las activaciones de SAMUR-PC durante los años 2009-2019.

Realización de los modelos 07-13 de Regresión lineal introduciendo técnicas como validación cruzada por si el conjunto de test seleccionado no fuera representativo.

Realización de los modelos 07-13 de Redes neuronales, correspondientes a la primera fase, es decir, desarrollo en local. Estudiando previamente la libreria utilizada, en este caso, Keras con el framework Tensorflow.

Búsqueda de información sobre Aprendizaje Automático y Redes Neuronales con el objetivo de encontrar opciones para mejorar los resultados obtenidos hasta la fecha. Así como para comprobar la correcta realización de las técnicas de entrenamiento y validación utilizadas a la hora de entrenar los distintos modelos de tal forma que estos no realizaran ningún tipo de sobreajuste (Overfitting).

Estudio de herramientas Cloud para poder crear redes neuronales de mayor tamaño y con distintas configuraciones de manera más rápida y eficaz. Para ello se intentó configurar las máquinas de Google Cloud no gratuitas aprovechando el saldo ofrecido por la ucm a los estudiantes. Tras varias dificultades a la hora de configurar el entorno, y cobros que se realizaron a la cuenta personal debido al uso del mismo, se indagó y se seleccionó Google Colaboratory para realizar esta segunda fase de Redes neuronales.

Petición a SAMUR-PC para poder subir los datos proporcionado a Google Drive, desde donde se podrían acceder para usarlos con Google Colaboratory.

SAMUR-PC, exigió la anonimización de datos de para poder subir y entrenar los modelos en Google Cloud. En este apartado se incluye el estudio de la API de Google Drive (PyDrive) para poder utilizar desde los cuadernos de Colaboratory los datos anonimizados alojados en Google Drive.

Realización de modelos 07-0x de Redes Neuronales , correspondientes a la segunda fase, es decir, utilizando los procesadores de Google. Aplicando nuevos valores, como aumentando número de capas, parámetros de regularización y número de nodos en cada una de las capas.

Integración de modelos en el prototipo creado para SAMUR-PC. Para ello se estudió la forma de exportar los modelos creados, indagando en librerias como pytorch o pickle y poder importarlos mediante un script de python.

Investigación sobre tecnologías para despliegue rápido y sencillo como Docker, utilizada para facilitar al tribunal la puesta en ejecución del prototipo web.

Traducción de los apartados de Introducción y Conclusiones de la presente memoria.

Elaboración del apartado conclusiones y trabajo a futuro de la presente memoria

Comprobación del correcto funcionamiento de Docker siguiendo el manual de usuario realizado para el personal de SAMUR-PC.

Las últimas semanas del proyecto fueron dedicadas a la corrección de la memoria desarrollada hasta el momento, asegurándose de el correcto cumplimiento de la normativa vigente en relación a los TFG.

Durante todo el desarrollo del TFG (desde la proposición de la idea a SAMUR-PC en Abril 2019, hasta la entrega del mismo en Junio 2020), se ha realizado una continúa investigación en relación a todos los frameworks, lenguajes, documentos sobre Machine

Learning, visualización de datos... Es decir ha habido un proceso iterativo constante de Idea, Experimentación y Aprendizaje.

Bibliografía

[1] George M Foster, Barbara Gallatin Anderson, et al. *Medical anthropology.* John Wiley & Sons, Inc. 605 3rd Avenue, New York, NY 10016, USA, 1978.

[2] Arden Dertat. Applied deep learning-part 1: Artificial neural networks. *Towards Data Science*, 8(08), 2017.

[3] Antonio Gulli and Sujit Pal. *Deep learning with Keras.* Packt Publishing Ltd, 2017.

[4] Jason Brownlee. Regression tutorial with keras deep learning library in python, 2016.

[5] Nikhil Ketkar. Introduction to keras. In *Deep learning with Python*, pages 97–111. Springer, 2017.

[6] François Chollet. *Deep Learning with Python.*

[7] Stacey Ronaghan. Deep learning: Which loss and activation functions should i use. 2018.

[8] Google Cloud. Primeros pasos: Entrenamiento y predicción con keras. Enlace: https://cloud.google.com/ai-platform/docs/getting-started-keras?authuser=1.

[9] Google. Google colaboratory. Enlace: https://colab.research.google.com/.

[10] Michael D. Harris. Colab and google sheets.

[11] Pedro Antonio Gonzalez Calero. Apuntes asignatura aprendizaje automático & big data.

[12] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems*, pages 6389–6399, 2018.

[13] Jason Brownlee. How to get started with deep learning for time series forecasting (7-day mini-course).

[14] Justin A Boyan and Michael L Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. In *Advances in neural information processing systems*, pages 671–678, 1994.

[15] Andrew Ng. Machine learning, standford university. Enlace: https://www.coursera.org/learn/machine-learning?ranMID=40328&ranEAID=hL3Qp0zRBOc&ranSiteID=hL3Qp0zRBOc-d3fGJLeHxFpeaycpoFpsdQ&siteID=hL3Qp0zRBOc-d3fGJLeHxFpeaycpoFpsdQ&utm_content=10&utm_medium=partners&utm_source=linkshare&utm_campaign=hL3Qp0zRBOc.

[16] Andriy Burkov. *The hundred-page machine learning book.* Andriy Burkov Quebec City, Can., 2019.

[17] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn TensorFlow.* Enlace: https://www.lpsm.paris/pageperso/has/source/Hand-on-ML.pdf.

[18] François Chollet. Sequence processing with convnets. Enlace: https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/6.4-sequence-processing-with-convnets.ipynb.

Capítulo 9

Anexo

En este capítulo se adjuntan toda la documentación realizada durante el proyecto.

Transformación Datos SAMUR- Protección Civil

Propósito del cuaderno

El objetivo del cuaderno es documentar el proceso ETL, es decir, los procesos de extracción, transformación y carga de datos aplicados a SAMUR-PC para:

- Creación de gráficas en el cuaderno Exploración.ipynb
- Anonimización de datos para la utilziación de los mismos de Redes Neuronales Cloud.ipynb

Librerias

In [1]:

```
import pandas as pd
from matplotlib import pyplot as plt
import numpy as np
import datetime
```

1. Creación de un único archivo xlsx a partir de las hojas de un xlsx

2009

In [1]:

```
import pandas as pd

# filenames
excel_names = ["samur2009_1.xlsx", "samur2009_2.xlsx", "samur2009_3.xlsx"]

# read them in
excels = [pd.ExcelFile(name) for name in excel_names]

# turn them into dataframes
frames = [x.parse(x.sheet_names[0], header=None,index_col=None) for x in excels]

# delete the first row for all frames except the first
# i.e. remove the header row -- assumes it's the first
frames[1:] = [df[1:] for df in frames[1:]]

# concatenate them..
combined = pd.concat(frames)

# write it out
combined.to_excel("samur2009.xlsx", header=False, index=False)
```

2010-2018

In [19]:

```
for i in [10,11,12,13,14,15,16,17,18] :
    # filenames
    excel_names = ["samur20"+str(i)+"_1.xlsx", "samur20"+str(i)+"_2.xlsx", "samur20"+str(i)+"_3.xlsx"]

    # read them in
    excels = [pd.ExcelFile(name) for name in excel_names]

    # turn them into dataframes
    frames = [x.parse(x.sheet_names[0], header=None,index_col=None) for x in excels]

    # delete the first row for all frames except the first
    # i.e. remove the header row -- assumes it's the first
    frames[1:] = [df[1:] for df in frames[1:]]

    # concatenate them..
    combined = pd.concat(frames)

    # write it out
    combined.to_excel("samur20"+str(i)+".xlsx", header=False, index=False)
```

2019

In [0]:

```
for i in [19] :
    # filenames
    excel_names = ["samur20"+str(i)+"_1.xlsx", "samur20"+str(i)+"_2.xlsx"]

    # read them in
    excels = [pd.ExcelFile(name) for name in excel_names]

    # turn them into dataframes
    frames = [x.parse(x.sheet_names[0], header=None,index_col=None) for x in excels]

    # delete the first row for all frames except the first
    # i.e. remove the header row -- assumes it's the first
    frames[1:] = [df[1:] for df in frames[1:]]

    # concatenate them..
    combined = pd.concat(frames)

    # write it out
    combined.to_excel("samur20"+str(i)+".xlsx", header=False, index=False)
```

2. Creación de un único archivo xlsx para la creación de un Informe de PowerBI

In [25]:

```
# filenames
excel_names = []
for i in [15,16,17,18,19]:
    excel_names.append("processed_data/samur20" + str(i) + ".xlsx")

print(excel_names)

# read them in
excels = [pd.ExcelFile(name) for name in excel_names]

# turn them into dataframes
frames = [x.parse(x.sheet_names[0], header=None,index_col=None) for x in excels]

# delete the first row for all frames except the first
# i.e. remove the header row -- assumes it's the first
frames[1:] = [df[1:] for df in frames[1:]]

# concatenate them..
combined = pd.concat(frames)

# write it out
combined.to_excel("samur_PowerBI.xlsx", header=False, index=False)
```

```
['processed_data/samur2015.xlsx', 'processed_data/samur2016.xlsx', 'processed_data/samur2017.xlsx',
'processed_data/samur2018.xlsx', 'processed_data/samur2019.xlsx']
```

3. Trasformaciones aplicadas a los datos

Lectura de datos

In [2]:

```
alerts_2009 = pd.read_excel("data/samur2009.xlsx")
alerts_2010 = pd.read_excel("data/samur2010.xlsx")
alerts_2011 = pd.read_excel("data/samur2011.xlsx")
alerts_2012 = pd.read_excel("data/samur2012.xlsx")
alerts_2013 = pd.read_excel("data/samur2013.xlsx")
alerts_2014 = pd.read_excel("data/samur2014.xlsx")
alerts_2015 = pd.read_excel("data/samur2015.xlsx")
alerts_2016 = pd.read_excel("data/samur2016.xlsx")
alerts_2017 = pd.read_excel("data/samur2017.xlsx")
alerts_2018 = pd.read_excel("data/samur2018.xlsx")
alerts_2019 = pd.read_excel("data/samur2019.xlsx")
```

Eliminar espacios de la columna DIA_SEMANA

In [3]:

```
alerts_2009["DIA_SEMANA"] = alerts_2009["DIA_SEMANA"].str.strip()
alerts_2010["DIA_SEMANA"] = alerts_2010["DIA_SEMANA"].str.strip()
alerts_2011["DIA_SEMANA"] = alerts_2011["DIA_SEMANA"].str.strip()
alerts_2012["DIA_SEMANA"] = alerts_2012["DIA_SEMANA"].str.strip()
alerts_2013["DIA_SEMANA"] = alerts_2013["DIA_SEMANA"].str.strip()
alerts_2014["DIA_SEMANA"] = alerts_2014["DIA_SEMANA"].str.strip()
alerts_2015["DIA_SEMANA"] = alerts_2015["DIA_SEMANA"].str.strip()
alerts_2016["DIA_SEMANA"] = alerts_2016["DIA_SEMANA"].str.strip()
alerts_2017["DIA_SEMANA"] = alerts_2017["DIA_SEMANA"].str.strip()
alerts_2018["DIA_SEMANA"] = alerts_2018["DIA_SEMANA"].str.strip()
alerts_2019["DIA_SEMANA"] = alerts_2019["DIA_SEMANA"].str.strip()
```

Crear nueva columna con las bases corregidas

In [0]:

```
alerts_2009["BASE_CORREGIDA"]=alerts_2009["BASE"].fillna(-1).astype({"BASE": int })
alerts_2009["BASE_CORREGIDA"].loc[(alerts_2009['BASE'] > 23)] = -1

alerts_2010["BASE_CORREGIDA"]=alerts_2010["BASE"].fillna(-1).astype({"BASE": int })
alerts_2010["BASE_CORREGIDA"].loc[(alerts_2010['BASE'] > 23)] = -1

alerts_2011["BASE_CORREGIDA"]=alerts_2011["BASE"].fillna(-1).astype({"BASE": int })
alerts_2011["BASE_CORREGIDA"].loc[(alerts_2011['BASE'] > 23)] = -1

alerts_2012["BASE_CORREGIDA"]=alerts_2012["BASE"].fillna(-1).astype({"BASE": int })
alerts_2012["BASE_CORREGIDA"].loc[(alerts_2012['BASE'] > 23)] = -1

alerts_2013["BASE_CORREGIDA"]=alerts_2013["BASE"].fillna(-1).astype({"BASE": int })
alerts_2013["BASE_CORREGIDA"].loc[(alerts_2013['BASE'] > 23)] = -1

alerts_2014["BASE_CORREGIDA"]=alerts_2014["BASE"].fillna(-1).astype({"BASE": int })
alerts_2014["BASE_CORREGIDA"].loc[(alerts_2014['BASE'] > 23)] = -1

alerts_2015["BASE_CORREGIDA"]=alerts_2015["BASE"].fillna(-1).astype({"BASE": int })
alerts_2015["BASE_CORREGIDA"].loc[(alerts_2015['BASE'] > 23)] = -1

alerts_2016["BASE_CORREGIDA"]=alerts_2016["BASE"].fillna(-1).astype({"BASE": int })
alerts_2016["BASE_CORREGIDA"].loc[(alerts_2016['BASE'] > 23)] = -1

alerts_2017["BASE_CORREGIDA"]=alerts_2017["BASE"].fillna(-1).astype({"BASE": int })
alerts_2017["BASE_CORREGIDA"].loc[(alerts_2017['BASE'] > 23)] = -1

alerts_2018["BASE_CORREGIDA"]=alerts_2018["BASE"].fillna(-1).astype({"BASE": int })
alerts_2018["BASE_CORREGIDA"].loc[(alerts_2018['BASE'] > 23)] = -1

alerts_2019["BASE_CORREGIDA"]=alerts_2019["BASE"].fillna(-1).astype({"BASE": int })
alerts_2019["BASE_CORREGIDA"].loc[(alerts_2019['BASE'] > 23)] = -1
```

Crear nueva columna para codificar DIA_SEMANA

Hay que tener cuiadado ya que los últimos avisos de cada día en la columna de DIA_SEMANA se les asigna el día siuguiente, es decir, si el aviso fue 01/01/2009, fue Jueves, pues en día de la semana pone en los últimos avisos de ese día Viernes. Tras consultar con Fernando Monforte Escobar, se decidió mantener el día correspondiente a ENTRADA_AVISO, en este caso, todos a Jueves

In [5]:

```
alerts_2009['DIA_CODIFICADO']= pd.to_datetime(alerts_2009['ENTRADA_AVISO']).dt.dayofweek
alerts_2010['DIA_CODIFICADO']= pd.to_datetime(alerts_2010['ENTRADA_AVISO']).dt.dayofweek
alerts_2011['DIA_CODIFICADO']= pd.to_datetime(alerts_2011['ENTRADA_AVISO']).dt.dayofweek
alerts_2012['DIA_CODIFICADO']= pd.to_datetime(alerts_2012['ENTRADA_AVISO']).dt.dayofweek
alerts_2013['DIA_CODIFICADO']= pd.to_datetime(alerts_2013['ENTRADA_AVISO']).dt.dayofweek
alerts_2014['DIA_CODIFICADO']= pd.to_datetime(alerts_2014['ENTRADA_AVISO']).dt.dayofweek
alerts_2015['DIA_CODIFICADO']= pd.to_datetime(alerts_2015['ENTRADA_AVISO']).dt.dayofweek
alerts_2016['DIA_CODIFICADO']= pd.to_datetime(alerts_2016['ENTRADA_AVISO']).dt.dayofweek
alerts_2017['DIA_CODIFICADO']= pd.to_datetime(alerts_2017['ENTRADA_AVISO']).dt.dayofweek
alerts_2018['DIA_CODIFICADO']= pd.to_datetime(alerts_2018['ENTRADA_AVISO']).dt.dayofweek
alerts_2019['DIA_CODIFICADO']= pd.to_datetime(alerts_2019['ENTRADA_AVISO']).dt.dayofweek
```

Crear nueva columna dia

In [6]:

```
alerts_2009['DIA'] = pd.to_datetime(alerts_2009['ENTRADA_AVISO']).dt.day
alerts_2010['DIA'] = pd.to_datetime(alerts_2010['ENTRADA_AVISO']).dt.day
alerts_2011['DIA'] = pd.to_datetime(alerts_2011['ENTRADA_AVISO']).dt.day
alerts_2012['DIA'] = pd.to_datetime(alerts_2012['ENTRADA_AVISO']).dt.day
alerts_2013['DIA'] = pd.to_datetime(alerts_2013['ENTRADA_AVISO']).dt.day
alerts_2014['DIA'] = pd.to_datetime(alerts_2014['ENTRADA_AVISO']).dt.day
alerts_2015['DIA'] = pd.to_datetime(alerts_2015['ENTRADA_AVISO']).dt.day
alerts_2016['DIA'] = pd.to_datetime(alerts_2016['ENTRADA_AVISO']).dt.day
alerts_2017['DIA'] = pd.to_datetime(alerts_2017['ENTRADA_AVISO']).dt.day
alerts_2018['DIA'] = pd.to_datetime(alerts_2018['ENTRADA_AVISO']).dt.day
alerts_2019['DIA'] = pd.to_datetime(alerts_2019['ENTRADA_AVISO']).dt.day
```

Codificar los turnos

In [7]:

```
turn_dictionary ={'M' : 0, 'T' : 1, 'N' : 2}
alerts_2009['TURNO_CODIFICADO'] = alerts_2009['TURNO'].map(turn_dictionary)
alerts_2010['TURNO_CODIFICADO'] = alerts_2010['TURNO'].map(turn_dictionary)
alerts_2011['TURNO_CODIFICADO'] = alerts_2011['TURNO'].map(turn_dictionary)
alerts_2012['TURNO_CODIFICADO'] = alerts_2012['TURNO'].map(turn_dictionary)
alerts_2013['TURNO_CODIFICADO'] = alerts_2013['TURNO'].map(turn_dictionary)
alerts_2014['TURNO_CODIFICADO'] = alerts_2014['TURNO'].map(turn_dictionary)
alerts_2015['TURNO_CODIFICADO'] = alerts_2015['TURNO'].map(turn_dictionary)
alerts_2016['TURNO_CODIFICADO'] = alerts_2016['TURNO'].map(turn_dictionary)
alerts_2017['TURNO_CODIFICADO'] = alerts_2017['TURNO'].map(turn_dictionary)
alerts_2018['TURNO_CODIFICADO'] = alerts_2018['TURNO'].map(turn_dictionary)
alerts_2019['TURNO_CODIFICADO'] = alerts_2019['TURNO'].map(turn_dictionary)
```

Codificar los dispositivos

In [8]:

```
device_dictionary ={
    'SVB' : 0,
    'SVA' : 1,
    'PC' : 2,
    'VPC': 3,
    'SVBS': 4,
    'CRE': 5,
    'CSA': 6,
    'DEPA': 7,
    'DIRC': 8,
    'ECO': 9,
    'UN': 10,
    'MPI': 11,
    'MUS': 12,
    'PCM': 13,
    'PRI': 14,
    'RDRP': 15,
    'REM': 16,
    'ROM': 17,
    'SPS': 18,
    'SUM': 19,
    'SVI': 20,
    'TDIR': 21,
    'TPM': 22,
    'TPP': 23,
    'UPR': 24,
    'URE': 25,
    'VCO': 26,
    'VDI': 27,
    'VDIR': 28,
    'VGE': 29,
    'VPC': 30,
    'YAN': 31
}

alerts_2009['ABREV_CODIFICADO'] = alerts_2009['ABREV'].str.strip().map(device_dictionary)
alerts_2010['ABREV_CODIFICADO'] = alerts_2010['ABREV'].str.strip().map(device_dictionary)
alerts_2011['ABREV_CODIFICADO'] = alerts_2011['ABREV'].str.strip().map(device_dictionary)
alerts_2012['ABREV_CODIFICADO'] = alerts_2012['ABREV'].str.strip().map(device_dictionary)
alerts_2013['ABREV_CODIFICADO'] = alerts_2013['ABREV'].str.strip().map(device_dictionary)
alerts_2014['ABREV_CODIFICADO'] = alerts_2014['ABREV'].str.strip().map(device_dictionary)
alerts_2015['ABREV_CODIFICADO'] = alerts_2015['ABREV'].str.strip().map(device_dictionary)
alerts_2016['ABREV_CODIFICADO'] = alerts_2016['ABREV'].str.strip().map(device_dictionary)
alerts_2017['ABREV_CODIFICADO'] = alerts_2017['ABREV'].str.strip().map(device_dictionary)
alerts_2018['ABREV_CODIFICADO'] = alerts_2018['ABREV'].str.strip().map(device_dictionary)
alerts_2019['ABREV_CODIFICADO'] = alerts_2019['ABREV'].str.strip().map(device_dictionary)
```

Anonimizar datos

In [22]:

```
columns = ["AÑO", "SUCESO", "MES", "DI", "BA_CO","DIA_CODIFICADO","DIA", "T_CO", "AB_CO"]
```

In [23]:

```
alerts_2009.rename(columns={'AÑO':'AÑO',"SUCESO":"SUCESO","MES":"MES", 'DISTRITO':'DI', 'BASE_CORREGIDA':'BA_CO',
"DIA_CODIFICADO":"DIA_CODIFICADO", 'ABREV_CODIFICADO':'AB_CO', "DIA":"DIA","TURNO_CODIFICADO":"T_CO"}, inplace=Tr
ue)
alerts_2010.rename(columns={'AÑO':'AÑO',"SUCESO":"SUCESO","MES":"MES", 'DISTRITO':'DI', 'BASE_CORREGIDA':'BA_CO',
"DIA_CODIFICADO":"DIA_CODIFICADO", 'ABREV_CODIFICADO':'AB_CO', "DIA":"DIA","TURNO_CODIFICADO":"T_CO" }, inplace=T
rue)
alerts_2011.rename(columns={'AÑO':'AÑO',"SUCESO":"SUCESO","MES":"MES", 'DISTRITO':'DI', 'BASE_CORREGIDA':'BA_CO',
"DIA_CODIFICADO":"DIA_CODIFICADO", 'ABREV_CODIFICADO':'AB_CO', "DIA":"DIA","TURNO_CODIFICADO":"T_CO" }, inplace=T
rue)
alerts_2012.rename(columns={'AÑO':'AÑO',"SUCESO":"SUCESO","MES":"MES", 'DISTRITO':'DI', 'BASE_CORREGIDA':'BA_CO',
"DIA_CODIFICADO":"DIA_CODIFICADO", 'ABREV_CODIFICADO':'AB_CO', "DIA":"DIA","TURNO_CODIFICADO":"T_CO" }, inplace=T
rue)
alerts_2013.rename(columns={'AÑO':'AÑO',"SUCESO":"SUCESO","MES":"MES", 'DISTRITO':'DI', 'BASE_CORREGIDA':'BA_CO',
"DIA_CODIFICADO":"DIA_CODIFICADO", 'ABREV_CODIFICADO':'AB_CO', "DIA":"DIA","TURNO_CODIFICADO":"T_CO" }, inplace=T
rue)
alerts_2014.rename(columns={'AÑO':'AÑO',"SUCESO":"SUCESO","MES":"MES", 'DISTRITO':'DI', 'BASE_CORREGIDA':'BA_CO',
"DIA_CODIFICADO":"DIA_CODIFICADO", 'ABREV_CODIFICADO':'AB_CO', "DIA":"DIA","TURNO_CODIFICADO":"T_CO" }, inplace=T
rue)
alerts_2015.rename(columns={'AÑO':'AÑO',"SUCESO":"SUCESO","MES":"MES", 'DISTRITO':'DI', 'BASE_CORREGIDA':'BA_CO',
"DIA_CODIFICADO":"DIA_CODIFICADO", 'ABREV_CODIFICADO':'AB_CO', "DIA":"DIA","TURNO_CODIFICADO":"T_CO" }, inplace=T
rue)
alerts_2016.rename(columns={'AÑO':'AÑO',"SUCESO":"SUCESO","MES":"MES", 'DISTRITO':'DI', 'BASE_CORREGIDA':'BA_CO',
"DIA_CODIFICADO":"DIA_CODIFICADO", 'ABREV_CODIFICADO':'AB_CO', "DIA":"DIA","TURNO_CODIFICADO":"T_CO" }, inplace=T
rue)
alerts_2017.rename(columns={'AÑO':'AÑO',"SUCESO":"SUCESO","MES":"MES", 'DISTRITO':'DI', 'BASE_CORREGIDA':'BA_CO',
"DIA_CODIFICADO":"DIA_CODIFICADO", 'ABREV_CODIFICADO':'AB_CO', "DIA":"DIA","TURNO_CODIFICADO":"T_CO" }, inplace=T
rue)
alerts_2018.rename(columns={'AÑO':'AÑO',"SUCESO":"SUCESO","MES":"MES", 'DISTRITO':'DI', 'BASE_CORREGIDA':'BA_CO',
"DIA_CODIFICADO":"DIA_CODIFICADO", 'ABREV_CODIFICADO':'AB_CO', "DIA":"DIA","TURNO_CODIFICADO":"T_CO" }, inplace=T
rue)
alerts_2019.rename(columns={'AÑO':'AÑO',"SUCESO":"SUCESO","MES":"MES", 'DISTRITO':'DI', 'BASE_CORREGIDA':'BA_CO',
"DIA_CODIFICADO":"DIA_CODIFICADO", 'ABREV_CODIFICADO':'AB_CO', "DIA":"DIA","TURNO_CODIFICADO":"T_CO" }, inplace=T
rue)
```

Guardar los datos

Datos no anonimizados para desarrollos en local

In [6]:

```
alerts_2009.to_excel("data/samur2009.xlsx", index= False)
alerts_2010.to_excel("data/samur2010.xlsx", index= False)
alerts_2011.to_excel("data/samur2011.xlsx", index= False)
alerts_2012.to_excel("data/samur2012.xlsx", index= False)
alerts_2013.to_excel("data/samur2013.xlsx", index= False)
alerts_2014.to_excel("data/samur2014.xlsx", index= False)
alerts_2015.to_excel("data/samur2015.xlsx", index= False)
alerts_2016.to_excel("data/samur2016.xlsx", index= False)
alerts_2017.to_excel("data/samur2017.xlsx", index= False)
alerts_2018.to_excel("data/samur2018.xlsx", index= False)
alerts_2019.to_excel("data/samur2019.xlsx", index= False)
```

Datos anonimizados para desarrollos en cloud

In [25]:

```
alerts_2009[columns].to_excel("data/c2009.xlsx", index= False)
alerts_2010[columns].to_excel("data/c2010.xlsx", index= False)
alerts_2011[columns].to_excel("data/c2011.xlsx", index= False)
alerts_2012[columns].to_excel("data/c2012.xlsx", index= False)
alerts_2013[columns].to_excel("data/c2013.xlsx", index= False)
alerts_2014[columns].to_excel("data/c2014.xlsx", index= False)
alerts_2015[columns].to_excel("data/c2015.xlsx", index= False)
alerts_2016[columns].to_excel("data/c2016.xlsx", index= False)
alerts_2017[columns].to_excel("data/c2017.xlsx", index= False)
alerts_2018[columns].to_excel("data/c2018.xlsx", index= False)
alerts_2019[columns].to_excel("data/c2019.xlsx", index= False)
```

Exploración Datos SAMUR - Proctección Civil

Propósito del cuaderno

El objetivo de este documento es la elaboración de gráficas en relación a los datos de SAMUR-PC para intentar descubrir tendencias o patrones en los avisos o activaciones bajo unas circunstancias concretas.

Para ello, se llevará a cabo un análisis exploratorio de los datos siguiendo las principales predicciones solicitadas por SAMUR-PC para obtener una mayor comprensión de los datos a fin de crear modelos de predicción. Las predicciones solicitadas fueron las siguientes:

- Número de avisos para un día determinado
- Número de avisos por tipo de dispositivo para una fecha determinada
- Número de avisos por distrito para una fecha determinada
- ¿Cuál es la relación entre el número de avisos entre los fines de semana y los días de semana?
- Número de avisos por turno en uno o todos los días de la semana
- Número de avisos por código para un día determinado
- Número de avisos por base

Librerías

In [1]:

```
import pandas as pd
from matplotlib import pyplot as plt
import numpy as np
from matplotlib import rc
from mpl_toolkits.mplot3d import Axes3D
```

Obtenemos los datos

In [2]:

```
alerts_2009 = pd.read_excel("data/samur2009.xlsx")
alerts_2010 = pd.read_excel("data/samur2010.xlsx")
alerts_2011 = pd.read_excel("data/samur2011.xlsx")
alerts_2012 = pd.read_excel("data/samur2012.xlsx")
alerts_2013 = pd.read_excel("data/samur2013.xlsx")
alerts_2014 = pd.read_excel("data/samur2014.xlsx")
alerts_2015 = pd.read_excel("data/samur2015.xlsx")
alerts_2016 = pd.read_excel("data/samur2016.xlsx")
alerts_2017 = pd.read_excel("data/samur2017.xlsx")
alerts_2018 = pd.read_excel("data/samur2018.xlsx")
alerts_2019 = pd.read_excel("data/samur2019.xlsx")
alerts_allyears = pd.concat([alerts_2009,alerts_2010, alerts_2011, alerts_2012, alerts_2013, alerts_2014,
                             alerts_2015, alerts_2016, alerts_2017, alerts_2018, alerts_2019], ignore_index=True)
```

In [3]:

```
alerts_allyears.head(2)
```

Out[3]:

	AÑO	SUCESO	INFORME	PACIENTE	MES	DIA_SEMANA	FECHA_FIN	CODIGO_INICIAL	CODIGO_FINAL	BASE	...	CLAVE_3	CL
0	2009	1	1	1	1	Jueves	01/01/2009	2.3	3.9	1.0	...	2009-01-01 00:09:44	2
1	2009	2	2	1	1	Jueves	01/01/2009	4.1	4.1	2.0	...	2009-01-01 00:13:08	

2 rows × 27 columns

In [4]:

```
alerts_allyears.tail(2)
```

Out[4]:

	AÑO	SUCESO	INFORME	PACIENTE	MES	DIA_SEMANA	FECHA_FIN	CODIGO_INICIAL	CODIGO_FINAL	BASE	...	CLAVE_3
1507322	2019	107885	123535	1	10	Jueves	01/11/2019	3.12	3.12	19.0	...	2019-11-01 00:18:07
1507323	2019	107886	123536	0	10	Jueves	01/11/2019	5.40	0.00	10.0	...	2019-11-01 00:04:13

2 rows × 27 columns

Número de avisos

Evolución de avisos por año y mes

In [5]:

```
alerts_per_month_2009 = alerts_2009[["SUCESO","MES"]].groupby(['MES']).count()
alerts_per_month_2010 = alerts_2010[["SUCESO","MES"]].groupby(['MES']).count()
alerts_per_month_2011 = alerts_2011[["SUCESO","MES"]].groupby(['MES']).count()
alerts_per_month_2012 = alerts_2012[["SUCESO","MES"]].groupby(['MES']).count()
alerts_per_month_2013 = alerts_2013[["SUCESO","MES"]].groupby(['MES']).count()
alerts_per_month_2014 = alerts_2014[["SUCESO","MES"]].groupby(['MES']).count()
alerts_per_month_2015 = alerts_2015[["SUCESO","MES"]].groupby(['MES']).count()
alerts_per_month_2016 = alerts_2016[["SUCESO","MES"]].groupby(['MES']).count()
alerts_per_month_2017 = alerts_2017[["SUCESO","MES"]].groupby(['MES']).count()
alerts_per_month_2018 = alerts_2018[["SUCESO","MES"]].groupby(['MES']).count()
alerts_per_month_2019 = alerts_2019[["SUCESO","MES"]].groupby(['MES']).count()

alerts_per_month=[alerts_per_month_2009, alerts_per_month_2010, alerts_per_month_2011, alerts_per_month_2012,
                  alerts_per_month_2013, alerts_per_month_2014, alerts_per_month_2015, alerts_per_month_2016,
                  alerts_per_month_2017, alerts_per_month_2018, alerts_per_month_2019]
```

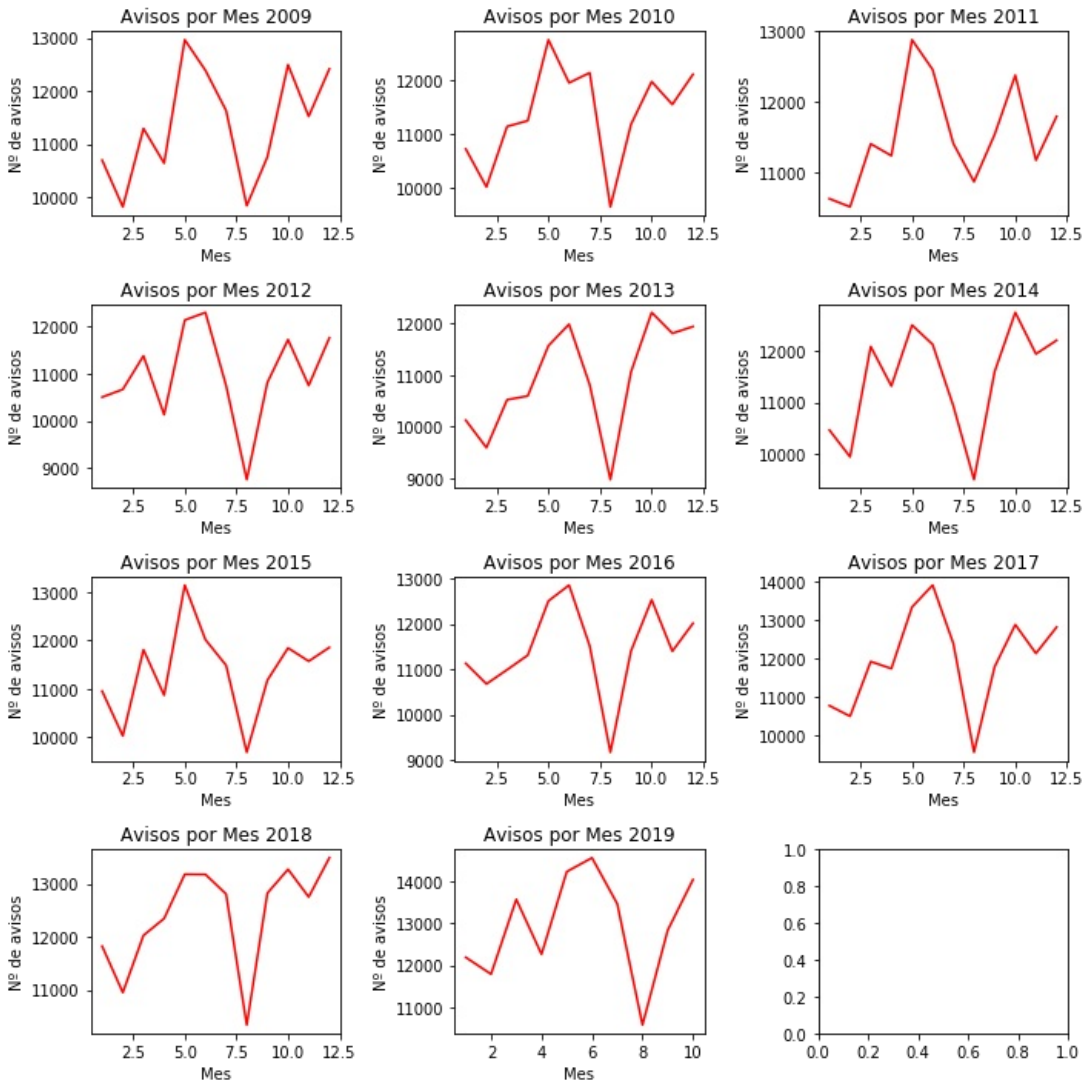
In [6]:

```
def plot_alerts_per_field(alerts_per_field,year,field): #year is the first of the years in alerts_per_field
    index=1
    plt.subplots(4,3,figsize=(10,10))
    for alerts in alerts_per_field:
        plt.subplot(4, 3, index)
        plt.tight_layout()
        plt.plot(alerts, color= "red")
        plt.title("Avisos por {} {}".format(field,year))
        plt.xlabel('{}'.format(field))
        plt.ylabel('Nº de avisos')
        year = year + 1
        index = index + 1
```

In [7]:

```
plot_alerts_per_field(alerts_per_month,2009,"Mes")
```

Out[7]:



Podemos ver en los gráficos de arriba que el mes con menos alertas es agosto. Los meses con mayor número de alertas son mayo entre 2009-2011 y junio entre 2012-2019

También se ve que algunos años en octubre se ha disparado el número de alertas o que el último registro que tenemos de diciembre también fue significativo, por lo que sería interesante en el futuro observar que ha pasado en diciembre de 2019.

Evolución de avisos por mes

In [8]:

```
alerts_allyears[["SUCESO","MES"]].groupby(['MES']).count().sort_values(["SUCESO"],ascending=False).rename(columns
={"SUCESO": "Nº Avisos"}).head(5)
```

Out[8]:

MES	Nº Avisos
5	141173
6	139692
10	138063
7	129312
3	128117

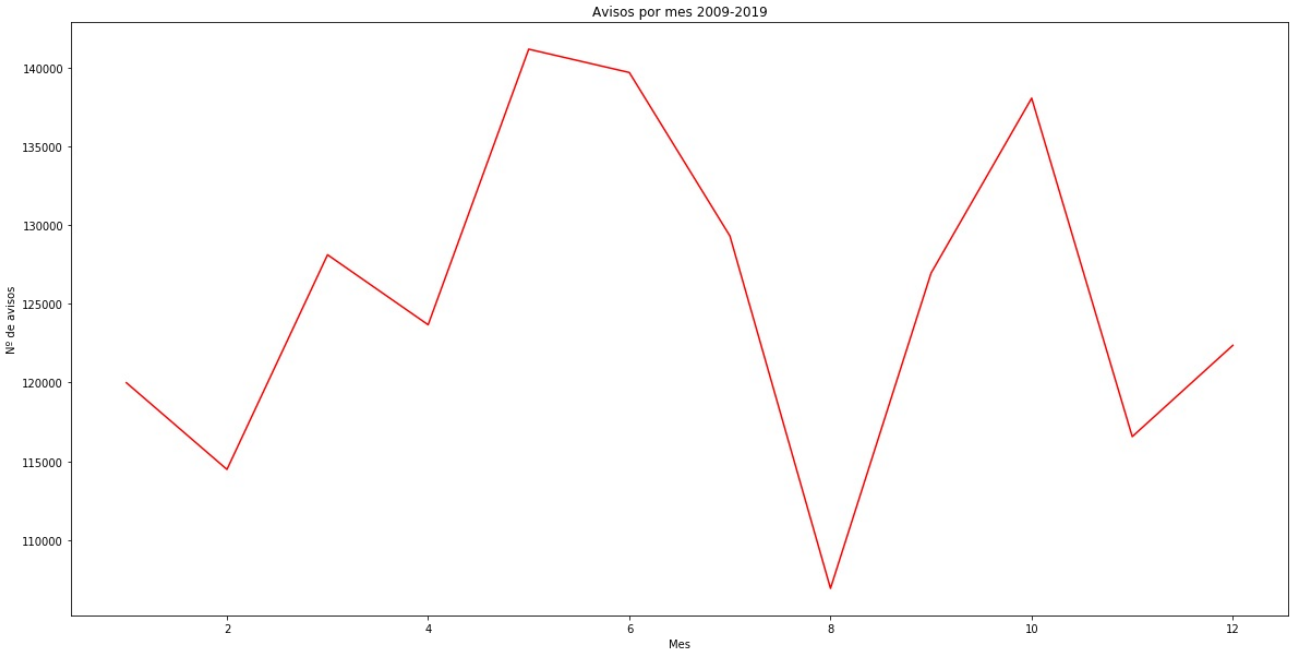
In [9]:

```
plt.figure(figsize=(20,10))
plt.plot(alerts_allyears[["SUCESO","MES"]].groupby(['MES']).count(), color="red")
plt.title("Avisos por mes 2009-2019")
plt.xlabel("Mes")
plt.ylabel('Nº de avisos')
```

Out[9]:

```
Text(0, 0.5, 'Nº de avisos')
```

Out[9]:



Si hacemos un recuento de todos los años juntos podemos ver que el mes con mayor número de alertas son mayo y junio.

Evolución de avisos por distrito y año

In [10]:

```
districts_info = pd.read_excel("data/distritos.xlsx", index_col=False)
```

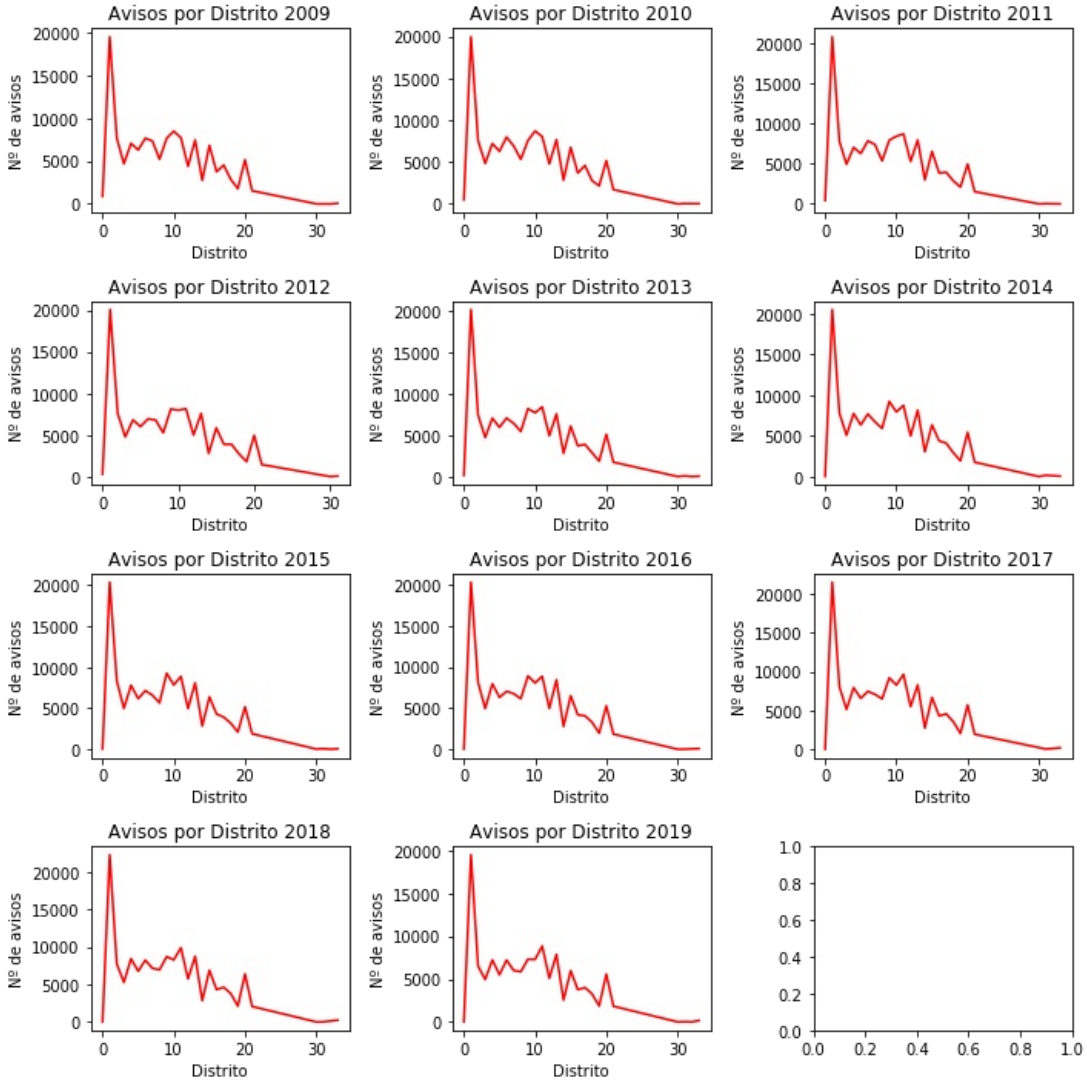
In [11]:

```
alerts_per_district_2009 = alerts_2009[["SUCESO","DISTRITO"]].groupby(['DISTRITO']).count()
alerts_per_district_2010 = alerts_2010[["SUCESO","DISTRITO"]].groupby(['DISTRITO']).count()
alerts_per_district_2011 = alerts_2011[["SUCESO","DISTRITO"]].groupby(['DISTRITO']).count()
alerts_per_district_2012 = alerts_2012[["SUCESO","DISTRITO"]].groupby(['DISTRITO']).count()
alerts_per_district_2013 = alerts_2013[["SUCESO","DISTRITO"]].groupby(['DISTRITO']).count()
alerts_per_district_2014 = alerts_2014[["SUCESO","DISTRITO"]].groupby(['DISTRITO']).count()
alerts_per_district_2015 = alerts_2015[["SUCESO","DISTRITO"]].groupby(['DISTRITO']).count()
alerts_per_district_2016 = alerts_2016[["SUCESO","DISTRITO"]].groupby(['DISTRITO']).count()
alerts_per_district_2017 = alerts_2017[["SUCESO","DISTRITO"]].groupby(['DISTRITO']).count()
alerts_per_district_2018 = alerts_2018[["SUCESO","DISTRITO"]].groupby(['DISTRITO']).count()
alerts_per_district_2019 = alerts_2019[["SUCESO","DISTRITO"]].groupby(['DISTRITO']).count()
alerts_per_district=[alerts_per_district_2009, alerts_per_district_2010, alerts_per_district_2011, alerts_per_dis
trict_2012,
                     alerts_per_district_2013, alerts_per_district_2014, alerts_per_district_2015, alerts_per_distric
t_2016,
                     alerts_per_district_2017, alerts_per_district_2018, alerts_per_district_2019]
```

In [12]:

```
plot_alerts_per_field(alerts_per_district,2009,"Distrito")
```

Out[12]:



Podemos ver en los gráficos de arriba que el distrito con el mayor número de alertas a lo largo de los años es el distrito número 1 que se corresponde con el ***CENTRO***

Avisos por distrito

In [13]:

```
alerts_per_district_allyears = pd.merge(left=districts_info,right=alerts_allyears[["SUCESO","DISTRITO"]].groupby(
['DISTRITO']).count().rename(columns = {"SUCESO" : "Nº Avisos"}), left_on='Distrito', right_on='DISTRITO').drop(c
olumns=['Distrito'])
alerts_per_district_allyears
```

Out[13]:

	Nombre	Nº Avisos
0	Centro	225093
1	Arganzuela	84619
2	Retiro	54337
3	Salamanca	82273
4	Chamartín	68452
5	Tetuán	82312
6	Chamberí	75145
7	Fuencarral-El Pardo	63619
8	Moncloa-Aravaca	92103
9	Latina	89002
10	Carabanchel	95890
11	Usera	55623
12	Puente de Vallecas	87849
13	Moratalaz	30940
14	Ciudad Lineal	70834
15	Hortaleza	44078
16	Villaverde	46120
17	Villa de Vallecas	34082
18	Vicálvaro	21672
19	San Blas- Canillejas	58869
20	Barajas	19232

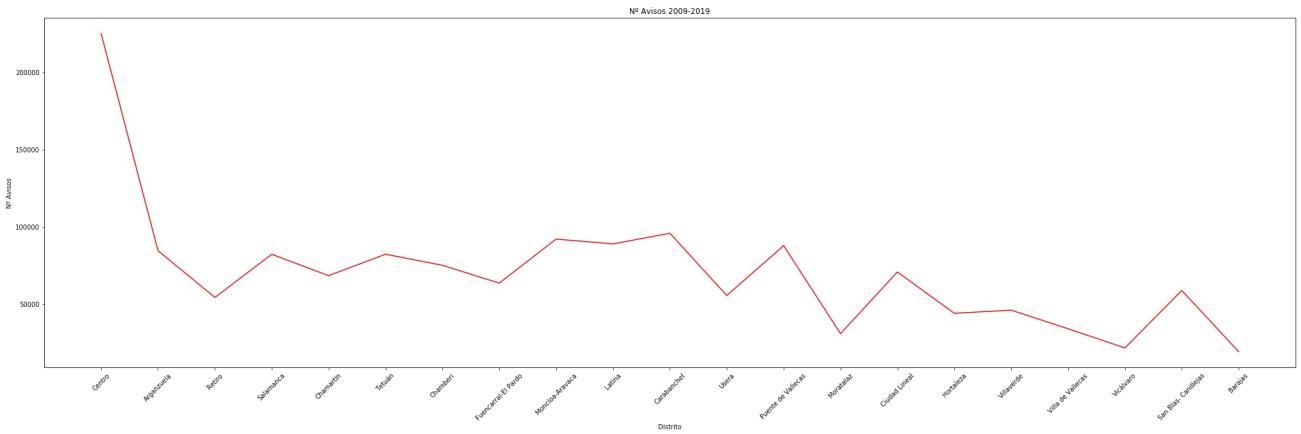
In [14]:

```
plt.figure(figsize=(35,10))
plt.plot(alerts_per_district_allyears.Nombre,alerts_per_district_allyears["Nº Avisos"], color="red")
plt.title("Nº Avisos 2009-2019")
plt.xlabel("Distrito")
plt.ylabel('Nº Avisos')
plt.xticks(rotation=45)
```

Out[14]:

```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20],
 <a list of 21 Text xticklabel objects>)
```

Out[14]:



Podemos ver en los gráficos de arriba que el distrito con el mayor número de alertas a lo largo de los años es el distrito número 1 que se corresponde con el ***CENTRO***

Evolución de avisos por dispositivo y año

In [15]:

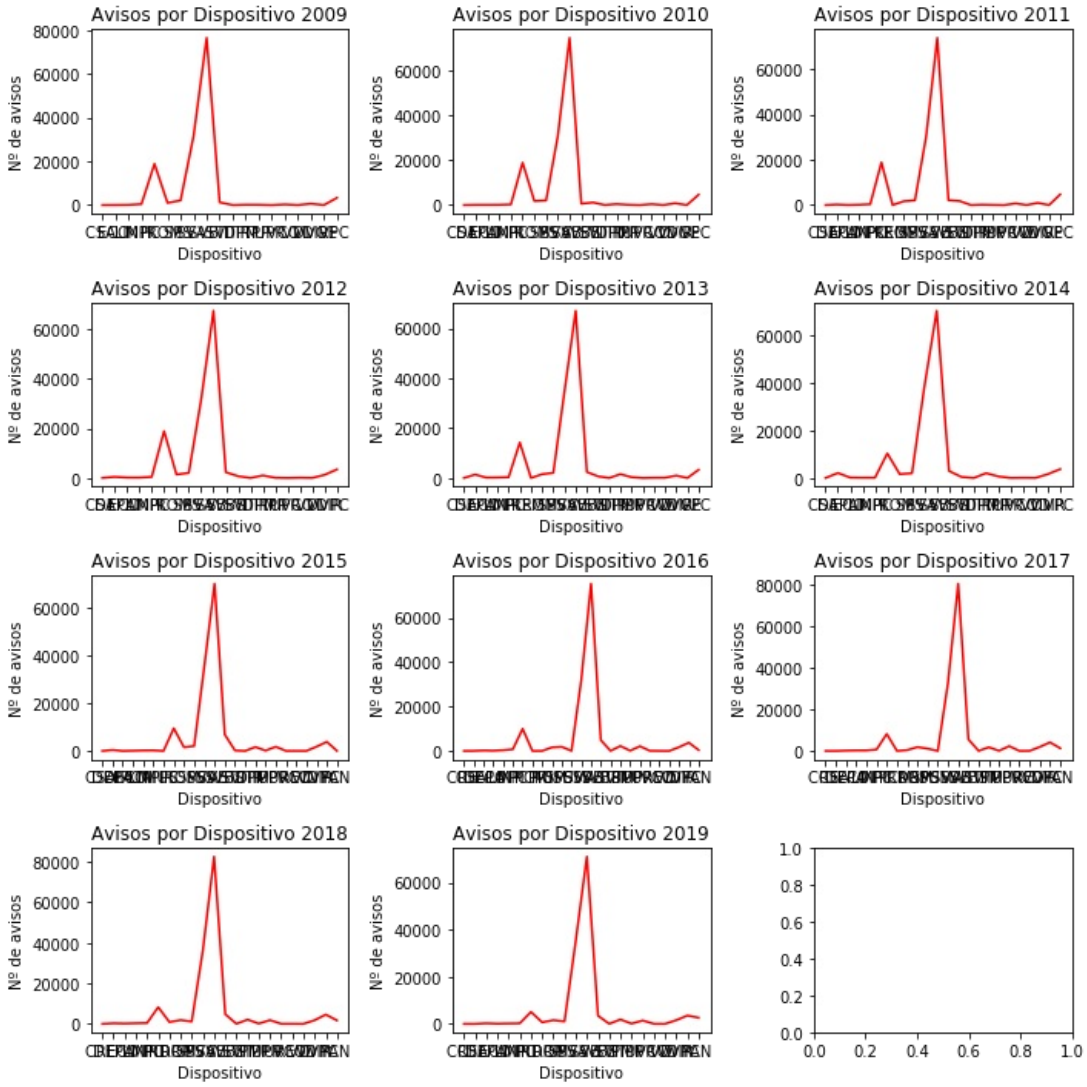
```
alerts_per_device_2009 = alerts_2009[["SUCESO","ABREV"]].groupby(['ABREV']).count()
alerts_per_device_2010 = alerts_2010[["SUCESO","ABREV"]].groupby(['ABREV']).count()
alerts_per_device_2011 = alerts_2011[["SUCESO","ABREV"]].groupby(['ABREV']).count()
alerts_per_device_2012 = alerts_2012[["SUCESO","ABREV"]].groupby(['ABREV']).count()
alerts_per_device_2013 = alerts_2013[["SUCESO","ABREV"]].groupby(['ABREV']).count()
alerts_per_device_2014 = alerts_2014[["SUCESO","ABREV"]].groupby(['ABREV']).count()
alerts_per_device_2015 = alerts_2015[["SUCESO","ABREV"]].groupby(['ABREV']).count()
alerts_per_device_2016 = alerts_2016[["SUCESO","ABREV"]].groupby(['ABREV']).count()
alerts_per_device_2017 = alerts_2017[["SUCESO","ABREV"]].groupby(['ABREV']).count()
alerts_per_device_2018 = alerts_2018[["SUCESO","ABREV"]].groupby(['ABREV']).count()
alerts_per_device_2019 = alerts_2019[["SUCESO","ABREV"]].groupby(['ABREV']).count()

alters_per_device=[alerts_per_device_2009, alerts_per_device_2010, alerts_per_device_2011, alerts_per_device_2012
,
                   alerts_per_device_2013, alerts_per_device_2014, alerts_per_device_2015, alerts_per_device_2016,
                   alerts_per_device_2017, alerts_per_device_2018, alerts_per_device_2019]
```

In [16]:

```
plot_alerts_per_field(alters_per_device,2009,"Dispositivo")
```

Out[16]:



Aunque no se ve muy bien el eje X de las gráficas, los dispositivos de los que hay más alertas corresponden con SVB (Soporte Vital Básico) y SVA (Soporte Vital Avanzado)

Avisos por dispositivo

In [17]:

```
alerts_allyears[["SUCESO","ABREV"]].groupby(['ABREV']).count().sort_values(["SUCESO"],ascending=False).rename(columns={"SUCESO": "Nº Avisos"}).head(5)
```

Out[17]:

ABREV	Nº Avisos
SVB	809889
SVA	369264
PC	141021
VPC	42939
SVBS	35950

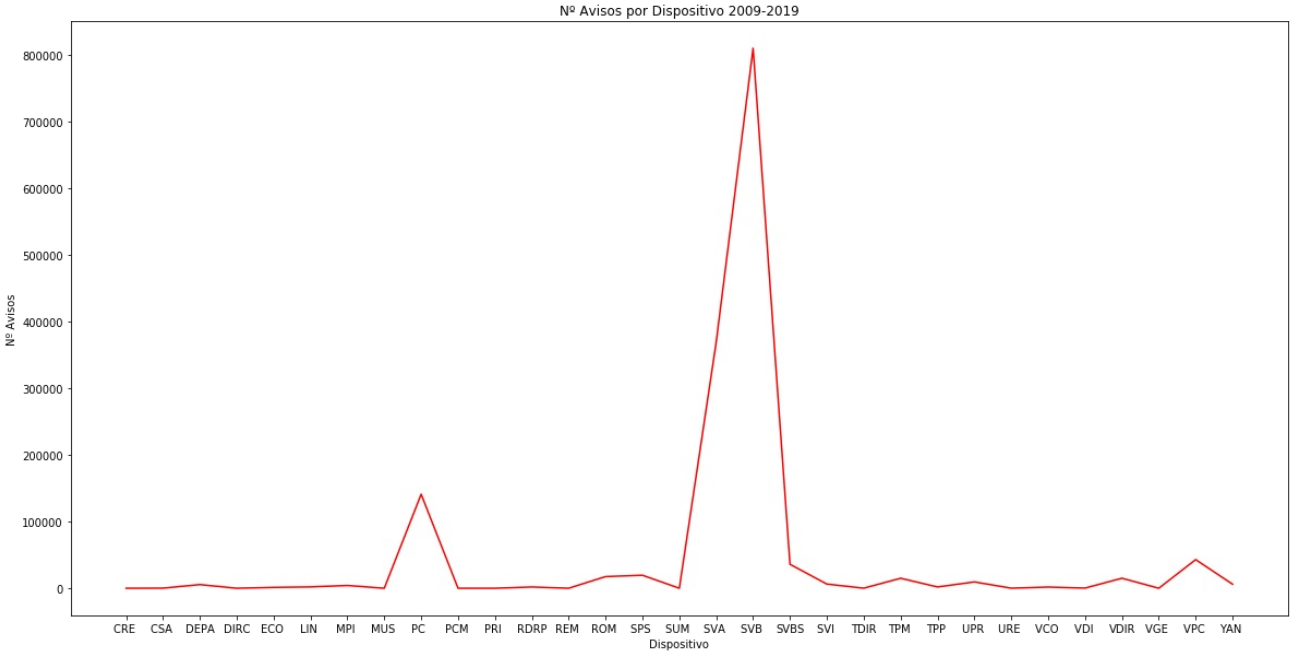
In [18]:

```
plt.figure(figsize=(20,10))
plt.plot(alerts_allyears[["SUCESO","ABREV"]].groupby(['ABREV']).count(), color="red")
plt.title("Nº Avisos por Dispositivo 2009-2019")
plt.xlabel("Dispositivo")
plt.ylabel('Nº Avisos')
```

Out[18]:

```
Text(0, 0.5, 'Nº Avisos')
```

Out[18]:



Los dispositivos de los que hay más alertas corresponden con SVB (Soporte Vital Básico) y SVA (Soporte Vital Avanzado)

Evolución de avisos por día de la semana

In [19]:

```
daily_alerts_count = pd.DataFrame(alerts_allyears[["SUCESO","DIA_SEMANA"]].groupby(['DIA_SEMANA']).count()).reind
ex(["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"]).rename(columns={"SUCESO": "Nº Avis
os"})
daily_alerts_count
```

Out[19]:

DIA_SEMANA	Nº Avisos
Lunes	203059
Martes	196504
Miércoles	201200
Jueves	209008
Viernes	228251
Sábado	239091
Domingo	230211

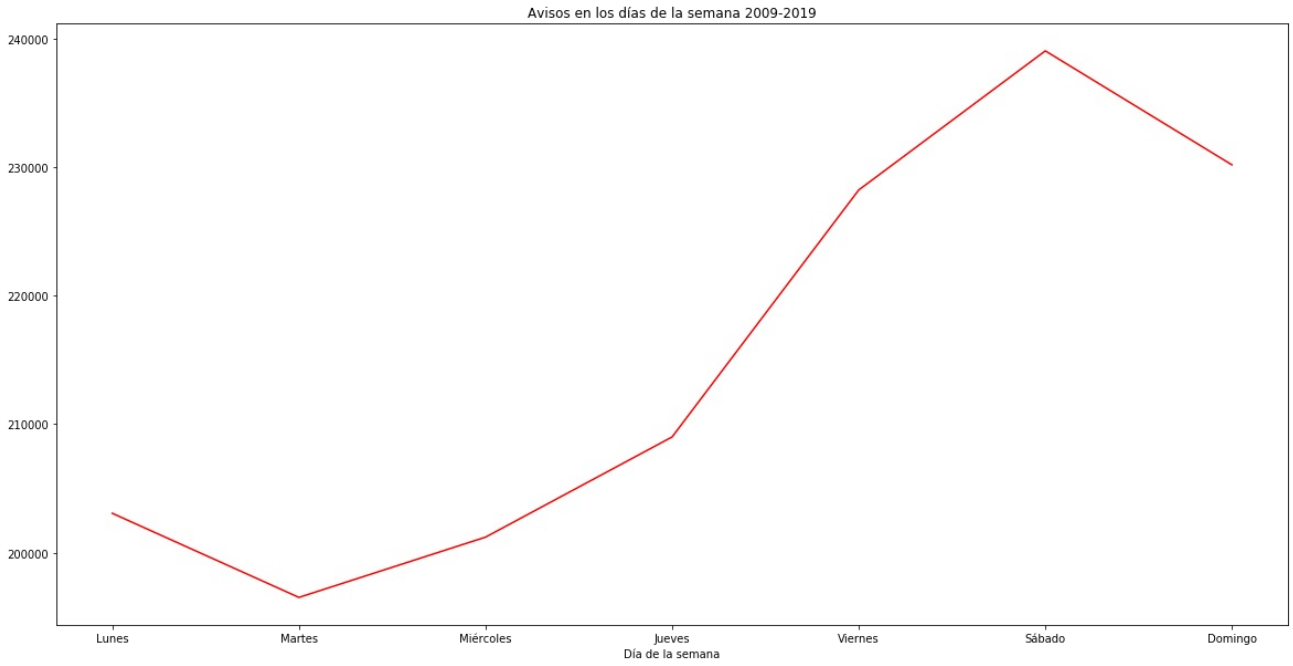
In [20]:

```
plt.figure(figsize=(20,10))
plt.plot(daily_alerts_count.reindex(["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"]),
color = "red")
plt.title("Avisos en los días de la semana 2009-2019")
plt.xlabel("Día de la semana")
```

Out[20]:

```
Text(0.5, 0, 'Día de la semana')
```

Out[20]:



In [21]:

```
daily_alerts_count["% Avisos"] = (daily_alerts_count["Nº Avisos"]/ daily_alerts_count["Nº Avisos"].sum()) * 100
daily_alerts_count
```

Out[21]:

DIA_SEMANA	Nº Avisos	% Avisos
Lunes	203059	13.471490
Martes	196504	13.036613
Miércoles	201200	13.348159
Jueves	209008	13.866163
Viernes	228251	15.142796
Sábado	239091	15.861951
Domingo	230211	15.272828

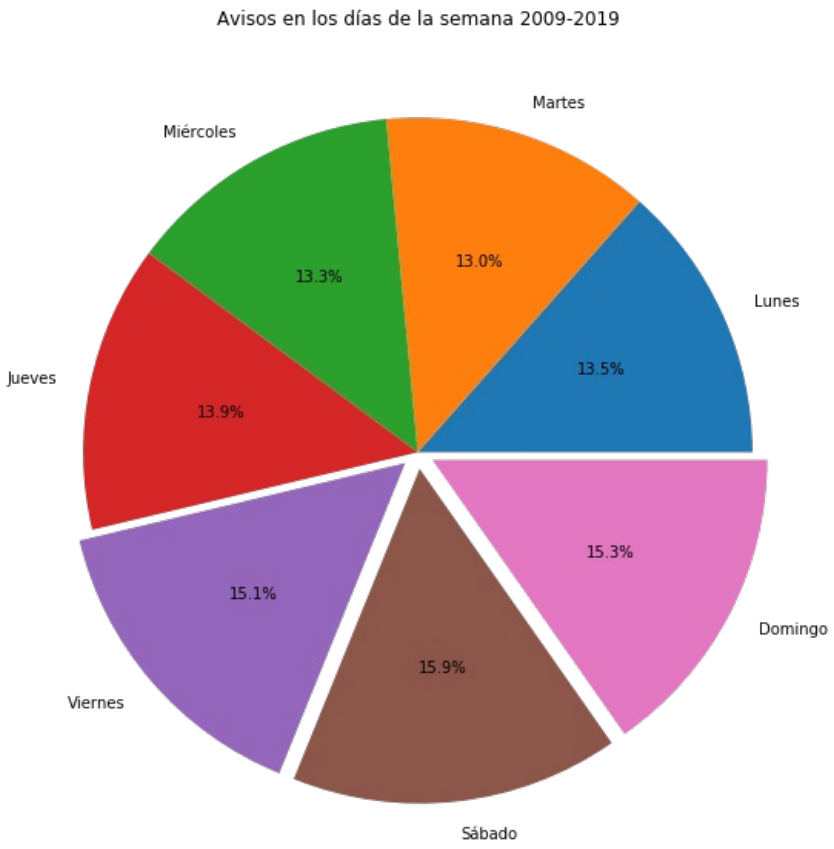
In [22]:

```
plt.figure(figsize=(20,10))
plt.pie(daily_alerts_count["Nº Avisos"], labels = daily_alerts_count.index, explode = (0, 0, 0, 0, 0.05 , 0.05, 0.05), autopct='%1.1f%%', radius = 1)
plt.title("Avisos en los días de la semana 2009-2019")
```

Out[22]:

```
Text(0.5, 1.0, 'Avisos en los días de la semana 2009-2019')
```

Out[22]:



En los fines de semana (viernes, sábado y domingo) es el momento en que se producen la mayoría de las alertas (en torno al 47% de las alertas totales pertenecen al fin de semana)

DIAS ENTRE SEMANA VS DIAS FIN DE SEMANA

In [23]:

```
daily_alerts_count["Tipo de día"] = ["Entre semana", "Entre semana", "Entre semana", "Entre semana", "Fin de Sema
na",
                                     "Fin de Semana", "Fin de Semana"]
daily_alerts_count
```

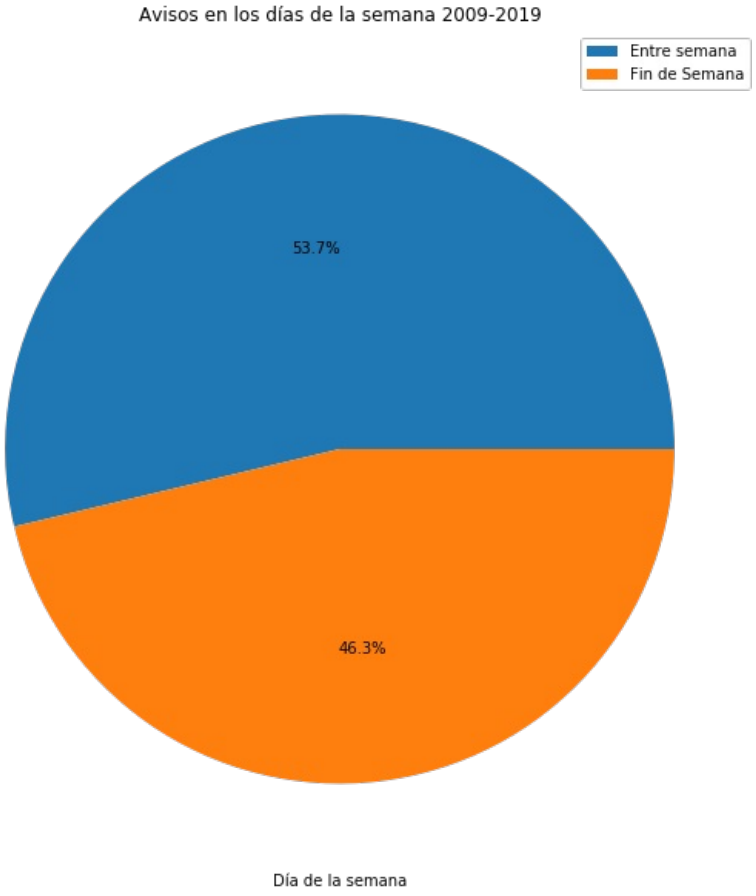
Out[23]:

	Nº Avisos	% Avisos	Tipo de día
DIA_SEMANA			
Lunes	203059	13.471490	Entre semana
Martes	196504	13.036613	Entre semana
Miércoles	201200	13.348159	Entre semana
Jueves	209008	13.866163	Entre semana
Viernes	228251	15.142796	Fin de Semana
Sábado	239091	15.861951	Fin de Semana
Domingo	230211	15.272828	Fin de Semana

In [24]:

```
plt.figure(figsize=(20,10))
plt.pie(daily_alerts_count[["Nº Avisos","Tipo de día"]].groupby(['Tipo de día']).sum()["Nº Avisos"],autopct='%1.1
f%%',radius = 1)
plt.legend(labels=daily_alerts_count["Tipo de día"].unique())
plt.title("Avisos en los días de la semana 2009-2019")
plt.xlabel("Día de la semana")
plt.margins(x=0,y=0)
```

Out[24]:



Número de avisos por turno

Horarios:

7:00 - 15:00 Mañana (M)

15:00 - 23:00 Tarde (T)

23:00 - 7:00 Noche (N)

In [28]:

```
alerts_per_turn = pd.DataFrame(alerts_allyears[["SUCESO", "TURNO"]].groupby(["TURNO"]).count().rename(columns={"SUCESO" : "Avisos"}))
alerts_per_turn
```

Out[28]:

TURNO	Avisos
M	587172
N	289979
T	630173

Observamos que cuando hay menos avisos es durante el turno de noche.

Número de avisos por turno y por día

Horarios:

7:00 - 15:00 Mañana (M)

15:00 - 23:00 Tarde (T)

23:00 - 7:00 Noche (N)

In [29]:

```
alerts_per_turn_and_day = pd.DataFrame(alerts_allyears[["SUCESO", "DIA_SEMANA", "TURNO"]].groupby(["DIA_SEMANA", "TURNO"]).count().rename(columns={"SUCESO" : "Nº Avisos"}))
alerts_per_turn_and_day
```

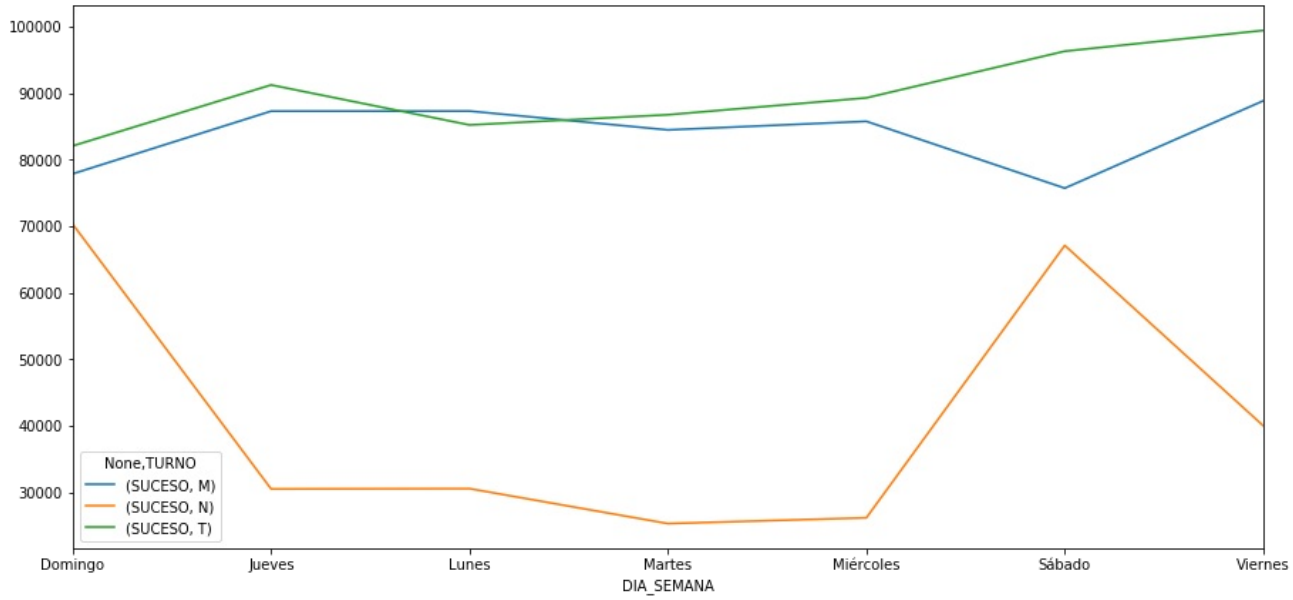
Out[29]:

DIA_SEMANA	TURNO	Nº Avisos
Domingo	M	77863
	N	70291
	T	82057
Jueves	M	87279
	N	30514
	T	91215
Lunes	M	87291
	N	30563
	T	85205
Martes	M	84458
	N	25310
	T	86736
Miércoles	M	85749
	N	26178
	T	89273
Sábado	M	75706
	N	67102
	T	96283
Viernes	M	88826
	N	40021
	T	99404

In [30]:

```
fig, ax = plt.subplots(figsize=(15,7))
pd.DataFrame(alerts_allyears[["SUCESO", "DIA_SEMANA", "TURNO"]].groupby(["DIA_SEMANA", "TURNO"]).count()).unstack
().plot(ax=ax)
plt.savefig("avisos_turnos_dia.png")
```

Out[30]:



Número de avisos por código

In [31]:

```
codes_info = pd.read_excel("data/codigos_SAMUR-PC.xlsx", index_col=False)
codes_info["Codigo"]=codes_info["Codigo"].astype('str')
```

In [32]:

```
alerts_per_code = alerts_allyears[["SUCESO", "CODIGO_FINAL"]].groupby(["CODIGO_FINAL"],as_index=False).count().re
name(columns={"SUCESO" : "Nº Avisos"})
alerts_per_code["CODIGO_FINAL"]=alerts_per_code["CODIGO_FINAL"].astype('str')

alerts_per_code = pd.merge(left=codes_info, right=alerts_per_code, left_on='Codigo', right_on='CODIGO_FINAL').sor
t_values(["Nº Avisos"],ascending=True)
alerts_per_code.sort_values(["Nº Avisos"],ascending=False).head(15)
```

Out[32]:

	Codigo	Descripcion	CODIGO_FINAL	Nº Avisos
11	2.3	Casual: caída, etc.	2.3	214346
31	3.12	Patología cardiovascular	3.12	167901
12	2.4	Agresión sin especificar	2.4	85463
0	0.1	No hay suceso	0.1	77124
2	1.2	Accidente con menos de 3 víctimas	1.2	70763
29	3.9	Intoxicación etílica	3.9	68058
9	2.1	Heridas	2.1	52436
5	1.5	Accidente de motocicleta	1.5	48370
21	3.1	Parada cardiorrespiratoria	3.1	43802
33	3.14	Patología digestiva	3.14	28102
47	5.4	Certificación psiquiátrica	5.4	25962
4	1.4	Atropello	1.4	21351
24	3.4	Convulsión	3.4	20355
48	5.5	Problema social	5.5	18520
32	3.13	Patología neurológica	3.13	18313

Se puede ver que los códigos con mayor número de avisos son 2.30 (poner lo que es), 0 (que es) y 3.12 (que es)

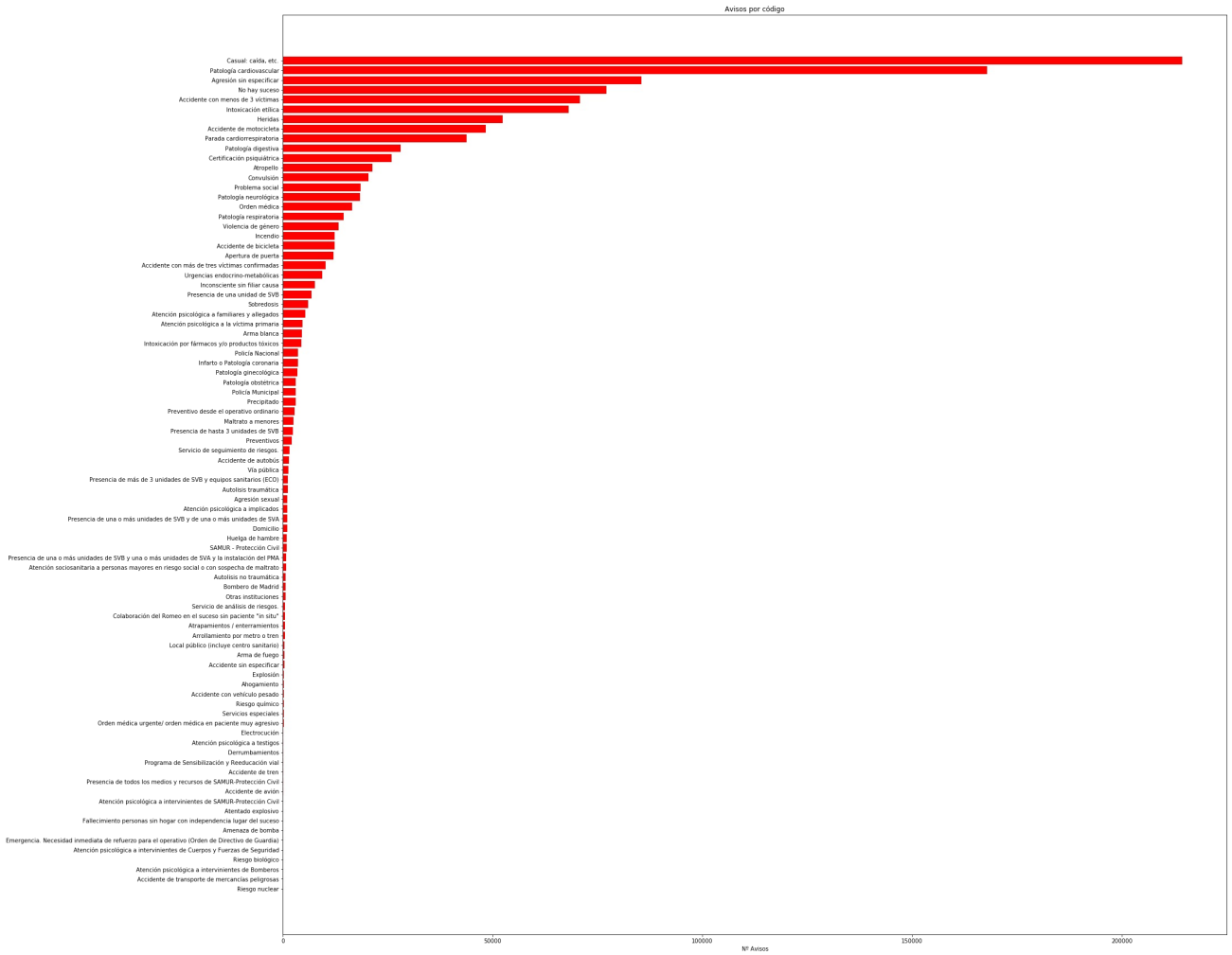
In [33]:

```
plt.figure(figsize=(30,30))
plt.barh(alerts_per_code["Descripcion"], alerts_per_code["Nº Avisos"], color="red",)
plt.title("Avisos por código")
plt.xlabel("Nº Avisos")
```

Out[33]:

```
Text(0.5, 0, 'Nº Avisos')
```

Out[33]:



Número de avisos por bases

Número de avisos por bases en os días de la semana

In [34]:

```
alerts_per_base_day = alerts_allyears[["SUCESO","DIA_SEMANA","BASE_CORREGIDA"]].groupby(["BASE_CORREGIDA",'DIA_SEMANA']).count().rename(columns={"SUCESO": "Nº Avisos"})
alerts_per_base_day.head(10)
```

Out[34]:

		Nº Avisos
BASE_CORREGIDA	**DIA_SEMANA**	
-1	**Domingo**	29658
	Jueves	19870
	Lunes	17031
	Martes	17581
	Miércoles	18732
	Sábado	29316
	Viernes	23467
0	**Domingo**	11
	Jueves	28
	Lunes	9

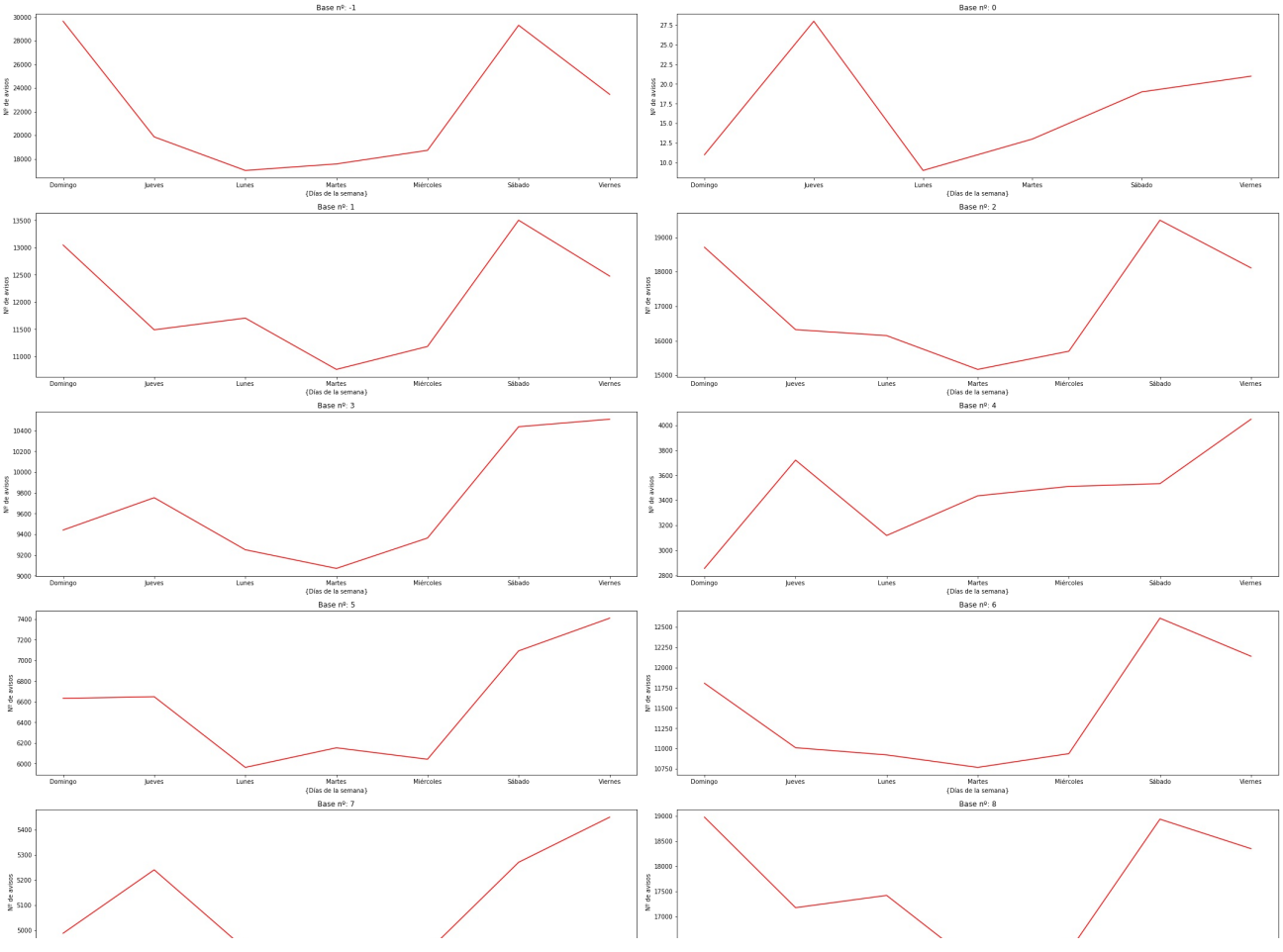
In [35]:

```
alerts_per_base_day.reset_index(level=1, inplace=True)
alerts_per_base_day = alerts_per_base_day.groupby("BASE_CORREGIDA")
```

In [36]:

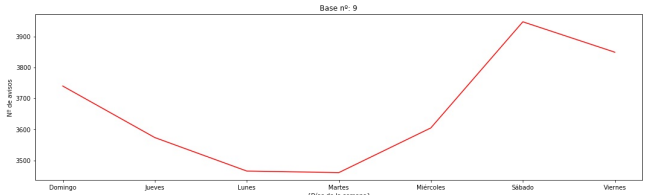
```
index=1
plt.subplots(13,2,figsize=(30,60))
for name, group in alerts_per_base_day:
    plt.subplot(13, 2, index)
    plt.tight_layout()
    plt.plot(group["DIA_SEMANA"], group["Nº Avisos"], color="red")
    plt.title("Base nº: {}".format(name))
    plt.xlabel('{Días de la semana}')
    plt.ylabel('Nº de avisos')
    index = index + 1
```

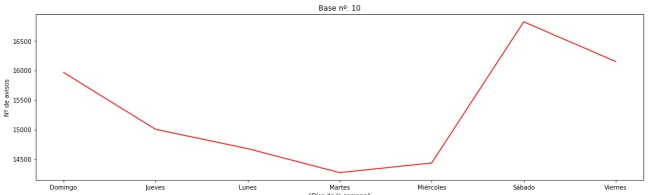
Out[36]:

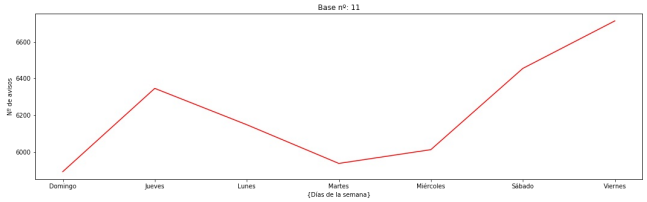


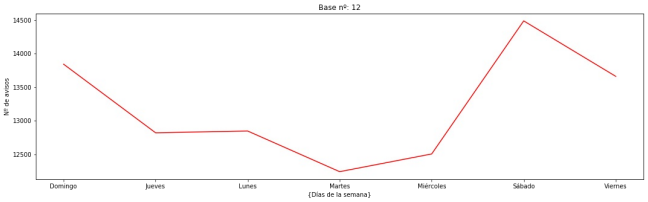

4900
Domingo
Jueves
Lunes
Martes
Miércoles
Sábado
Viernes
{Días de la semana}

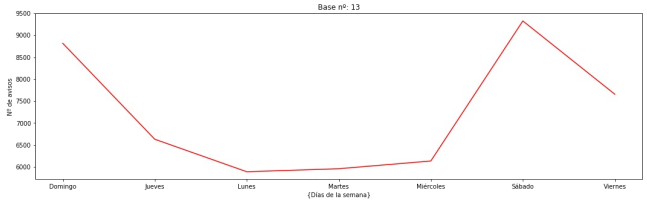

16500
16000
Domingo
Jueves
Lunes
Martes
Miércoles
Sábado
Viernes
{Días de la semana}

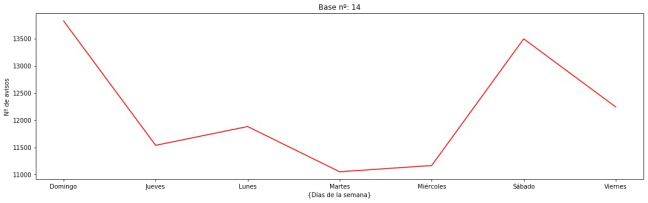

Base nº: 9
Nº de avisos
3900
3800
3700
3600
3500
Domingo
Jueves
Lunes
Martes
Miércoles
Sábado
Viernes
{Días de la semana}

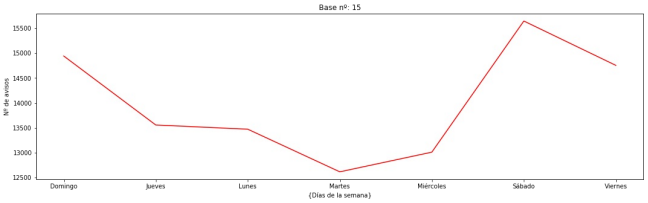

Base nº: 10
Nº de avisos
16500
16000
15500
15000
14500
Domingo
Jueves
Lunes
Martes
Miércoles
Sábado
Viernes
{Días de la semana}

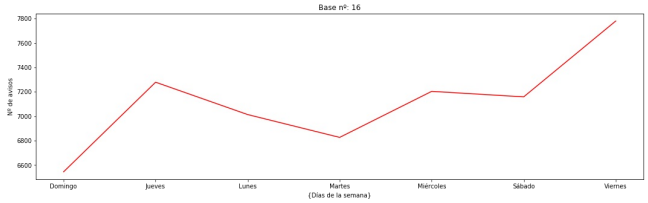

Base nº: 11
Nº de avisos
6600
6400
6200
6000
Domingo
Jueves
Lunes
Martes
Miércoles
Sábado
Viernes
{Días de la semana}

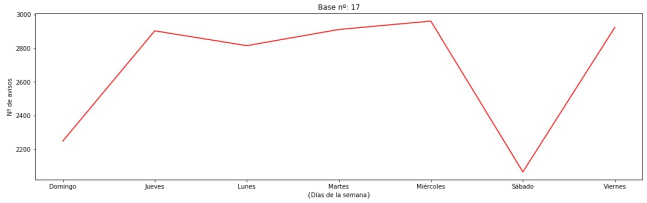

Base nº: 12
Nº de avisos
14500
14000
13500
13000
12500
Domingo
Jueves
Lunes
Martes
Miércoles
Sábado
Viernes
{Días de la semana}

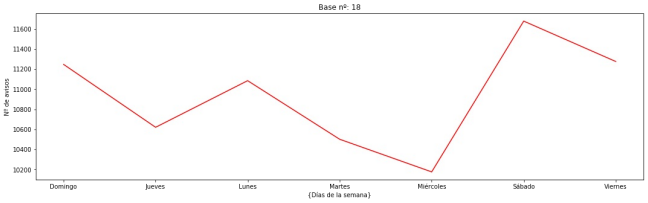

Base nº: 13
Nº de avisos
9500
9000
8500
8000
7500
7000
6500
6000
Domingo
Jueves
Lunes
Martes
Miércoles
Sábado
Viernes
{Días de la semana}

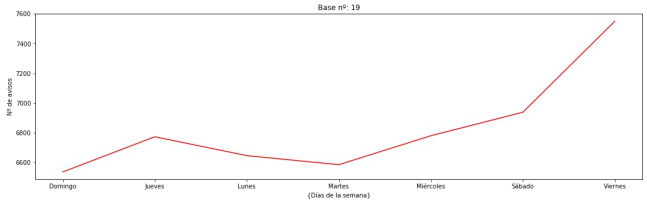

Base nº: 14
Nº de avisos
13500
13000
12500
12000
11500
11000
Domingo
Jueves
Lunes
Martes
Miércoles
Sábado
Viernes
{Días de la semana}

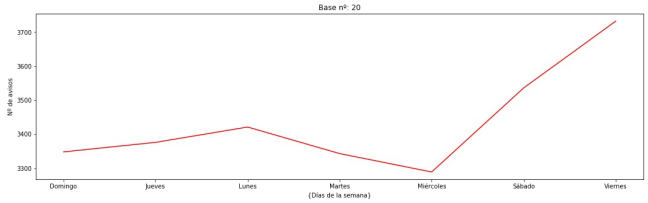

Base nº: 15
Nº de avisos
15500
15000
14500
14000
13500
13000
12500
Domingo
Jueves
Lunes
Martes
Miércoles
Sábado
Viernes
{Días de la semana}

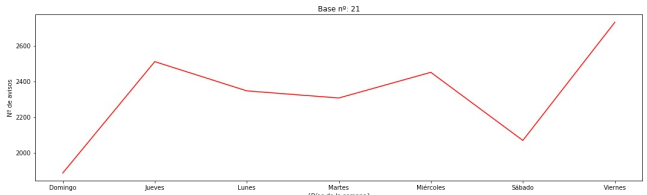

Base nº: 16
Nº de avisos
7800
7600
7400
7200
7000
6800
6600
Domingo
Jueves
Lunes
Martes
Miércoles
Sábado
Viernes
{Días de la semana}

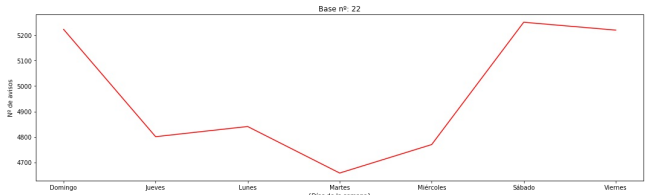

Base nº: 17
Nº de avisos
3000
2800
2600
2400
2200
Domingo
Jueves
Lunes
Martes
Miércoles
Sábado
Viernes
{Días de la semana}

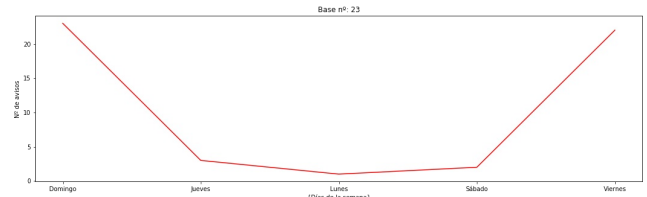

Base nº: 18
Nº de avisos
11600
11400
11200
11000
10800
10600
10400
10200
Domingo
Jueves
Lunes
Martes
Miércoles
Sábado
Viernes
{Días de la semana}

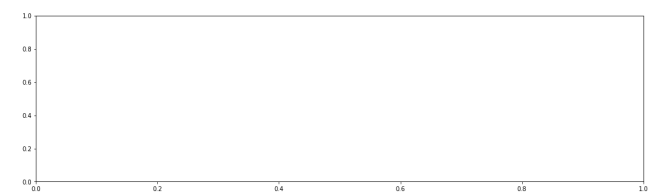

Base nº: 19
Nº de avisos
7600
7400
7200
7000
6800
6600
Domingo
Jueves
Lunes
Martes
Miércoles
Sábado
Viernes
{Días de la semana}


Base nº: 20
Nº de avisos
3700
3600
3500
3400
3300
Domingo
Jueves
Lunes
Martes
Miércoles
Sábado
Viernes
{Días de la semana}


Base nº: 21
Nº de avisos
2600
2400
2200
2000
Domingo
Jueves
Lunes
Martes
Miércoles
Sábado
Viernes
{Días de la semana}


Base nº: 22
Nº de avisos
5200
5100
5000
4900
4800
4700
Domingo
Jueves
Lunes
Martes
Miércoles
Sábado
Viernes
{Días de la semana}


Base nº: 23
Nº de avisos
20
15
10
5
0
Domingo
Jueves
Lunes
Sábado
Viernes
{Días de la semana}


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

Regresión Lineal SAMUR- Protección Civil

Propósito del documento

El objetivo de este documento es la creación de modelos utilizando como técnica de aprendizaje Regresión lineal, para predecir el número de avisos o activaciones que recebirá SAMUR-PC para un determinado día y unas determinadas cirucnstancias en la ciudad de Madrid.

Regresión Lineal: Consiste en generar un recta de la forma hΘ(x) =Θ0+ Θ1x1+...+ Θnxn para predecir el valor de la variable objetivo a través de la combinación lineal de las varaibles independientes. Se utilizarán tanto el método de la ecuación normal estudiado en la asignatura Aprendizaje Automático & Big Data como la clase LinearRegression de la libreria de sklearn para construir el modelo.

Para comprobar la validez del modelo se utilizarán diferentes métricas.

- RMSE: Raíz del Error cuadrático medio
- MAE: Media del error absoluto de las predcciones.

Los datos de los años 2009-2019 se utilizarán para entrenar el modelo, excepto 2018 que se utilizará para como datos de test, para predecir el valor de la variable objetivo. A estos datos se le eliminaran aquellos valores atípicos.l

Por si acaso 2018 no fuera una muestra representativa de los datos a nivel general, se utilizarán también validación cruzada, es decir, se escogerán porciones de los datos para entrenamiento y test calculando las métricas obtenidas para estos datos, despues de ello la media de las métricas obtenidas de las particiones será la métrica final.

Librerías

In [1]:

```
import pandas as pd
from matplotlib import pyplot as plt
import numpy as np
from matplotlib import rc
from mpl_toolkits.mplot3d import Axes3D
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from math import sqrt
from scipy import stats
from sklearn.model_selection import cross_validate
import pickle
```

Obtenemos los datos

In [2]:

```
alerts_2009 = pd.read_excel("data/samur2009.xlsx")
alerts_2010 = pd.read_excel("data/samur2010.xlsx")
alerts_2011 = pd.read_excel("data/samur2011.xlsx")
alerts_2012 = pd.read_excel("data/samur2012.xlsx")
alerts_2013 = pd.read_excel("data/samur2013.xlsx")
alerts_2014 = pd.read_excel("data/samur2014.xlsx")
alerts_2015 = pd.read_excel("data/samur2015.xlsx")
alerts_2016 = pd.read_excel("data/samur2016.xlsx")
alerts_2017 = pd.read_excel("data/samur2017.xlsx")
alerts_2018 = pd.read_excel("data/samur2018.xlsx")
alerts_2019 = pd.read_excel("data/samur2019.xlsx")
alerts_allyears = pd.concat([alerts_2009,alerts_2010, alerts_2011, alerts_2012, alerts_2013, alerts_2014,
                             alerts_2015, alerts_2016, alerts_2017, alerts_2018, alerts_2019], ignore_index=True)
```

In [3]:

```
train_dataframe = pd.concat([alerts_2009,alerts_2010, alerts_2011, alerts_2012, alerts_2013, alerts_2014,
                             alerts_2015, alerts_2016, alerts_2017, alerts_2019], ignore_index=True)
test_dataframe = alerts_2018
```

Las funciones utilizadas para facilitar la regresión lineal serán:

normal_ecuation: nos devuelve el valor de las thetas que minimizan la distancia de los puntos a la linea de la regresión lineal.

normal_ecuation_predict: predict devulve el valor de la variable objetivo obtenida a partir de : hΘ(x) = Θ0+ Θ1x1+...+ Θnxn

rmse_cross_val: devuelve el rmse para unos datos y el modelo dado por parámetro

Por tanto para cada modelo a realizar el flujo será el siguiente:

1. Obtenemos variables independientes (X), variable objetivo(Y), variables dependientes de entrenamiento y test(X_train, X_test) y variable independiente objetivo de entrenamiento y test (Y_train, Y_test). Además se eliminaran los outliers.
2. Calculamos la ecuación normal para construir el modelo para obtener el vetor de thetas que minimizan el coste y predecimos el varor de una muestra
3. Aplicamos la clase LinearRegression de sklearn para construir el modelo, lo entrenamos.
4. Calculamos RMSE y MAE del modelo para esos datos de entrenamiento (2009-2017 \ 2019) y test(2018)
5. Calculamos RMSE y MAE mediante validación cruzada
6. Exportamos el modelo creado para poder utilizarlo

In [4]:

```
def normal_ecuation(X,Y):
    m = np.shape(X)[0]
    X = np.hstack([np.ones([m,1]),X])
    x_transpose = np.transpose(X)
    x_transpose_dot_x = x_transpose.dot(X)
    term_1 = np.linalg.pinv(x_transpose_dot_x)
    term_2 = x_transpose.dot(Y)
    return term_1.dot(term_2)
```

In [5]:

```
def normal_ecuation_predict(thetas, values):
    prediction = thetas[0]
    for i in range(len(values)):
        prediction += thetas[i+1]* values[i]
    return prediction
```

In [6]:

```
def rmse_cross_val(estimator, X, y):
    y_pred = estimator.predict(X)
    return np.sqrt(mean_squared_error(y, y_pred))
```

In [7]:

```
def print_prediction(real, pred, dias):
    """ Prints two plots: 1:a bar for real data and a bar for predicted data 2:the difference between real and pr
edicted data
    """
    #Print real and predicted data
    y = np.zeros((real.shape[0],3))
    y[:,0]= np.arange(real.shape[0])
    y[:,1] = real
    y[:,2] = pred
    df = pd.DataFrame(y, columns=["Días", "Real", "Predicción"]).head(dias)
    ax = df.plot(x="Días", y=["Real", "Predicción"], kind="bar")
    plt.show()


    #Print the difference
    y = np.zeros((real.shape[0],2))
    y[:,0] = np.arange(real.shape[0])
    y[:,1] = real - pred
    df = pd.DataFrame(y, columns=["Días", "Diferencia"]).head(dias)
    ax = df.plot(x="Días", y="Diferencia", kind="bar")
    plt.show()
```

In [8]:

```
def info_model(Y_test, Y_pred):
    # The coefficients
    print('Coefficients: ', model.coef_)
    # The mean squared error
    print('RMSE: %.2f'
      % sqrt(mean_squared_error(Y_test, Y_pred)))
    print('MAE: {}'.format(mean_absolute_error(Y_test,Y_pred)))
```

Modelo nº 1: Nº de avisos en función del Mes y el día

Paso nº 1: Obtenemos variables

In [225]:

```
#Define X and Y
data = alerts_allyears[["AÑO", "MES", "DIA", "SUCESO"]].groupby(["AÑO", "MES", "DIA"]).count()
data = data.reset_index()
data = data[(np.abs(stats.zscore(data)) < 3).all(axis=1)] #Remove outliers
X = data[["MES","DIA"]]
Y = data["SUCESO"]

#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "SUCESO"]].groupby(["AÑO", "MES", "DIA"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Remove outliers
data_train = data_train.reset_index()
Y_train = data_train["SUCESO"]
X_train = data_train[["MES","DIA"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "SUCESO"]].groupby(["AÑO", "MES", "DIA"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Remove outliers
data_test = data_test.reset_index()
Y_test = data_test["SUCESO"]
X_test = data_test[["MES","DIA"]]
```

In [226]:

```
data_train.head(5)
```

Out[226]:

	AÑO	MES	DIA	SUCESO
0	2009	1	1	522
1	2009	1	2	329
2	2009	1	3	322
3	2009	1	4	314
4	2009	1	5	355

In [227]:

```
print("#" * 50)
print(X_train[:5])
print("#" * 50)
print(X_test[:5])
```

```
##################################################
   MES  DIA
0    1    1
1    1    2
2    1    3
3    1    4
4    1    5
##################################################
   MES  DIA
0    1    1
1    1    2
2    1    3
3    1    4
4    1    5
```

In [228]:

```
print(X_train.shape)
print(Y_train.shape)
```

```
(3580, 2)
(3580,)
```

Paso nº 2: Creamos modelo y predecimos mediante la ecuación normal

In [229]:

```
normal_thetas = normal_ecuation(X_train,Y_train)
normal_thetas
```

Out[229]:

```
array([ 3.68979286e+02,  1.78127326e+00, -2.04433296e-01])
```

Paso nº 3 : Creamos modelo y predecimos con la librería de sklearn

In [230]:

```
model = LinearRegression()
model.fit(X_train,Y_train)
Y_pred = model.predict(X_test)
```

Paso nº 4 : Calculamos RMSE y MAE del modelo para esos datos de entrenamiento (2009-2017 \ 2019) y test(2018)

In [231]:

```
info_model(Y_test, Y_pred)
```

```
Coefficients:  [ 1.78127326 -0.2044333 ]
RMSE: 63.68
MAE: 49.55568812411586
```

Paso nº 5 : Calculamos RMSE y MAE del modelo con validación cruzada

In [232]:

```
scoring = {"mae": "neg_mean_absolute_error", "rmse": rmse_cross_val}
estimator = model
scores = cross_validate(estimator, X,
                        Y, scoring=scoring,
                         cv=100, return_train_score=True)
```

In [233]:

```
pd.DataFrame(scores).mean()
```

Out[233]:

```
fit_time       0.001765
score_time     0.001464
test_mae     -48.326585
train_mae    -48.114409
test_rmse     59.355493
train_rmse    60.507404
dtype: float64
```

In [234]:

```
print("[MODELO ECUACIÓN NORMAL] Para el uno de enero se estiman un total de {} avisos".format(normal_ecuation_pre
dict(normal_thetas,[1,1])))
print("[MODELO SKLEARN] Para el uno de enero se estiman un total de {} avisos".format(model.predict(pd.DataFrame(
{'col1': [1], 'col2': [1]}))[0]))
```

```
[MODELO ECUACIÓN NORMAL] Para el uno de enero se estiman un total de 370.5561262971938 avisos
[MODELO SKLEARN] Para el uno de enero se estiman un total de 370.5561262971986 avisos
```

Como podemos observar tanto la implementación del método normal como la propia de la librería de sklearn nos devuelven el mismo resultado. A continuación mostramos los datos predecidos y los reales para algunas muestras de 2018

In [235]:

```
pd.DataFrame({'Y_test': Y_test, 'Y_pred': Y_pred}).sample(5)
```

Out[235]:

	Y_test	Y_pred
9	334	368.716227
304	418	387.959992
154	325	378.644759
33	442	371.928533
31	406	372.337400

Paso nº6: Exportamos el modelo

In [236]:

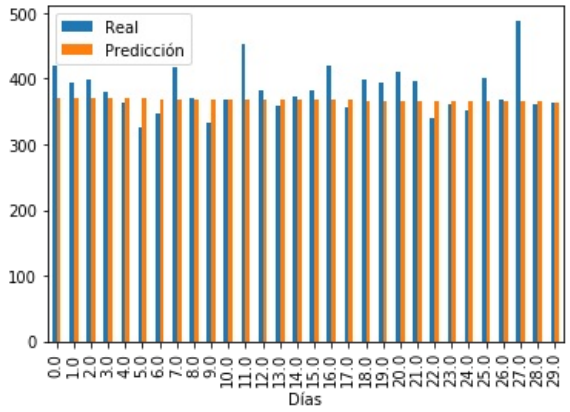
```
pickle.dump(model, open('RL-1.sav', 'wb'))
```

Paso nº7: Mostramos los datos reales y los datos predecidos para el primer mes de 2018

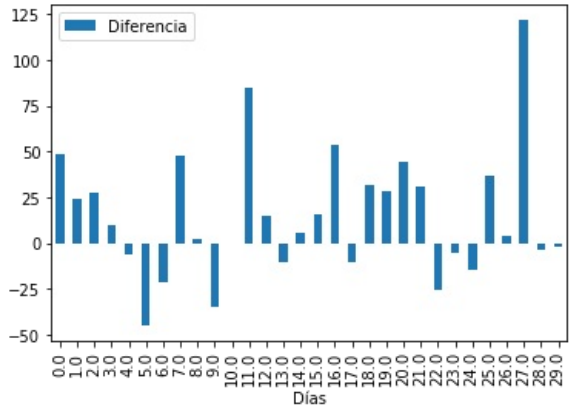
In [237]:

```
print_prediction(Y_test, Y_pred, 30)
```

Out[237]:



Out[237]:



Modelos nº 2: Nº de avisos para un Mes, un día, un día de la semana

Paso nº 1: Obtenemos variables

In [238]:

```
#Define X and Y
data = alerts_allyears[["AÑO", "MES", "DIA_CODIFICADO", "DIA", "SUCESO"]].groupby(["AÑO", "MES", "DIA", "DIA_CODI
FICADO"]).count()
data = data.reset_index()
data = data[(np.abs(stats.zscore(data)) < 3).all(axis=1)] #Remove outliers
X = data[["MES", "DIA", "DIA_CODIFICADO"]]
Y = data["SUCESO"]

#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA_CODIFICADO", "DIA", "SUCESO"]].groupby(["AÑO", "MES", "DIA", "DI
A_CODIFICADO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA_CODIFICADO", "DIA", "SUCESO"]].groupby(["AÑO", "MES", "DIA", "DIA_
CODIFICADO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO"]]
```

In [239]:

```
data_train.head(5)
```

Out[239]:

	AÑO	MES	DIA	DIA_CODIFICADO	SUCESO
0	2009	1	1	3	522
1	2009	1	2	4	329
2	2009	1	3	5	322
3	2009	1	4	6	314
4	2009	1	5	0	355

In [240]:

```
print("#" * 50)
print(X_train[:5])
print("#" * 50)
print(X_test[:5])
```

```
##################################################
   MES  DIA  DIA_CODIFICADO
0    1    1               3
1    1    2               4
2    1    3               5
3    1    4               6
4    1    5               0
##################################################
   MES  DIA  DIA_CODIFICADO
0    1    1               0
1    1    2               1
2    1    3               2
3    1    4               3
4    1    5               4
```

In [241]:

```
print(X_train.shape)
print(Y_train.shape)
```

```
(3586, 3)
(3586,)
```

Paso nº 2: Creamos modelo y predecimos mediante la ecuación normal

In [242]:

```
normal_thetas = normal_ecuation(X_train,Y_train)
normal_thetas
```

Out[242]:

```
array([ 3.33823992e+02,  1.76489355e+00, -2.52350219e-01,  1.20686561e+01])
```

Paso nº 3 : Creamos modelo y predecimos con la librería de sklearn

In [243]:

```
model = LinearRegression()
model.fit(X_train,Y_train)
Y_pred = model.predict(X_test)
```

Paso nº 4 : Calculamos RMSE y MAE del modelo para esos datos de entrenamiento (2009-2017 \ 2019) y test(2018)

In [244]:

```
info_model(Y_test,Y_pred)
```

```
Coefficients:  [ 1.76489355 -0.25235022 12.06865614]
RMSE: 59.28
MAE: 46.828947499251065
```

Paso nº 5 : Calculamos RMSE y MAE del modelo con validación cruzada

In [245]:

```
scoring = {"mae": "neg_mean_absolute_error", "rmse": rmse_cross_val}
estimator = model
scores = cross_validate(estimator, X,
                        Y, scoring=scoring,
                         cv=100, return_train_score=True)
```

In [246]:

```
pd.DataFrame(scores).mean()
```

Out[246]:

```
fit_time       0.001918
score_time     0.001577
test_mae     -44.375185
train_mae    -44.101062
test_rmse     54.429388
train_rmse    56.269491
dtype: float64
```

In [247]:

```
print("[MODELO ECUACIÓN NORMAL] Para el uno de enero (Lunes) se estiman un total de {} avisos".format(normal_ecua
tion_predict(normal_thetas,[1,1,0])))
print("[MODELO SKLEARN] Para el uno de enero (Lunes) se estiman un total de {} avisos".format(model.predict(pd.Da
taFrame({'col1': [1], 'col2': [1], 'col3': [0]}))[0]))
```

```
[MODELO ECUACIÓN NORMAL] Para el uno de enero (Lunes) se estiman un total de 335.3365354207504 aviso
s
[MODELO SKLEARN] Para el uno de enero (Lunes) se estiman un total de 335.33653542074353 avisos
```

A continuación mostramos los datos predecidos y los reales para algunas muestras de 2018

In [248]:

```
pd.DataFrame({'Y_test': Y_test, 'Y_pred': Y_pred}).sample(5)
```

Out[248]:

				Y_test	Y_pred
AÑO	MES	DIA	DIA_CODIFICADO		
2018	10	11	3	455	384.903044
	8	1	2	367	371.828103
	7	23	0	467	340.374192
	3	27	1	392	344.373873
	11	29	3	418	382.125633

Paso nº6: Exportamos el modelo

In [249]:

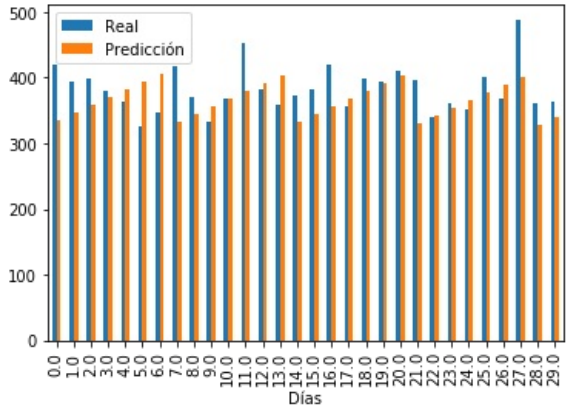
```
pickle.dump(model, open('RL-2.sav', 'wb'))
```

Paso nº7: Mostramos los datos reales y los datos predecidos para el primer mes de 2018

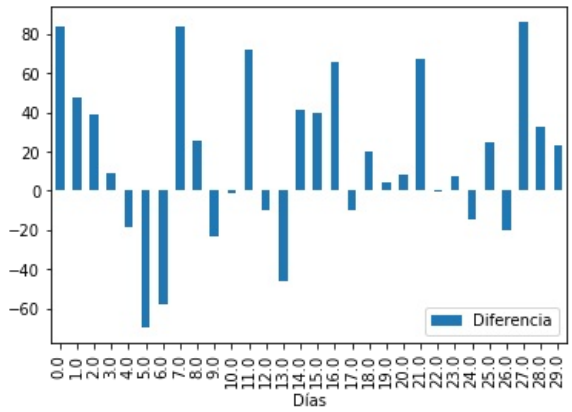
In [250]:

```
print_prediction(Y_test, Y_pred, 30)
```

Out[250]:



Out[250]:



Modelo nº 3: Nº de avisos para un Mes, día, dia de la semana y base

Paso nº 1: Obtenemos variables

In [251]:

```
#Define X and Y
data = alerts_allyears[["AÑO", "MES", "DIA", "DIA_CODIFICADO","BASE_CORREGIDA", "SUCESO"]].groupby(["AÑO", "MES",
"DIA", "DIA_CODIFICADO","BASE_CORREGIDA"]).count()
data = data.reset_index()
data = data[(np.abs(stats.zscore(data)) < 3).all(axis=1)] #Remove outliers
X = data[["MES", "DIA", "DIA_CODIFICADO","BASE_CORREGIDA"]]
Y = data["SUCESO"]

#Define X and Y for training

data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO","BASE_CORREGIDA", "SUCESO"]].groupby(["AÑO",
"MES", "DIA", "DIA_CODIFICADO","BASE_CORREGIDA"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO","BASE_CORREGIDA"]]

data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO","BASE_CORREGIDA", "SUCESO"]].groupby(["AÑO", "M
ES", "DIA", "DIA_CODIFICADO","BASE_CORREGIDA"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO","BASE_CORREGIDA"]]
```

In [252]:

```
data_train.head(5)
```

Out[252]:

	AÑO	MES	DIA	DIA_CODIFICADO	BASE_CORREGIDA	SUCESO
0	2009	1	1	3	-1	28
1	2009	1	1	3	3	31
2	2009	1	1	3	4	18
3	2009	1	1	3	5	22
4	2009	1	1	3	6	35

In [253]:

```
print("#" * 50)
print(X_train[:5])
print("#" * 50)
print(X_test[:5])
```

```
##################################################
   MES  DIA  DIA_CODIFICADO  BASE_CORREGIDA
0    1    1               3              -1
1    1    1               3               3
2    1    1               3               4
3    1    1               3               5
4    1    1               3               6
##################################################
   MES  DIA  DIA_CODIFICADO  BASE_CORREGIDA
0    1    1               0               1
1    1    1               0               2
2    1    1               0               5
3    1    1               0               7
4    1    1               0               8
```

In [254]:

```
print(X_train.shape)
print(Y_train.shape)
```

```
(71074, 4)
(71074,)
```

Paso nº 2: Creamos modelo y predecimos mediante la ecuación normal

In [255]:

```
normal_thetas = normal_ecuation(X_train,Y_train)
normal_thetas
```

Out[255]:

```
array([23.0528755 ,  0.10290525, -0.04685085,  0.461534  , -0.5607977 ])
```

Paso nº 3 : Creamos modelo y predecimos con la librería de sklearn

In [256]:

```
model = LinearRegression()
model.fit(X_train,Y_train)
Y_pred = model.predict(X_test)
```

Paso nº 4 : Calculamos RMSE y MAE del modelo para esos datos de entrenamiento (2009-2017 \ 2019) y test(2018)

In [257]:

```
info_model(Y_test,Y_pred)
```

```
Coefficients:  [ 0.10290525 -0.04685085  0.461534   -0.5607977 ]
RMSE: 10.23
MAE: 7.953080686884549
```

Paso nº 5 : Calculamos RMSE y MAE del modelo con validación cruzada

In [258]:

```
scoring = {"mae": "neg_mean_absolute_error", "rmse": rmse_cross_val}
estimator = model
scores = cross_validate(estimator, X,
                        Y, scoring=scoring,
                         cv=100, return_train_score=True)
```

In [259]:

```
pd.DataFrame(scores).mean()
```

Out[259]:

```
fit_time       0.004855
score_time     0.001719
test_mae      -8.476275
train_mae     -8.470001
test_rmse     10.299351
train_rmse    10.318222
dtype: float64
```

In [260]:

```
print("[MODELO ECUACIÓN NORMAL] Para un Lunes 1 de enero en la base 1 se estiman un total de {} avisos".format(no
rmal_ecuation_predict(normal_thetas,[1,1,0,1])))
print("[MODELO SKLEARN] Para un Lunes 1 de enero en la base 1 se estiman un total de {} avisos".format(model.pred
ict(pd.DataFrame({'col1': [1], 'col2': [1], 'col3': [0], 'col4': [0]}))[0]))
```

```
[MODELO ECUACIÓN NORMAL] Para un Lunes 1 de enero en la base 1 se estiman un total de 22.54813220475
005 avisos
[MODELO SKLEARN] Para un Lunes 1 de enero en la base 1 se estiman un total de 23.10892990455299 avis
os
```

A continuación mostramos los datos predecidos y los reales para algunas muestras de 2018

In [261]:

```
pd.DataFrame({'Y_test': Y_test, 'Y_pred': Y_pred}).sample(5)
```

Out[261]:

					Y_test	Y_pred
AÑO	MES	DIA	DIA_CODIFICADO	BASE_CORREGIDA		
2018	3	19	0	-1	34	23.032223
	7	31	1	-1	28	23.343168
	10	16	1	20	5	12.577894
	6	14	3	18	9	14.304638
	1	17	2	2	29	22.160789

Paso nº6: Exportamos el modelo

In [262]:

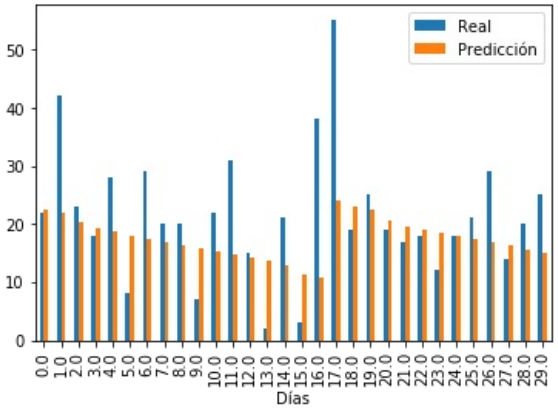
```
pickle.dump(model, open('RL-3.sav', 'wb'))
```

Paso nº7: Mostramos los datos reales y los datos predecidos para el primer mes de 2018

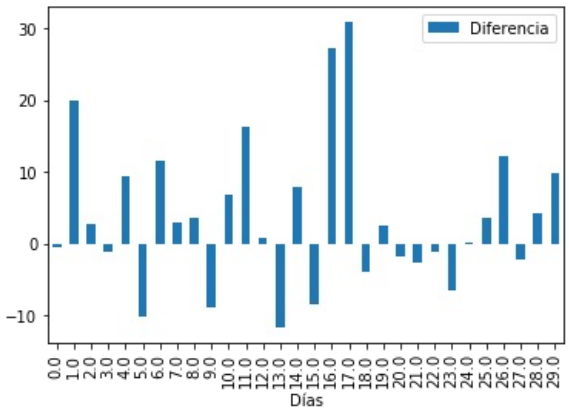
In [263]:

```
print_prediction(Y_test, Y_pred, 30)
```

Out[263]:



Out[263]:



Modelo nº 4: Nº de avisos para un Mes, día, día de la semana y distrito

Paso nº 1: Obtenemos variables

In [264]:

```
#Define X and Y
data = alerts_allyears[["AÑO", "MES", "DIA", "DIA_CODIFICADO","DISTRITO", "SUCESO"]].groupby(["AÑO", "MES", "DIA"
, "DIA_CODIFICADO","DISTRITO"]).count()
data = data.reset_index()
data = data[(np.abs(stats.zscore(data)) < 3).all(axis=1)] #Remove outliers
X = data[["MES", "DIA", "DIA_CODIFICADO","DISTRITO"]]
Y = data["SUCESO"]

#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO","DISTRITO", "SUCESO"]].groupby(["AÑO", "MES",
"DIA", "DIA_CODIFICADO","DISTRITO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO","DISTRITO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO","DISTRITO", "SUCESO"]].groupby(["AÑO", "MES", "
DIA", "DIA_CODIFICADO","DISTRITO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO","DISTRITO"]]
```

In [265]:

```
data_train.head(5)
```

Out[265]:

	AÑO	MES	DIA	DIA_CODIFICADO	DISTRITO	SUCESO
0	2009	1	1	3	0.0	3
1	2009	1	1	3	2.0	17
2	2009	1	1	3	3.0	9
3	2009	1	1	3	4.0	24
4	2009	1	1	3	5.0	26

In [266]:

```
print("#" * 50)
print(X_train[:5])
print("#" * 50)
print(X_test[:5])
```

```
##################################################
   MES  DIA  DIA_CODIFICADO  DISTRITO
0    1    1               3       0.0
1    1    1               3       2.0
2    1    1               3       3.0
3    1    1               3       4.0
4    1    1               3       5.0
##################################################
   MES  DIA  DIA_CODIFICADO  DISTRITO
0    1    1               0       2.0
1    1    1               0       3.0
2    1    1               0       4.0
3    1    1               0       5.0
4    1    1               0       6.0
```

In [267]:

```
print(X_train.shape)
print(Y_train.shape)
```

```
(75588, 4)
(75588,)
```

Paso nº 2: Creamos modelo y predecimos mediante la ecuación normal

In [268]:

```
normal_thetas = normal_ecuation(X_train,Y_train)
normal_thetas
```

Out[268]:

```
array([24.14705427,  0.03614092, -0.028293  ,  0.19160552, -0.74242762])
```

Paso nº 3 : Creamos modelo y predecimos con la librería de sklearn

In [269]:

```
model = LinearRegression()
model.fit(X_train,Y_train)
Y_pred = model.predict(X_test)
```

Paso nº 4 : Calculamos RMSE y MAE del modelo para esos datos de entrenamiento (2009-2017 \ 2019) y test(2018)

In [270]:

```
info_model(Y_test,Y_pred)
```

```
Coefficients:  [ 0.03614092 -0.028293    0.19160552 -0.74242762]
RMSE: 8.82
MAE: 6.616918177606124
```

Paso nº 5 : Calculamos RMSE y MAE del modelo con validación cruzada

In [271]:

```
scoring = {"mae": "neg_mean_absolute_error", "rmse": rmse_cross_val}
estimator = model
scores = cross_validate(estimator, X,
                        Y, scoring=scoring,
                         cv=100, return_train_score=True)
```

In [272]:

```
pd.DataFrame(scores).mean()
```

Out[272]:

```
fit_time      0.006267
score_time    0.002184
test_mae     -6.605206
train_mae    -6.601651
test_rmse     8.595199
train_rmse    8.601261
dtype: float64
```

In [273]:

```
print("[MODELO ECUACIÓN NORMAL] Para un Lunes 1 de enero en el distrito 1 se estiman un total de {} avisos".forma
t(normal_ecuation_predict(normal_thetas,[1,1,0,1])))
print("[MODELO SKLEARN] Para un Lunes 1 de enero en el distrito 1 se estiman un total de {} avisos".format(model.
predict(pd.DataFrame({'col1': [1], 'col2': [1], 'col3': [0], 'col4': [1]}))[0]))
```

```
[MODELO ECUACIÓN NORMAL] Para un Lunes 1 de enero en el distrito 1 se estiman un total de 23.4124745
72031996 avisos
[MODELO SKLEARN] Para un Lunes 1 de enero en el distrito 1 se estiman un total de 23.412474572032075
avisos
```

In [274]:

```
pd.DataFrame({'Y_test': Y_test, 'Y_pred': Y_pred}).sample(5)
```

Out[274]:

					Y_test	Y_pred
AÑO	MES	DIA	DIA_CODIFICADO	DISTRITO		
2018	11	11	6	9.0	29	18.701166
	4	6	4	2.0	32	23.403427
	7	16	0	8.0	15	18.007932
	12	17	0	18.0	3	10.736067
	11	29	3	9.0	24	17.617075

Paso nº6: Exportamos el modelo

In [275]:

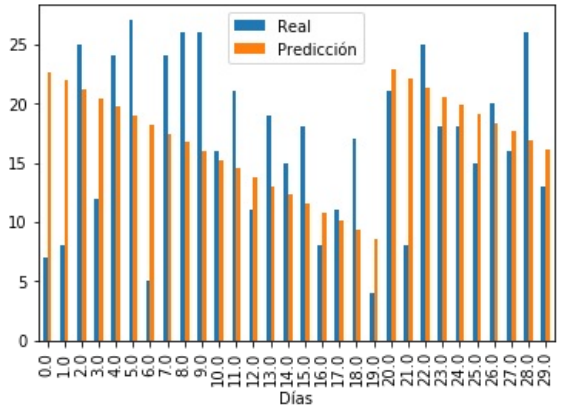
```
pickle.dump(model, open('RL-4.sav', 'wb'))
```

Paso nº7: Mostramos los datos reales y los datos predecidos para el primer mes de 2018

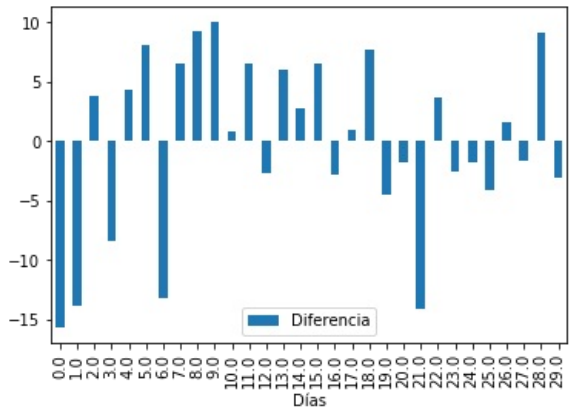
In [276]:

```
print_prediction(Y_test, Y_pred, 30)
```

Out[276]:



Out[276]:



Modelo nº 5 : Nº de avisos para un Mes, día, día de la semana y turno

Paso nº 1: Obtenemos variables

In [277]:

```
#Define X and Y
data = alerts_allyears[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO", "SUCESO"]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO"]).count()
data = data.reset_index()
data = data[(np.abs(stats.zscore(data)) < 3).all(axis=1)] #Remove outliers
X = data[["MES", "DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO"]]
Y = data["SUCESO"]

#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO", "SUCESO"]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO", "SUCESO"]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO"]]
```

In [278]:

```
data_train.head(5)
```

Out[278]:

	AÑO	MES	DIA	DIA_CODIFICADO	TURNO_CODIFICADO	SUCESO
0	2009	1	1	3	0	190
1	2009	1	1	3	1	116
2	2009	1	1	3	2	216
3	2009	1	2	4	0	147
4	2009	1	2	4	1	135

In [279]:

```
print("#" * 50)
print(X_train[:5])
print("#" * 50)
print(X_test[:5])
```

```
##################################################
   MES  DIA  DIA_CODIFICADO  TURNO_CODIFICADO
0    1    1               3                 0
1    1    1               3                 1
2    1    1               3                 2
3    1    2               4                 0
4    1    2               4                 1
##################################################
   MES  DIA  DIA_CODIFICADO  TURNO_CODIFICADO
0    1    1               0                 0
1    1    1               0                 1
2    1    1               0                 2
3    1    2               1                 0
4    1    2               1                 1
```

In [280]:

```
print(X_train.shape)
print(Y_train.shape)
```

```
(10988, 4)
(10988,)
```

Paso nº 2: Creamos modelo y predecimos mediante la ecuación normal

In [281]:

```
normal_thetas = normal_ecuation(X_train,Y_train)
normal_thetas
```

Out[281]:

```
array([153.31168754,   0.60210871,  -0.43918437,   3.84501978,
       -37.9244844 ])
```

Paso nº 3 : Creamos modelo y predecimos con la librería de sklearn

In [282]:

```
model = LinearRegression()
model.fit(X_train,Y_train)
Y_pred = model.predict(X_test)
```

Paso nº 4 : Calculamos RMSE y MAE del modelo para esos datos de entrenamiento (2009-2017 \ 2019) y test(2018)

In [283]:

```
info_model(Y_test,Y_pred)
```

```
Coefficients:  [  0.60210871  -0.43918437   3.84501978 -37.9244844 ]
RMSE: 44.02
MAE: 35.73217718790271
```

Paso nº 5 : Calculamos RMSE y MAE del modelo con validación cruzada

In [284]:

```
scoring = {"mae": "neg_mean_absolute_error", "rmse": rmse_cross_val}
estimator = model
scores = cross_validate(estimator, X,
                        Y, scoring=scoring,
                         cv=100, return_train_score=True)
```

In [285]:

```
pd.DataFrame(scores).mean()
```

Out[285]:

```
fit_time       0.002068
score_time     0.001432
test_mae     -33.524432
train_mae    -33.476265
test_rmse     41.034374
train_rmse    41.184239
dtype: float64
```

In [286]:

```
print("[MODELO ECUACIÓN NORMAL] Para un Lunes 1 de enero en el turno de noche se estiman un total de {} avisos".f
ormat(normal_ecuation_predict(normal_thetas,[1,1,0,2])))
print("[MODELO SKLEARN] Para un Lunes 1 de enero en el turno de noche se estiman un total de {} avisos".format(mo
del.predict(pd.DataFrame({'col1': [1], 'col2': [1], 'col3': [0], 'col4': [2]}))[0]))
```

```
[MODELO ECUACIÓN NORMAL] Para un Lunes 1 de enero en el turno de noche se estiman un total de 77.625
6430737378 avisos
[MODELO SKLEARN] Para un Lunes 1 de enero en el turno de noche se estiman un total de 77.62564307373
788 avisos
```

In [287]:

```
pd.DataFrame({'Y_test': Y_test, 'Y_pred': Y_pred}).sample(5)
```

Out[287]:

					Y_test	Y_pred
AÑO	MES	DIA	DIA_CODIFICADO	TURNO_CODIFICADO		
2018	4	27	4	0	162	159.242223
	8	7	1	0	136	158.899286
		17	4	2	61	90.193533
	4	4	2	2	27	85.804456
		11	2	0	181	158.579134

Paso nº6: Exportamos el modelo

In [288]:

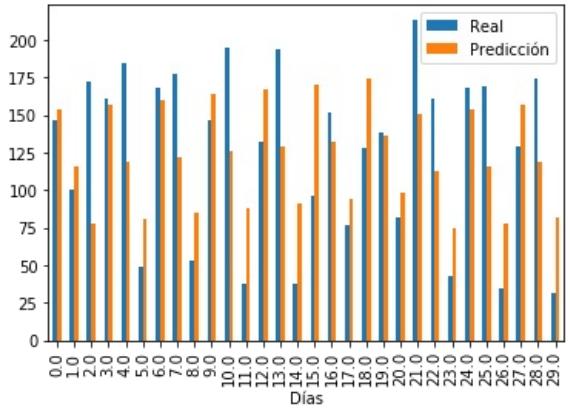
```
pickle.dump(model, open('RL-5.sav', 'wb'))
```

Paso nº7: Mostramos los datos reales y los datos predecidos para el primer mes de 2018

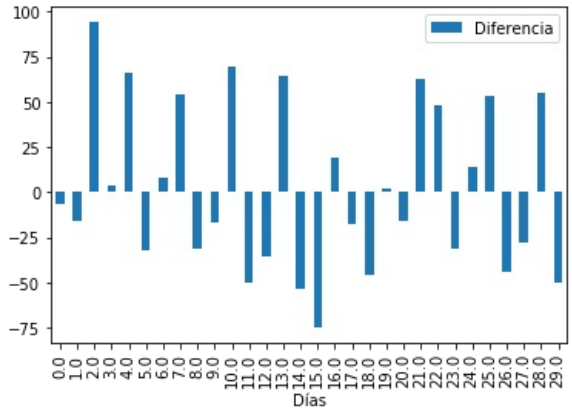
In [289]:

```
print_prediction(Y_test, Y_pred, 30)
```

Out[289]:



Out[289]:



Modelo nº 6: Nº de avisos para un Mes, día, dia de la semana, base y turno

Paso nº 1: Obtenemos variables

In [290]:

```
#Define X and Y
data = alerts_allyears[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "TURNO_CODIFICADO", "SUCESO"]].g
roupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "TURNO_CODIFICADO"]).count()
data = data.reset_index()
data = data[(np.abs(stats.zscore(data)) < 3).all(axis=1)] #Remove outliers
X = data[["MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "TURNO_CODIFICADO"]]
Y = data["SUCESO"]

#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "TURNO_CODIFICADO", "SUCES
O"]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "TURNO_CODIFICADO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "TURNO_CODIFICADO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "TURNO_CODIFICADO", "SUCESO"
]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "TURNO_CODIFICADO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "TURNO_CODIFICADO"]]
```

In [291]:

```
print("#" * 50)
print(X_train[:5])
print("#" * 50)
print(X_test[:5])
```

```
##################################################
   MES  DIA  DIA_CODIFICADO  BASE_CORREGIDA  TURNO_CODIFICADO
0    1    1               3              -1                 0
1    1    1               3              -1                 1
2    1    1               3              -1                 2
3    1    1               3               1                 0
4    1    1               3               1                 1
##################################################
   MES  DIA  DIA_CODIFICADO  BASE_CORREGIDA  TURNO_CODIFICADO
0    1    1               0              -1                 0
1    1    1               0              -1                 1
2    1    1               0               1                 0
3    1    1               0               1                 1
4    1    1               0               1                 2
```

In [292]:

```
print(X_train.shape)
print(Y_train.shape)
```

```
(181943, 5)
(181943,)
```

Paso nº 2: Creamos modelo y predecimos mediante la ecuación normal

In [293]:

```
normal_thetas = normal_ecuation(X_train,Y_train)
normal_thetas
```

Out[293]:

```
array([ 9.55654085e+00,  4.24518740e-02, -8.85283340e-03,  1.20328847e-01,
       -1.73633137e-01, -1.25910345e+00])
```

Paso nº 3 : Creamos modelo y predecimos con la librería de sklearn

In [294]:

```
model = LinearRegression()
model.fit(X_train,Y_train)
Y_pred = model.predict(X_test)
```

Paso nº 4 : Calculamos RMSE y MAE del modelo para esos datos de entrenamiento (2009-2017 \ 2019) y test(2018)

In [295]:

```
info_model(Y_test,Y_pred)
```

```
Coefficients:  [ 0.04245187 -0.00885283  0.12032885 -0.17363314 -1.25910345]
RMSE: 4.44
MAE: 3.4214187832888774
```

Paso nº 5 : Calculamos RMSE y MAE del modelo con validación cruzada

In [296]:

```
scoring = {"mae": "neg_mean_absolute_error", "rmse": rmse_cross_val}
estimator = model
scores = cross_validate(estimator, X,
                        Y, scoring=scoring,
                         cv=100, return_train_score=True)
```

In [297]:

```
pd.DataFrame(scores).mean()
```

Out[297]:

```
fit_time      0.013055
score_time    0.001750
test_mae     -3.365493
train_mae    -3.363141
test_rmse     4.186263
train_rmse    4.194009
dtype: float64
```

In [298]:

```
print("[MODELO ECUACIÓN NORMAL] Para un Sabado 1 de enero en la base 1 se estiman un total de {} avisos el turno
de noche".format(normal_ecuation_predict(normal_thetas,[1,1,5,1,2])))
print("[MODELO SKLEARN] Para un Sabado 1 de enero en la base 1 se estiman un total de {} avisos el turno de noche
".format(model.predict(pd.DataFrame({'col1': [1], 'col2': 1, 'col3': [5], 'col4': [1], 'col5': [2]}))[0]))
```

```
[MODELO ECUACIÓN NORMAL] Para un Sabado 1 de enero en la base 1 se estiman un total de 7.49994409007
8051 avisos el turno de noche
[MODELO SKLEARN] Para un Sabado 1 de enero en la base 1 se estiman un total de 7.499944090077918 avi
sos el turno de noche
```

In [299]:

```
pd.DataFrame({'Y_test': Y_test, 'Y_pred': Y_pred}).sample(5)
```

Out[299]:

						Y_test	Y_pred
AÑO	MES	DIA	DIA_CODIFICADO	BASE_CORREGIDA	TURNO_CODIFICADO		
2018	4	8	6	21	0	1	6.731203
	3	23	4	19	0	5	6.662567
	7	16	0	14	1	4	6.022091
	3	4	6	21	0	2	6.724162
	8	24	4	22	0	12	6.345074

Paso nº6: Exportamos el modelo

In [300]:

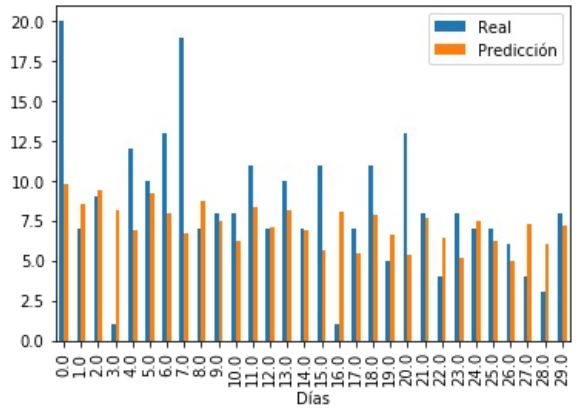
```
pickle.dump(model, open('RL-6.sav', 'wb'))
```

Paso nº7: Mostramos los datos reales y los datos predecidos para el primer mes de 2018

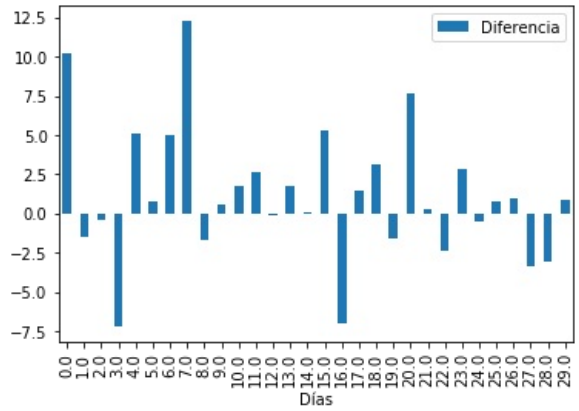
In [301]:

```
print_prediction(Y_test, Y_pred, 30)
```

Out[301]:



Out[301]:



Modelo nº 7: nº de Avisos para un mes, día , día de la semana, base y dispositivo

Paso nº 1: Obtenemos variables

In [302]:

```
#Define X and Y
data = alerts_allyears[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "ABREV_CODIFICADO", "SUCESO"]].g
roupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "ABREV_CODIFICADO"]).count()
data = data.reset_index()
data = data[(np.abs(stats.zscore(data)) < 3).all(axis=1)] #Remove outliers
X = data[["MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "ABREV_CODIFICADO"]]
Y = data["SUCESO"]

#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "ABREV_CODIFICADO", "SUCES
O"]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "ABREV_CODIFICADO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "ABREV_CODIFICADO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "ABREV_CODIFICADO", "SUCESO"
]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "ABREV_CODIFICADO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "ABREV_CODIFICADO"]]
```

In [303]:

```
print("#" * 50)
print(X_train[:5])
print("#" * 50)
print(X_test[:5])
```

```
##################################################
   MES  DIA  DIA_CODIFICADO  BASE_CORREGIDA  ABREV_CODIFICADO
0    1    1               3              -1               0.0
1    1    1               3              -1               2.0
2    1    1               3              -1              20.0
3    1    1               3              -1              28.0
4    1    1               3              -1              30.0
##################################################
   MES  DIA  DIA_CODIFICADO  BASE_CORREGIDA  ABREV_CODIFICADO
0    1    1               0              -1               1.0
1    1    1               0              -1               2.0
2    1    1               0              -1              17.0
3    1    1               0              -1              28.0
4    1    1               0              -1              30.0
```

In [304]:

```
print(X_train.shape)
print(Y_train.shape)
```

```
(154816, 5)
(154816,)
```

Paso nº 2: Creamos modelo y predecimos mediante la ecuación normal

In [305]:

```
normal_thetas = normal_ecuation(X_train,Y_train)
normal_thetas
```

Out[305]:

```
array([ 9.2005595 ,  0.06312296, -0.01608942,  0.09874278, -0.04923552,
       -0.18980698])
```

Paso nº 3 : Creamos modelo y predecimos con la librería de sklearn

In [306]:

```
model = LinearRegression()
model.fit(X_train,Y_train)
Y_pred = model.predict(X_test)
```

Paso nº 4 : Calculamos RMSE y MAE del modelo para esos datos de entrenamiento (2009-2017 \ 2019) y test(2018)

In [307]:

```
info_model(Y_test,Y_pred)
```

```
Coefficients:  [ 0.06312296 -0.01608942  0.09874278 -0.04923552 -0.18980698]
RMSE: 5.54
MAE: 4.235293196411715
```

Paso nº 5 : Calculamos RMSE y MAE del modelo con validación cruzada

In [308]:

```
scoring = {"mae": "neg_mean_absolute_error", "rmse": rmse_cross_val}
estimator = model
scores = cross_validate(estimator, X,
                        Y, scoring=scoring,
                         cv=100, return_train_score=True)
```

In [309]:

```
pd.DataFrame(scores).mean()
```

Out[309]:

```
fit_time      0.011774
score_time    0.001845
test_mae     -3.985829
train_mae    -3.983335
test_rmse     5.038017
train_rmse    5.046201
dtype: float64
```

In [310]:

```
print("[MODELO ECUACIÓN NORMAL] Para un Lunes 1 de enero en la base 1 de tipo SVB se estiman un total de {}
avisos".format(normal_ecuation_predict(normal_thetas,[1,1,0,1,0])))
print("[MODELO SKLEARN] Para un Sabado 1 de enero en la base 1 se estiman un total de {} avisos el turno de noche
".format(model.predict(pd.DataFrame({'col1': [1], 'col2': 1, 'col3': [0], 'col4': [1], 'col5': [0]}))[0]))
```

```
[MODELO ECUACIÓN NORMAL] Para un Lunes 1 de enero en la base 1 de tipo SVB se estiman un total de 9.
198357525946083 avisos
[MODELO SKLEARN] Para un Sabado 1 de enero en la base 1 se estiman un total de 9.198357525946639 avi
sos el turno de noche
```

In [311]:

```
pd.DataFrame({'Y_test': Y_test, 'Y_pred': Y_pred}).sample(5)
```

Out[311]:

						Y_test	Y_pred
AÑO	MES	DIA	DIA_CODIFICADO	BASE_CORREGIDA	ABREV_CODIFICADO		
2018	5	4	4	7	0.0	11	9.502139
	1	18	3	22	4.0	3	7.427892
	9	11	1	7	0.0	8	9.345777
	12	22	5	-1	23.0	1	5.781457
		7	4	15	0.0	1	9.501847

Paso nº6: Exportamos el modelo

In [312]:

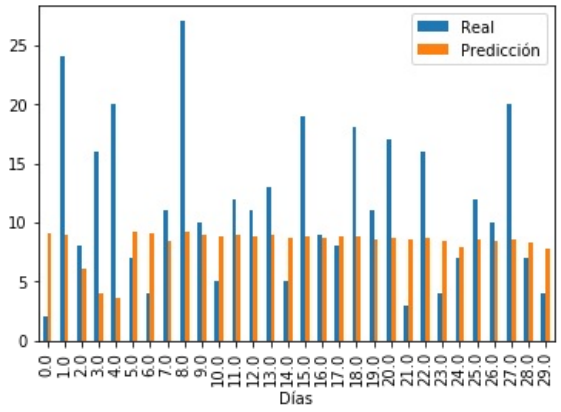
```
pickle.dump(model, open('RL-7.sav', 'wb'))
```

Paso nº7: Mostramos los datos reales y los datos predecidos para el primer mes de 2018

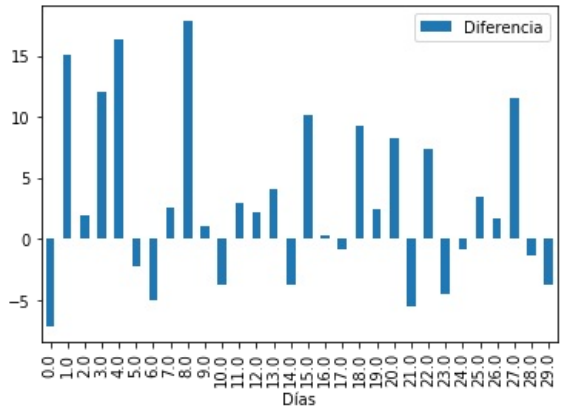
In [313]:

```
print_prediction(Y_test, Y_pred, 30)
```

Out[313]:



Out[313]:



Modelo nº 8: nº de Avisos para un mes, día , día de la semana, distrito y dispositivo

Paso nº 1: Obtenemos variables

In [314]:

```
#Define X and Y
data = alerts_allyears[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DISTRITO", "ABREV_CODIFICADO", "SUCESO"]].groupby
(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DISTRITO", "ABREV_CODIFICADO"]).count()
data = data.reset_index()
data = data[(np.abs(stats.zscore(data)) < 3).all(axis=1)] #Remove outliers
X = data[["MES", "DIA", "DIA_CODIFICADO", "DISTRITO", "ABREV_CODIFICADO"]]
Y = data["SUCESO"]

#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DISTRITO", "ABREV_CODIFICADO", "SUCESO"]].g
roupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DISTRITO", "ABREV_CODIFICADO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO", "DISTRITO", "ABREV_CODIFICADO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DISTRITO", "ABREV_CODIFICADO", "SUCESO"]].gro
upby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DISTRITO", "ABREV_CODIFICADO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO", "DISTRITO", "ABREV_CODIFICADO"]]
```

In [315]:

```
print("#" * 50)
print(X_train[:5])
print("#" * 50)
print(X_test[:5])
```

```
##################################################
   MES  DIA  DIA_CODIFICADO  DISTRITO  ABREV_CODIFICADO
0    1    1               3       0.0               0.0
1    1    1               3       0.0               1.0
2    1    1               3       1.0              20.0
3    1    1               3       1.0              30.0
4    1    1               3       2.0               0.0
##################################################
   MES  DIA  DIA_CODIFICADO  DISTRITO  ABREV_CODIFICADO
0    1    1               0       1.0               1.0
1    1    1               0       1.0               2.0
2    1    1               0       1.0               4.0
3    1    1               0       1.0              17.0
4    1    1               0       1.0              28.0
```

In [316]:

```
print(X_train.shape)
print(Y_train.shape)
```

```
(292191, 5)
(292191,)
```

Paso nº 2: Creamos modelo y predecimos mediante la ecuación normal

In [317]:

```
normal_thetas = normal_ecuation(X_train,Y_train)
normal_thetas
```

Out[317]:

```
array([ 6.60923012e+00,  6.62959883e-03, -4.96878290e-03,  2.38826966e-02,
       -1.13366318e-01, -1.77309511e-01])
```

Paso nº 3 : Creamos modelo y predecimos con la librería de sklearn

In [318]:

```
pickle.dump(model, open('NN/NN-11.sav', 'wb'))model = LinearRegression()
model.fit(X_train,Y_train)
Y_pred = model.predict(X_test)
```

Paso nº 4 : Calculamos RMSE y MAE del modelo para esos datos de entrenamiento (2009-2017 \ 2019) y test(2018)º

In [319]:

```
info_model(Y_test,Y_pred)
```

```
Coefficients:  [ 0.0066296  -0.00496878  0.0238827  -0.11336632 -0.17730951]
RMSE: 3.70
MAE: 2.6659227166511856
```

Paso nº 5 : Calculamos RMSE y MAE del modelo con validación cruzada

In [320]:

```
scoring = {"mae": "neg_mean_absolute_error", "rmse": rmse_cross_val}
estimator = model
scores = cross_validate(estimator, X,
                        Y, scoring=scoring,
                         cv=100, return_train_score=True)
```

In [321]:

```
pd.DataFrame(scores).mean()
```

Out[321]:

```
fit_time      0.027923
score_time    0.002527
test_mae     -2.598303
train_mae    -2.597732
test_rmse     3.482221
train_rmse    3.486937
dtype: float64
```

In [322]:

```
print("[MODELO ECUACIÓN NORMAL] Para un Lunes 1 de enero en la distrito 1 se estiman un total de {} avisos para ambulancias del tipo SBV".format(normal_ecuation_predict(normal_thetas,[1,1,0,1,0])))
print("[MODELO SKLEARN] Para un Lunes 1 de enero en la distrito 1 se estiman un total de {} avisos para ambulancias del tipo SBV".format(model.predict(pd.DataFrame({'col1': [1], 'col2': 1, 'col3': [0], 'col4': [1], 'col5': [0]}))[0]))
```

```
[MODELO ECUACIÓN NORMAL] Para un Lunes 1 de enero en la distrito 1 se estiman un total de 6.49752462
1380319 avisos para ambulancias del tipo SBV
[MODELO SKLEARN] Para un Lunes 1 de enero en la distrito 1 se estiman un total de 6.497524621380556
avisos para ambulancias del tipo SBV
```

In [323]:

```
pd.DataFrame({'Y_test': Y_test, 'Y_pred': Y_pred}).sample(5)
```

Out[323]:

						Y_test	Y_pred
AÑO	MES	DIA	DIA_CODIFICADO	DISTRITO	ABREV_CODIFICADO		
2018	11	29	3	12.0	0.0	5	5.249313
	10	26	4	17.0	0.0	6	4.714641
	1	4	3	1.0	1.0	8	6.376957
	10	3	2	19.0	30.0	1	-0.764860
		1	0	6.0	4.0	2	5.281121

Paso nº6: Exportamos el modelo

In [324]:

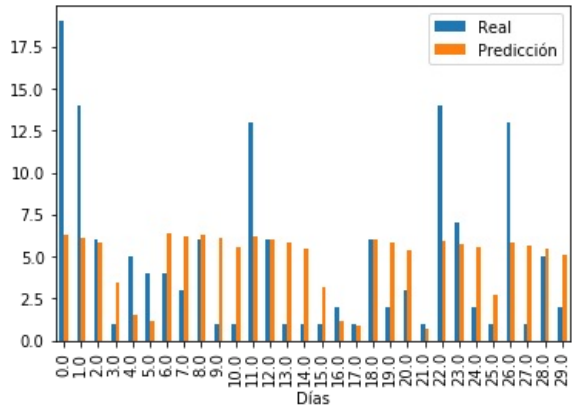
```
pickle.dump(model, open('RL-8.sav', 'wb'))
```

Paso nº7: Mostramos los datos reales y los datos predecidos para el primer mes de 2018

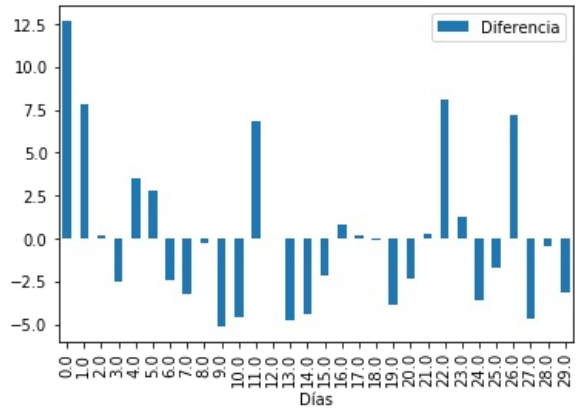
In [325]:

```
print_prediction(Y_test, Y_pred, 30)
```

Out[325]:



Out[325]:



Modelo nº 9: nº de Avisos para un mes, día , día de la semana y dispositivo

Paso nº 1: Obtenemos variables

In [326]:

```
#Define X and Y
data = alerts_allyears[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "ABREV_CODIFICADO", "SUCESO"]].groupby(["AÑO", "ME
S", "DIA", "DIA_CODIFICADO", "ABREV_CODIFICADO"]).count()
data = data.reset_index()
data = data[(np.abs(stats.zscore(data)) < 3).all(axis=1)] #Remove outliers
X = data[["MES", "DIA", "DIA_CODIFICADO", "ABREV_CODIFICADO"]]
Y = data["SUCESO"]

#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "ABREV_CODIFICADO", "SUCESO"]].groupby(["AÑO
", "MES", "DIA", "DIA_CODIFICADO", "ABREV_CODIFICADO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO", "ABREV_CODIFICADO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "ABREV_CODIFICADO", "SUCESO"]].groupby(["AÑO",
"MES", "DIA", "DIA_CODIFICADO", "ABREV_CODIFICADO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO", "ABREV_CODIFICADO"]]
```

In [327]:

```
print("#" * 50)
print(X_train[:5])
print("#" * 50)
print(X_test[:5])
```

```
##################################################
   MES  DIA  DIA_CODIFICADO  ABREV_CODIFICADO
0    1    1               3               1.0
1    1    1               3               2.0
2    1    1               3              18.0
3    1    1               3              20.0
4    1    1               3              28.0
##################################################
   MES  DIA  DIA_CODIFICADO  ABREV_CODIFICADO
0    1    1               0               0.0
1    1    1               0               1.0
2    1    1               0               2.0
3    1    1               0               4.0
4    1    1               0              17.0
```

In [328]:

```
print(X_train.shape)
print(Y_train.shape)
```

```
(36822, 4)
(36822,)
```

Paso nº 2: Creamos modelo y predecimos mediante la ecuación normal

In [329]:

```
normal_thetas = normal_ecuation(X_train,Y_train)
normal_thetas
```

Out[329]:

```
array([75.37025297, -0.17416895, -0.09854685, -0.19463651, -2.85111426])
```

Paso nº 3 : Creamos modelo y predecimos con la librería de sklearn

In [330]:

```
model = LinearRegression()
model.fit(X_train,Y_train)
Y_pred = model.predict(X_test)
```

Paso nº 4 : Calculamos RMSE y MAE del modelo para esos datos de entrenamiento (2009-2017 \ 2019) y test(2018)

In [331]:

```
info_model(Y_test,Y_pred)
```

```
Coefficients:  [-0.17416895 -0.09854685 -0.19463651 -2.85111426]
RMSE: 43.01
MAE: 30.198921796149307
```

Paso nº 5 : Calculamos RMSE y MAE del modelo con validación cruzada

In [332]:

```
scoring = {"mae": "neg_mean_absolute_error", "rmse": rmse_cross_val}
estimator = model
scores = cross_validate(estimator, X,
                        Y, scoring=scoring,
                         cv=100, return_train_score=True)
```

In [333]:

```
pd.DataFrame(scores).mean()
```

Out[333]:

```
fit_time       0.004261
score_time     0.001981
test_mae     -28.970393
train_mae    -28.963979
test_rmse     40.707315
train_rmse    40.783684
dtype: float64
```

In [334]:

```
print("[MODELO ECUACIÓN NORMAL] Para un Lunes 1 de enero se estiman un total de {} avisos de ambulancias del tipo SVB".format(normal_ecuation_predict(normal_thetas,[1,1,0,0])))
print("[MODELO SKLEARN] Para un Lunes 1 de enero se estiman un total de {} avisos deambulancias del tipo SVB".format(model.predict(pd.DataFrame({'col1': [1], 'col2': 1, 'col3': [0], 'col4': [0]}))[0]))
```

```
[MODELO ECUACIÓN NORMAL] Para un Lunes 1 de enero se estiman un total de 75.09753717347755 avisos de ambulancias del tipo SVB
[MODELO SKLEARN] Para un Lunes 1 de enero se estiman un total de 75.09753717346992 avisos deambulancias del tipo SVB
```

In [335]:

```
pd.DataFrame({'Y_test': Y_test, 'Y_pred': Y_pred}).sample(5)
```

Out[335]:

					Y_test	Y_pred
AÑO	MES	DIA	DIA_CODIFICADO	ABREV_CODIFICADO		
2018	4	11	2	24.0	2	4.773547
	6	23	5	28.0	6	-8.745720
	1	22	0	1.0	96	70.176939
	10	2	1	2.0	22	67.534605
		5	4	7.0	2	52.399483

Paso nº6: Exportamos el modelo

In [336]:

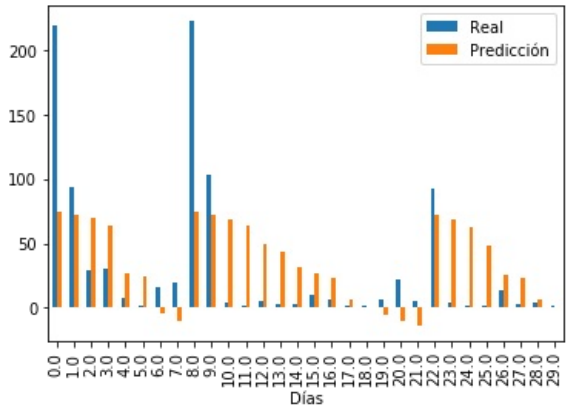
```
pickle.dump(model, open('RL-9.sav', 'wb'))
```

Paso nº7: Mostramos los datos reales y los datos predecidos para el primer mes de 2018

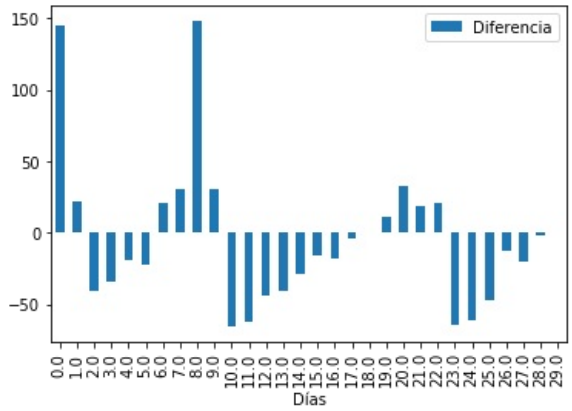
In [337]:

```
print_prediction(Y_test, Y_pred, 30)
```

Out[337]:



Out[337]:



Modelo nº 10: nº de Avisos para un mes, día , día de la semana, base, turno y dispositivo

Paso nº 1: Obtenemos variables

In [338]:

```
#Define X and Y
data = alerts_allyears[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "TURNO_CODIFICADO", "ABREV_CODIF
ICADO", "SUCESO"]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "TURNO_CODIFICADO", "ABREV_C
ODIFICADO"]).count()
data = data.reset_index()
data = data[(np.abs(stats.zscore(data)) < 3).all(axis=1)] #Remove outliers
X = data[["MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "TURNO_CODIFICADO", "ABREV_CODIFICADO"]]
Y = data["SUCESO"]

#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "TURNO_CODIFICADO", "ABREV
_CODIFICADO", "SUCESO"]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "TURNO_CODIFICADO", "A
BREV_CODIFICADO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "TURNO_CODIFICADO", "ABREV_CODIFICADO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "TURNO_CODIFICADO", "ABREV_C
ODIFICADO", "SUCESO"]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "TURNO_CODIFICADO", "ABR
EV_CODIFICADO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "TURNO_CODIFICADO", "ABREV_CODIFICADO"]]
```

In [339]:

```
print("#" * 50)
print(X_train[:5])
print("#" * 50)
print(X_test[:5])
```

```
##################################################
   MES  DIA  DIA_CODIFICADO  BASE_CORREGIDA  TURNO_CODIFICADO  \
0    1    1               3              -1                 0
1    1    1               3              -1                 0
2    1    1               3              -1                 0
3    1    1               3              -1                 0
4    1    1               3              -1                 1

   ABREV_CODIFICADO
0               0.0
1              20.0
2              28.0
3              30.0
4              20.0
##################################################
   MES  DIA  DIA_CODIFICADO  BASE_CORREGIDA  TURNO_CODIFICADO  \
0    1    1               0              -1                 0
1    1    1               0              -1                 0
2    1    1               0              -1                 0
3    1    1               0              -1                 0
4    1    1               0              -1                 1

   ABREV_CODIFICADO
0               2.0
1              17.0
2              28.0
3              30.0
4              17.0
```

In [340]:

```
print(X_train.shape)
print(Y_train.shape)
```

```
(329170, 6)
(329170,)
```

Paso nº 2: Creamos modelo y predecimos mediante la ecuación normal

In [341]:

```
normal_thetas = normal_ecuation(X_train,Y_train)
normal_thetas
```

Out[341]:

```
array([ 4.57324759e+00,  2.97943430e-02, -3.88372846e-03,  4.24419692e-02,
       -2.99446949e-02, -4.25833201e-01, -7.22271956e-02])
```

Paso nº 3 : Creamos modelo y predecimos con la librería de sklearn

In [342]:

```
model = LinearRegression()
model.fit(X_train,Y_train)
Y_pred = model.predict(X_test)
```

Paso nº 4 : Calculamos RMSE y MAE del modelo para esos datos de entrenamiento (2009-2017 \ 2019) y test(2018)

In [343]:

```
info_model(Y_test,Y_pred)
```

```
Coefficients:  [ 0.02979434 -0.00388373  0.04244197 -0.02994469 -0.4258332  -0.0722272 ]
RMSE: 2.43
MAE: 1.8940791387935303
```

Paso nº 5 : Calculamos RMSE y MAE del modelo con validación cruzada

In [344]:

```
scoring = {"mae": "neg_mean_absolute_error", "rmse": rmse_cross_val}
estimator = model
scores = cross_validate(estimator, X,
                        Y, scoring=scoring,
                         cv=100, return_train_score=True)
```

In [345]:

```
pd.DataFrame(scores).mean()
```

Out[345]:

```
fit_time      0.036656
score_time    0.002770
test_mae     -1.762102
train_mae    -1.761315
test_rmse     2.225873
train_rmse    2.230413
dtype: float64
```

In [346]:

```
print("[MODELO ECUACIÓN NORMAL] Para un Lunes 1 de enero en la base 1 se estiman un total de {} avisos para ambul
ancias SBV en el turno de noche".format(normal_ecuation_predict(normal_thetas,[1,1,0,1,2,0])))
print("[MODELO SKLEARN] Para un Lunes 1 de enero en la base 1 se estiman un total de {} avisos para ambulancias S
BV en el turno de noche".format(model.predict(pd.DataFrame({'col1': [1], 'col2': 1, 'col3': [0], 'col4': [1], 'co
l5': [2], 'col': [0]}))[0]))
```

```
[MODELO ECUACIÓN NORMAL] Para un Lunes 1 de enero en la base 1 se estiman un total de 3.717547108968
7426 avisos para ambulancias SBV en el turno de noche
[MODELO SKLEARN] Para un Lunes 1 de enero en la base 1 se estiman un total de 3.717547108968912 avis
os para ambulancias SBV en el turno de noche
```

In [347]:

```
pd.DataFrame({'Y_test': Y_test, 'Y_pred': Y_pred}).sample(5)
```

Out[347]:

							Y_test	Y_pred
AÑO	MES	DIA	DIA_CODIFICADO	BASE_CORREGIDA	TURNO_CODIFICADO	ABREV_CODIFICADO		
2018	9	30	6	8	1	0.0	7	4.314146
	11	9	4	1	1	0.0	4	4.580022
	6	6	2	5	0	4.0	2	4.374963
	3	23	4	11	2	1.0	4	3.489788
	10	28	6	8	0	1.0	3	4.705314

Paso nº6: Exportamos el modelo

In [348]:

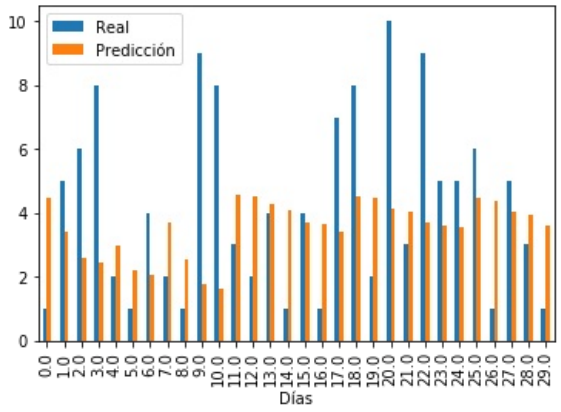
```
pickle.dump(model, open('RL-10.sav', 'wb'))
```

Paso nº7: Mostramos los datos reales y los datos predecidos para el primer mes de 2018

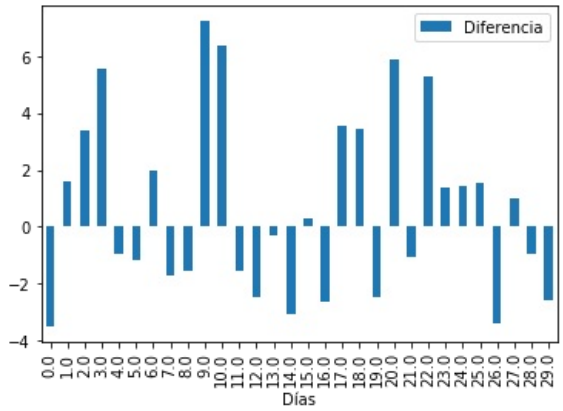
In [349]:

```
print_prediction(Y_test, Y_pred, 30)
```

Out[349]:



Out[349]:



Modelo nº 11: nº de Avisos para un mes, día , día de la semana, distrito, turno y dispositivo

Paso nº 1: Obtenemos variables

In [350]:

```
#Define X and Y
data = alerts_allyears[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DISTRITO", "TURNO_CODIFICADO", "ABREV_CODIFICADO"
, "SUCESO"]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DISTRITO", "TURNO_CODIFICADO", "ABREV_CODIFICADO"])
.count()
data = data.reset_index()
data = data[(np.abs(stats.zscore(data)) < 3).all(axis=1)] #Remove outliers
X = data[["MES", "DIA", "DIA_CODIFICADO", "DISTRITO", "TURNO_CODIFICADO", "ABREV_CODIFICADO"]]
Y = data["SUCESO"]

#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DISTRITO", "TURNO_CODIFICADO", "ABREV_CODIF
ICADO", "SUCESO"]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DISTRITO", "TURNO_CODIFICADO", "ABREV_CODIFIC
ADO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO", "DISTRITO", "TURNO_CODIFICADO", "ABREV_CODIFICADO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DISTRITO", "TURNO_CODIFICADO", "ABREV_CODIFIC
ADO", "SUCESO"]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DISTRITO", "TURNO_CODIFICADO", "ABREV_CODIFICAD
O"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO", "DISTRITO", "TURNO_CODIFICADO", "ABREV_CODIFICADO"]]
```

In [351]:

```
print("#" * 50)
print(X_train[:5])
print("#" * 50)
print(X_test[:5])
```

```
##################################################
   MES  DIA  DIA_CODIFICADO  DISTRITO  TURNO_CODIFICADO  ABREV_CODIFICADO
0    1    1               3       0.0                 1               0.0
1    1    1               3       0.0                 1               1.0
2    1    1               3       1.0                 0               1.0
3    1    1               3       1.0                 0               2.0
4    1    1               3       1.0                 1               1.0
##################################################
   MES  DIA  DIA_CODIFICADO  DISTRITO  TURNO_CODIFICADO  ABREV_CODIFICADO
0    1    1               0       1.0                 0               1.0
1    1    1               0       1.0                 0               4.0
2    1    1               0       1.0                 0              17.0
3    1    1               0       1.0                 0              28.0
4    1    1               0       1.0                 0              30.0
```

In [352]:

```
print(X_train.shape)
print(Y_train.shape)
```

```
(508437, 6)
(508437,)
```

Paso nº 2: Creamos modelo y predecimos mediante la ecuación normal

In [353]:

```
normal_thetas = normal_ecuation(X_train,Y_train)
normal_thetas
```

Out[353]:

```
array([ 3.46288816e+00,  4.13844926e-03, -1.46582506e-03,  7.18304248e-03,
       -5.15455400e-02, -2.97087765e-01, -6.66337709e-02])
```

Paso nº 3 : Creamos modelo y predecimos con la librería de sklearn

In [354]:

```
model = LinearRegression()
model.fit(X_train,Y_train)
Y_pred = model.predict(X_test)
```

Paso nº 4 : Calculamos RMSE y MAE del modelo para esos datos de entrenamiento (2009-2017 \ 2019) y test(2018)

In [355]:

```
info_model(Y_test,Y_pred)
```

```
Coefficients:  [ 0.00413845 -0.00146583  0.00718304 -0.05154554 -0.29708776 -0.06663377]
RMSE: 1.82
MAE: 1.3188307796230794
```

Paso nº 5 : Calculamos RMSE y MAE del modelo con validación cruzada

In [356]:

```
scoring = {"mae": "neg_mean_absolute_error", "rmse": rmse_cross_val}
estimator = model
scores = cross_validate(estimator, X,
                        Y, scoring=scoring,
                         cv=100, return_train_score=True)
```

In [357]:

```
pd.DataFrame(scores).mean()
```

Out[357]:

```
fit_time      0.057143
score_time    0.003164
test_mae     -1.261996
train_mae    -1.261815
test_rmse     1.672245
train_rmse    1.673436
dtype: float64
```

In [358]:

```
print("[MODELO ECUACIÓN NORMAL] Para un Lunes 1 de enero en el distrito 1 se estiman un total de {} avisos para ambulancias de tipo SBV en el turno de noche".format(normal_ecuation_predict(normal_thetas,[1,1,0,1,2,0])))
print("[MODELO SKLEARN] Para un Lunes 1 de enero en el distrito 1 se estiman un total de {} avisos ambulancias de tipo SBV en el turno de noche".format(model.predict(pd.DataFrame({'col1': [1], 'col2': 1, 'col3': [0], 'col4': [1], 'col5': [2], 'col6': [0]}))[0]))
```

```
[MODELO ECUACIÓN NORMAL] Para un Lunes 1 de enero en el distrito 1 se estiman un total de 2.81983971
88845714 avisos para ambulancias de tipo SBV en el turno de noche
[MODELO SKLEARN] Para un Lunes 1 de enero en el distrito 1 se estiman un total de 2.819839718884572
avisos ambulancias de tipo SBV en el turno de noche
```

In [359]:

```
pd.DataFrame({'Y_test': Y_test, 'Y_pred': Y_pred}).sample(5)
```

Out[359]:

							Y_test	Y_pred
AÑO	MES	DIA	DIA_CODIFICADO	DISTRITO	TURNO_CODIFICADO	ABREV_CODIFICADO		
2018	5	16	2	13.0	1	17.0	1	1.374539
	1	24	2	2.0	1	30.0	1	1.047021
	3	9	4	10.0	0	0.0	8	2.975388
		8	3	1.0	1	18.0	1	1.937085
	10	8	0	4.0	1	1.0	5	2.922642

Paso nº6: Exportamos el modelo

In [360]:

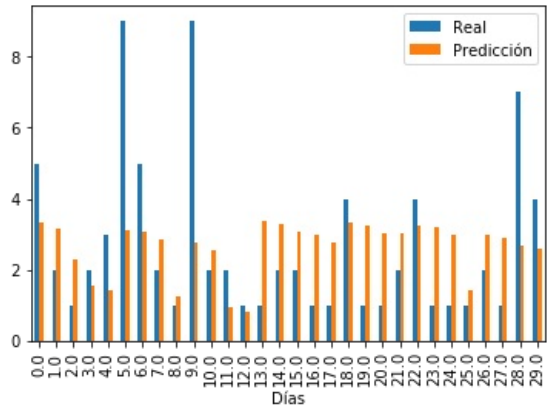
```
pickle.dump(model, open('RL-11.sav', 'wb'))
```

Paso nº7: Mostramos los datos reales y los datos predecidos para el primer mes de 2018

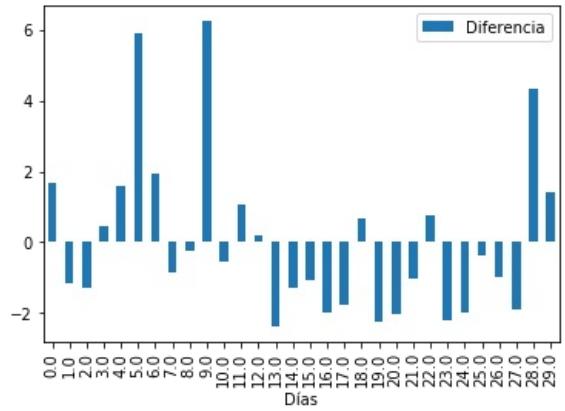
In [361]:

```
print_prediction(Y_test, Y_pred, 30)
```

Out[361]:



Out[361]:



Modelo nº 12: nº de Avisos para un mes, día , día de la semana, dispositivo y turno

Paso nº 1: Obtenemos variables

In [362]:

```
#Define X and Y
data = alerts_allyears[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO", "ABREV_CODIFICADO", "SUCESO"]]
.groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO", "ABREV_CODIFICADO"]).count()
data = data.reset_index()
data = data[(np.abs(stats.zscore(data)) < 3).all(axis=1)] #Remove outliers
X = data[["MES", "DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO", "ABREV_CODIFICADO"]]
Y = data["SUCESO"]

#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO", "ABREV_CODIFICADO", "SUC
ESO"]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO", "ABREV_CODIFICADO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO", "ABREV_CODIFICADO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO", "ABREV_CODIFICADO", "SUCES
O"]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO", "ABREV_CODIFICADO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO", "ABREV_CODIFICADO"]]
```

In [363]:

```
print("#" * 50)
print(X_train[:5])
print("#" * 50)
print(X_test[:5])
```

```
##################################################
   MES  DIA  DIA_CODIFICADO  TURNO_CODIFICADO  ABREV_CODIFICADO
0    1    1               3                 0               1.0
1    1    1               3                 0               2.0
2    1    1               3                 0              18.0
3    1    1               3                 0              20.0
4    1    1               3                 0              28.0
##################################################
   MES  DIA  DIA_CODIFICADO  TURNO_CODIFICADO  ABREV_CODIFICADO
0    1    1               0                 0               0.0
1    1    1               0                 0               1.0
2    1    1               0                 0               2.0
3    1    1               0                 0               4.0
4    1    1               0                 0              17.0
```

In [364]:

```
print(X_train.shape)
print(Y_train.shape)
```

```
(82649, 5)
(82649,)
```

Paso nº 2: Creamos modelo y predecimos mediante la ecuación normalº

In [365]:

```
normal_thetas = normal_ecuation(X_train,Y_train)
normal_thetas
```

Out[365]:

```
array([ 3.03347541e+01, -8.79798109e-03, -3.44460815e-02,  4.10171241e-01,
       -2.93161640e+00, -1.09294907e+00])
```

Paso nº 3 : Creamos modelo y predecimos con la librería de sklearn

In [366]:

```
model = LinearRegression()
model.fit(X_train,Y_train)
Y_pred = model.predict(X_test)
```

Paso nº 4 : Calculamos RMSE y MAE del modelo para esos datos de entrenamiento (2009-2017 \ 2019) y test(2018)

In [367]:

```
info_model(Y_test,Y_pred)
```

```
Coefficients:  [-0.00879798 -0.03444608  0.41017124 -2.9316164  -1.09294907]
RMSE: 17.76
MAE: 12.59147508115712
```

Paso nº 5 : Calculamos RMSE y MAE del modelo con validación cruzada

In [368]:

```
scoring = {"mae": "neg_mean_absolute_error", "rmse": rmse_cross_val}
estimator = model
scores = cross_validate(estimator, X,
                        Y, scoring=scoring,
                         cv=100, return_train_score=True)
```

In [369]:

```
pd.DataFrame(scores).mean()
```

Out[369]:

```
fit_time       0.007903
score_time     0.002070
test_mae     -11.858235
train_mae    -11.856230
test_rmse     16.316918
train_rmse    16.331172
dtype: float64
```

In [370]:

```
print("[MODELO ECUACIÓN NORMAL] Para un Lunes 1 de enero se estiman un total de {} avisos para ambulancias de tip
o SVB en el turno de noche".format(normal_ecuation_predict(normal_thetas,[1,1,0,2,0])))
print("[MODELO SKLEARN] Para un Lunes 1 de enero se estiman un total de {} avisos ambulancias de tipo SVB en el t
urno de noche".format(model.predict(pd.DataFrame({'col1': [1], 'col2': 1, 'col3': [0], 'col4': [2], 'col5': [0]})
)[0]))
```

```
[MODELO ECUACIÓN NORMAL] Para un Lunes 1 de enero se estiman un total de 24.428277213850652 avisos p
ara ambulancias de tipo SVB en el turno de noche
[MODELO SKLEARN] Para un Lunes 1 de enero se estiman un total de 24.428277213850073 avisos ambulanci
as de tipo SVB en el turno de noche
```

In [371]:

```
pd.DataFrame({'Y_test': Y_test, 'Y_pred': Y_pred}).sample(5)
```

Out[371]:

						Y_test	Y_pred
AÑO	**MES**	**DIA**	**DIA_CODIFICADO**	**TURNO_CODIFICADO**	**ABREV_CODIFICADO**		
2018	**6**	**10**	**6**	**2**	**2.0**	13	24.349402
	5	**29**	**1**	**0**	**0.0**	92	29.701999
		20	**6**	**1**	**30.0**	6	-3.657219
	4	**17**	**1**	**1**	**22.0**	3	3.147654
	12	**30**	**6**	**0**	**28.0**	2	1.054249

Paso nº6: Exportamos el modelo

In [372]:

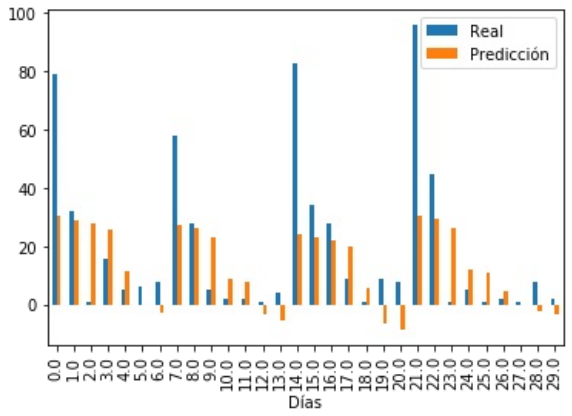
```
pickle.dump(model, open('RL-12.sav', 'wb'))
```

Paso nº7: Mostramos los datos reales y los datos predecidos para el primer mes de 2018

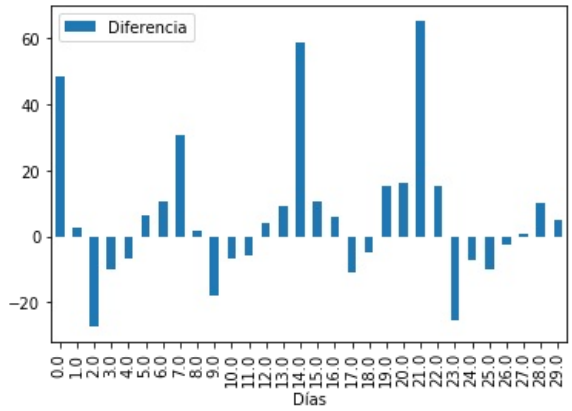
In [373]:

```
print_prediction(Y_test, Y_pred, 30)
```

Out[373]:



Out[373]:



Modelo nº 13: nº de Avisos para un mes, día , día de la semana, distrito, turno

In [9]:

```
#Define X and Y
data = alerts_allyears[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DISTRITO", "TURNO_CODIFICADO", "SUCESO"]].groupby
(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DISTRITO", "TURNO_CODIFICADO"]).count()
data = data.reset_index()
data = data[(np.abs(stats.zscore(data)) < 3).all(axis=1)] #Remove outliers
X = data[["MES", "DIA", "DIA_CODIFICADO", "DISTRITO", "TURNO_CODIFICADO"]]
Y = data["SUCESO"]

#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DISTRITO", "TURNO_CODIFICADO", "SUCESO"]].g
roupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DISTRITO", "TURNO_CODIFICADO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO", "DISTRITO", "TURNO_CODIFICADO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DISTRITO", "TURNO_CODIFICADO", "SUCESO"]].gro
upby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DISTRITO", "TURNO_CODIFICADO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO", "DISTRITO", "TURNO_CODIFICADO"]]
```

In [10]:

```
print("#" * 50)
print(X_train[:5])
print("#" * 50)
print(X_test[:5])
```

```
##################################################
   MES  DIA  DIA_CODIFICADO  DISTRITO  TURNO_CODIFICADO
0    1    1               3       0.0                 1
1    1    1               3       2.0                 0
2    1    1               3       2.0                 1
3    1    1               3       2.0                 2
4    1    1               3       3.0                 0
##################################################
   MES  DIA  DIA_CODIFICADO  DISTRITO  TURNO_CODIFICADO
0    1    1               0       1.0                 1
1    1    1               0       2.0                 0
2    1    1               0       2.0                 1
3    1    1               0       2.0                 2
4    1    1               0       3.0                 0
```

In [11]:

```
print(X_train.shape)
print(Y_train.shape)
```

```
(202443, 5)
(202443,)
```

Paso nº 2: Creamos modelo y predecimos mediante la ecuación normal

In [12]:

```
normal_thetas = normal_ecuation(X_train,Y_train)
normal_thetas
```

Out[12]:

```
array([ 9.96382364e+00,  1.30984980e-02, -4.91118668e-04,  6.76978813e-02,
       -2.55186601e-01, -1.50571637e+00])
```

Paso nº 3 : Creamos modelo y predecimos con la librería de sklearn

In [13]:

```
model = LinearRegression()
model.fit(X_train,Y_train)
Y_pred = model.predict(X_test)
```

Paso nº 4 : Calculamos RMSE y MAE del modelo para esos datos de entrenamiento (2009-2017 \ 2019) y test(2018)

In [14]:

```
info_model(Y_test,Y_pred)
```

```
Coefficients:  [ 1.30984980e-02 -4.91118668e-04  6.76978813e-02 -2.55186601e-01
 -1.50571637e+00]
RMSE: 4.17
MAE: 3.1713426636777755
```

Paso nº 5 : Calculamos RMSE y MAE del modelo con validación cruzada

In [426]:

```
scoring = {"mae": "neg_mean_absolute_error", "rmse": rmse_cross_val}
estimator = model
scores = cross_validate(estimator, X,
                        Y, scoring=scoring,
                         cv=100, return_train_score=True)
```

In [427]:

```
pd.DataFrame(scores).mean()
```

Out[427]:

```
fit_time      0.015635
score_time    0.001737
test_mae     -3.118035
train_mae    -3.117057
test_rmse     4.038131
train_rmse    4.039752
dtype: float64
```

In [428]:

```
print("[MODELO ECUACIÓN NORMAL] Para un Lunes 1 de enero se estiman un total de {} avisos para el distrito 1 en e
l turno de noche".format(normal_ecuation_predict(normal_thetas,[1,1,0,1,2])))
print("[MODELO SKLEARN] Para un Lunes 1 de enero se estiman un total de {} avisos para el distrito 1 en el turno
de noche".format(model.predict(pd.DataFrame({'col1': [1], 'col2': 1, 'col3': [0], 'col4': [1], 'col5':[2]}))[0]))
```

```
[MODELO ECUACIÓN NORMAL] Para un Lunes 1 de enero se estiman un total de 6.710419032687319 avisos pa
ra el distrito 1 en el turno de noche
[MODELO SKLEARN] Para un Lunes 1 de enero se estiman un total de 6.710419032686967 avisos para el di
strito 1 en el turno de noche
```

In [429]:

```
pd.DataFrame({'Y_test': Y_test, 'Y_pred': Y_pred}).sample(5)
```

Out[429]:

						Y_test	Y_pred
AÑO	MES	DIA	DIA_CODIFICADO	DISTRITO	TURNO_CODIFICADO		
2018	8	5	6	9.0	2	9	5.154615
	6	11	0	3.0	1	8	7.755868
	9	16	6	16.0	0	2	6.400241
		7	4	15.0	2	2	3.505077
	1	28	6	2.0	2	4	6.761650

Paso nº6: Exportamos el modelo

In [431]:

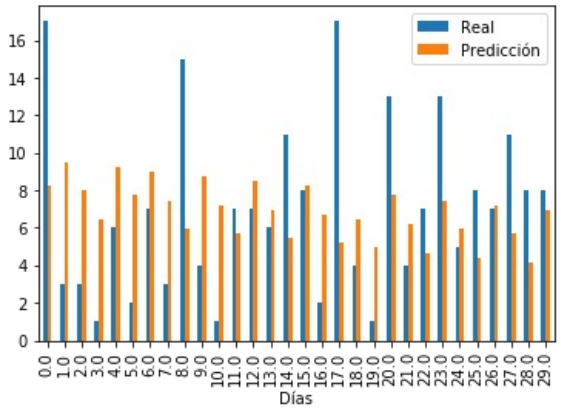
```
pickle.dump(model, open('RL-13.sav', 'wb'))
```

Paso nº7: Mostramos los datos reales y los datos predecidos para el primer mes de 2018

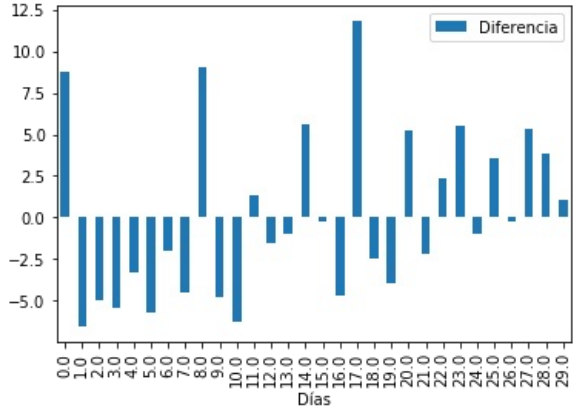
In [432]:

```
print_prediction(Y_test, Y_pred, 30)
```

Out[432]:



Out[432]:



Redes Neuronales Fase I SAMUR - Protección Civil

Propósito del documento

El objetivo de este documento es la creación de modelos utilizando como técnica de aprendizaje Redes neuronales, para predecir el número de avisos o activaciones que recibirá SAMUR-PC para un determinado día y unas determinadas cirucnstancias en la ciudad de Madrid.

Este documento contiene el código desarrollado para la primera Fase de Redes Neuronales, es decir, el desarrollo se realiza de forma en local utilizando los equipos de los estudiantes, por ello, en esta fase, se crean redes neuronales de cuatroo capas (capa de entrada, dos capas ocultas y capa de salida.

Esta primera fase esta formada por dos documentos:

- Redes Neuronales Fase I
- Redes Neuronales Fase I (II)

Para cada uno de los modelos se desarrollarán los siguientes pasos:

1. Obtención de variables
2. Transformación de datos para prerparalos para modelos Deep Learning utilizando la librería Keras
3. Búsqueda del mejor modelo
4. Creación del mejor modelo
5. Búsqueda del mejor modelo
6. Exportación del modelo

Librerías

In [1]:

```
import pandas as pd
from matplotlib import pyplot as plt
import numpy as np
from math import sqrt
from scipy import stats
import pickle
import math
import warnings
warnings.filterwarnings("ignore")
```

Semilla para generar modelos reproducibles

In [2]:

```
np.random.seed(2)
```

Librerias Deep Learning

In [3]:

```
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn import metrics
from keras.layers import Flatten
from keras.layers import LSTM
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
tf.random.set_seed(2)
```

```
Using TensorFlow backend.
```

Obtenemos los datos

In [4]:

```
alerts_2009 = pd.read_excel("data/samur2009.xlsx")
alerts_2010 = pd.read_excel("data/samur2010.xlsx")
alerts_2011 = pd.read_excel("data/samur2011.xlsx")
alerts_2012 = pd.read_excel("data/samur2012.xlsx")
alerts_2013 = pd.read_excel("data/samur2013.xlsx")
alerts_2014 = pd.read_excel("data/samur2014.xlsx")
alerts_2015 = pd.read_excel("data/samur2015.xlsx")
alerts_2016 = pd.read_excel("data/samur2016.xlsx")
alerts_2017 = pd.read_excel("data/samur2017.xlsx")
alerts_2018 = pd.read_excel("data/samur2018.xlsx")
alerts_2019 = pd.read_excel("data/samur2019.xlsx")
alerts_allyears = pd.concat([alerts_2009,alerts_2010, alerts_2011, alerts_2012, alerts_2013, alerts_2014,
                             alerts_2015, alerts_2016, alerts_2017, alerts_2018, alerts_2019], ignore_index=True)
```

In [5]:

```
train_dataframe = pd.concat([alerts_2009,alerts_2010, alerts_2011, alerts_2012, alerts_2013, alerts_2014,
                             alerts_2015, alerts_2016, alerts_2017, alerts_2019], ignore_index=True)
test_dataframe = alerts_2018
```

In [6]:

```
def print_prediction(real, pred, dias):
    """ Prints two plots: 1:a bar for real data and a bar for predicted data 2:the difference between real and pr
edicted data
    """
    #Print real and predicted data
    y = np.zeros((real.shape[0],3))
    y[:,0]= np.arange(real.shape[0])
    y[:,1] = real
    y[:,2] = pred
    df = pd.DataFrame(y, columns=["Días", "Real", "Predicción"]).head(dias)
    ax = df.plot(x="Días", y=["Real", "Predicción"], kind="bar")
    plt.show()


    #Print the difference
    y = np.zeros((real.shape[0],2))
    y[:,0] = np.arange(real.shape[0])
    y[:,1] = real - pred
    df = pd.DataFrame(y, columns=["Días", "Diferencia"]).head(dias)
    ax = df.plot(x="Días", y="Diferencia", kind="bar")
    plt.show()
```

In [7]:

```
def lstm_model_two_layers(i,j,num_features):
    """Create Neural Network with with two lstm layers with sizes i, j"""
    model = Sequential()
    model.add(LSTM(int(i), activation='relu',return_sequences=True, input_shape=(num_features,1)))
    model.add(LSTM(int(j), activation='relu'))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mse')
    return model
```

In [1]:

```
def draw_layer_size(X_train,X_test,Y_test,num_features,size=100):
    """Function which generes models and get the best rmse of all models generated"""
    rmse_values = {}
    min_rmse = math.inf
    sol="La mejor red tiene dos capas de {} y {} nodos".format(0,0)
    dots = np.linspace(50,size,size/50)
    for i in np.nditer(dots):
        for j in np.nditer(dots):
            key = "{}-{}".format(i,j)
            if key not in rmse_values.keys():
                model = lstm_model_two_layers(i,j,num_features)
                model.fit(X_train, Y_train, epochs=150, verbose=0)
                Y_pred = model.predict(X_test, verbose=0)[:,0]
                new_rmse = np.sqrt(metrics.mean_squared_error(Y_test, Y_pred))
                print("i ={} , j={} --> RMSE:{}".format(i, j, new_rmse))
                rmse_values[key]=new_rmse
                if min_rmse > new_rmse:
                    min_rmse = new_rmse
                    sol = "La mejor red tiene dos capas de {} y {} nodos".format(i,j)
    print(sol + "---> RMSE:{}".format(min_rmse))
```

Modelo nº 1: Nº de avisos en función del Mes y el día

Paso nº 1: Obtenemos variables

In [10]:

```
#Define X and Y
data = alerts_allyears[["AÑO", "MES", "DIA", "SUCESO"]].groupby(["AÑO", "MES", "DIA"]).count()
data = data.reset_index()
data = data[(np.abs(stats.zscore(data)) < 3).all(axis=1)] #Remove outliers
X = data[["MES","DIA"]]
Y = data["SUCESO"]

#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "SUCESO"]].groupby(["AÑO", "MES", "DIA"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Remove outliers
data_train = data_train.reset_index()
Y_train = data_train["SUCESO"]
X_train = data_train[["MES","DIA"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "SUCESO"]].groupby(["AÑO", "MES", "DIA"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Remove outliers
data_test = data_test.reset_index()
Y_test = data_test["SUCESO"]
X_test = data_test[["MES","DIA"]]
```

In [10]:

```
data_train.head(5)
```

Out[10]:

	AÑO	MES	DIA	SUCESO
0	2009	1	1	522
1	2009	1	2	329
2	2009	1	3	322
3	2009	1	4	314
4	2009	1	5	355

In [11]:

```
print("#" * 50)
print(X_train[:5])
print("#" * 50)
print(X_test[:5])
```

```
##################################################
   MES  DIA
0    1    1
1    1    2
2    1    3
3    1    4
4    1    5
##################################################
   MES  DIA
0    1    1
1    1    2
2    1    3
3    1    4
4    1    5
```

In [12]:

```
print(X_train.shape)
print(Y_train.shape)
```

```
(3580, 2)
(3580,)
```

Parte nº 2: Trasformamos los datos a array de numpy para poder utilziarlos en Keras

In [11]:

```
X_train = X_train.to_numpy()
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.to_numpy()
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

Parte nº 3: Buscamos el mejor modelo

In [14]:

```
draw_layer_size(X_train,X_test,Y_test,2,300)
```

```
WARNING:tensorflow:From C:\Users\manu_\Anaconda3\lib\site-packages\keras\backend\tensorflow_backend.
py:422: The name tf.global_variables is deprecated. Please use tf.compat.v1.global_variables instead
.

i =50.0 , j=50.0 --> RMSE:58.373286821128346
i =50.0 , j=100.0 --> RMSE:58.905546633480036
i =50.0 , j=150.0 --> RMSE:59.96399488641204
i =50.0 , j=200.0 --> RMSE:59.271781574493396
i =50.0 , j=250.0 --> RMSE:62.33451312217199
i =50.0 , j=300.0 --> RMSE:62.69009309129902
i =100.0 , j=50.0 --> RMSE:62.27830479752886
i =100.0 , j=100.0 --> RMSE:66.20475533059428
i =100.0 , j=150.0 --> RMSE:59.703170340574424
i =100.0 , j=200.0 --> RMSE:59.72894995338572
i =100.0 , j=250.0 --> RMSE:57.073494304689916
i =100.0 , j=300.0 --> RMSE:68.72145592867986
i =150.0 , j=50.0 --> RMSE:55.92384366185265
i =150.0 , j=100.0 --> RMSE:64.68923662199414
i =150.0 , j=150.0 --> RMSE:61.39060473383055
i =150.0 , j=200.0 --> RMSE:59.39151312192766
i =150.0 , j=250.0 --> RMSE:54.42245467597345
i =150.0 , j=300.0 --> RMSE:58.78832977441892
i =200.0 , j=50.0 --> RMSE:51.58902077807659
i =200.0 , j=100.0 --> RMSE:53.69138769206057
i =200.0 , j=150.0 --> RMSE:54.61362721723543
i =200.0 , j=200.0 --> RMSE:58.424499347104984
i =200.0 , j=250.0 --> RMSE:62.951566072245
i =200.0 , j=300.0 --> RMSE:65.4604869105284
i =250.0 , j=50.0 --> RMSE:53.19968277792412
i =250.0 , j=100.0 --> RMSE:54.04331858644949
i =250.0 , j=150.0 --> RMSE:66.84931885150844
i =250.0 , j=200.0 --> RMSE:58.17398626210035
i =250.0 , j=250.0 --> RMSE:61.24640604180267
i =250.0 , j=300.0 --> RMSE:59.11161486252686
i =300.0 , j=50.0 --> RMSE:59.03741987813885
i =300.0 , j=100.0 --> RMSE:61.790251320288746
i =300.0 , j=150.0 --> RMSE:52.674077848545124
i =300.0 , j=200.0 --> RMSE:52.19881661954344
i =300.0 , j=250.0 --> RMSE:61.46424138672339
i =300.0 , j=300.0 --> RMSE:64.71407611252295
La mejor red tiene dos capas de 200.0 y 50.0 nodos---> RMSE:51.58902077807659
```

Nos quedamos con la red que minimza el RMSE y creamos el modelo

Parte nº4: Creamos el modelo

In [16]:

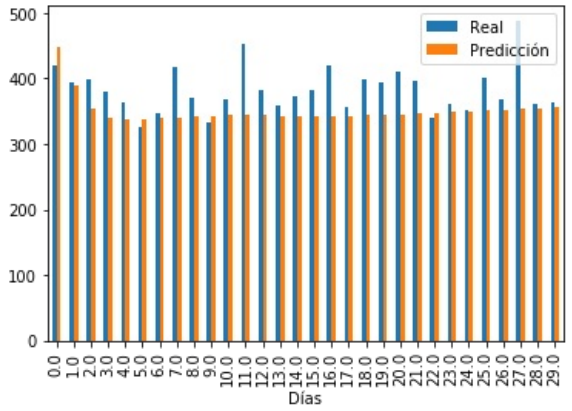
```
model = lstm_model_two_layers(200,50,num_features=2)
```

Parte nº5: Entrenamos y predecimos los valores de test

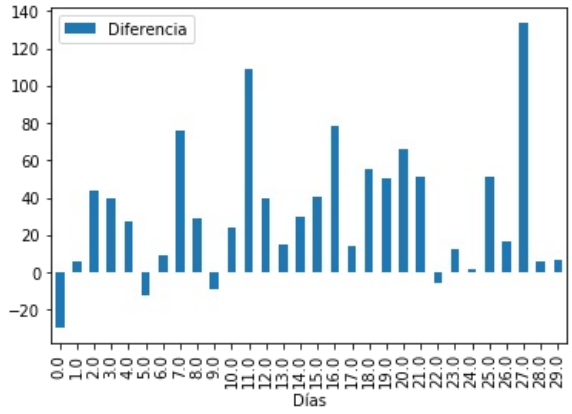
In [17]:

```
model.fit(X_train, Y_train, epochs=150, verbose=0)
Y_pred = model.predict(X_test, verbose=0)[:,0]
print_prediction(Y_test, Y_pred, 30)
```

Out[17]:



Out[17]:



Parte nº6: Guardamos el modelo creado

In [21]:

```
pickle.dump(model, open('NN/NN-1.sav', 'wb'))
```

Modelo nº 2: Nº de avisos para un Mes, un día, un día de la semana

Paso nº 1: Obtenemos variables

In [8]:

```
#Define X and Y
data = alerts_allyears[["AÑO", "MES", "DIA_CODIFICADO", "DIA", "SUCESO"]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO"]).count()
data = data.reset_index()
data = data[(np.abs(stats.zscore(data)) < 3).all(axis=1)] #Remove outliers
X = data[["MES", "DIA", "DIA_CODIFICADO"]]
Y = data["SUCESO"]

#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA_CODIFICADO", "DIA", "SUCESO"]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA_CODIFICADO", "DIA", "SUCESO"]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO"]]
```

In [9]:

```
data_train.head(5)
```

Out[9]:

	AÑO	MES	DIA	DIA_CODIFICADO	SUCESO
0	2009	1	1	3	522
1	2009	1	2	4	329
2	2009	1	3	5	322
3	2009	1	4	6	314
4	2009	1	5	0	355

In [10]:

```
print("#" * 50)
print(X_train[:5])
print("#" * 50)
print(X_test[:5])
```

```
##################################################
   MES  DIA  DIA_CODIFICADO
0    1    1               3
1    1    2               4
2    1    3               5
3    1    4               6
4    1    5               0
##################################################
   MES  DIA  DIA_CODIFICADO
0    1    1               0
1    1    2               1
2    1    3               2
3    1    4               3
4    1    5               4
```

In [11]:

```
print(X_train.shape)
print(Y_train.shape)
```

```
(3586, 3)
(3586,)
```

Parte nº 2: Trasformamos los datos a array de numpy para poder utilziarlos en Keras

In [12]:

```
X_train = X_train.to_numpy()
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.to_numpy()
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

Parte nº 3: Buscamos el mejor modelo

In [0]:

```
draw_layer_size(X_train,X_test,Y_test,num_features=3,size=300)
```

```
WARNING:tensorflow:From C:\Users\manu_\Anaconda3\lib\site-packages\keras\backend\tensorflow_backend.
py:422: The name tf.global_variables is deprecated. Please use tf.compat.v1.global_variables instead
.

i =50.0 , j=50.0 --> RMSE:57.66528116701713
i =50.0 , j=100.0 --> RMSE:51.9713112586715
i =50.0 , j=150.0 --> RMSE:49.2608830448229
i =50.0 , j=200.0 --> RMSE:45.96725774476892
i =50.0 , j=250.0 --> RMSE:52.20653000266393
i =50.0 , j=300.0 --> RMSE:65.20746692938549
i =100.0 , j=50.0 --> RMSE:45.88559010455626
i =100.0 , j=100.0 --> RMSE:54.24658150207637
i =100.0 , j=150.0 --> RMSE:49.232801671035375
i =100.0 , j=200.0 --> RMSE:57.10909543724828
i =100.0 , j=250.0 --> RMSE:56.32454223024714
i =100.0 , j=300.0 --> RMSE:43.95057384569682
i =150.0 , j=50.0 --> RMSE:52.05912299739321
i =150.0 , j=100.0 --> RMSE:49.22628825858631
i =150.0 , j=150.0 --> RMSE:47.46046881548932
i =150.0 , j=200.0 --> RMSE:55.62092868212133
i =150.0 , j=250.0 --> RMSE:57.5293265976064
i =150.0 , j=300.0 --> RMSE:56.85268171875784
i =200.0 , j=50.0 --> RMSE:49.554974851724296
i =200.0 , j=100.0 --> RMSE:68.89457771425762
i =200.0 , j=150.0 --> RMSE:49.146009864427796
i =200.0 , j=200.0 --> RMSE:46.66962825380208
i =200.0 , j=250.0 --> RMSE:47.19708127211655
i =200.0 , j=300.0 --> RMSE:53.02090503501426
i =250.0 , j=50.0 --> RMSE:59.07118997940907
```

Nos quedamos con la red que minimza el RMSE y creamos el modelo

Parte nº4: Creamos el modelo

In [18]:

```
model = lstm_model_two_layers(100, 300, num_features=3)
```

Parte nº5: Entrenamos y predecimos los valores de test

In [19]:

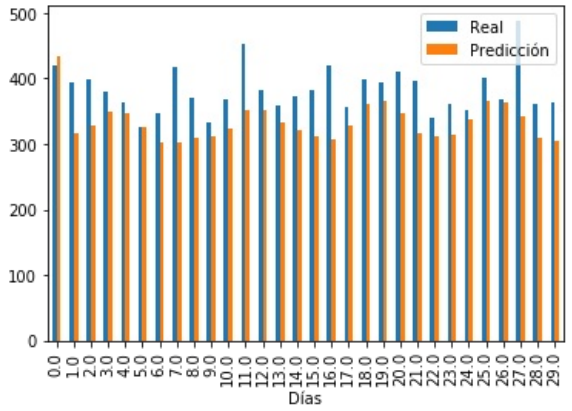
```
model.fit(X_train, Y_train, epochs=150, verbose=0)
Y_pred = model.predict(X_test, verbose=0)[:,0]
```

Parte nº5 : Obtenemos RMSE

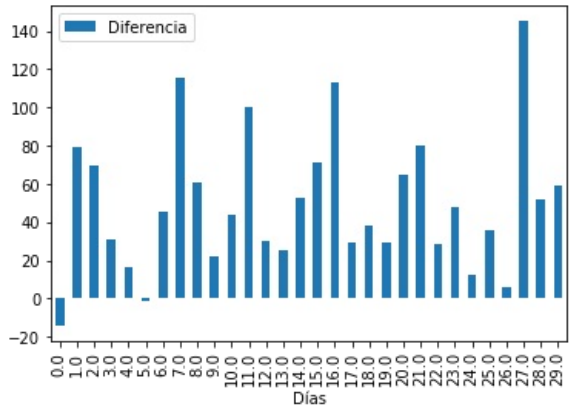
In [21]:

```
model.fit(X_train, Y_train, epochs=150, verbose=0)
Y_pred = model.predict(X_test, verbose=0)[:,0]
print_prediction(Y_test, Y_pred, 30)
```

Out[21]:



Out[21]:



Parte nº6: Guardamos el modelo creado

In [22]:

```
pickle.dump(model, open('NN/NN-2.sav', 'wb'))
```

Modelo nº 3: Nº de avisos para un Mes, día, dia de la semana y base

Paso nº 1: Obtenemos variables

In [26]:

```
#Define X and Y
data = alerts_allyears[["AÑO", "MES", "DIA", "DIA_CODIFICADO","BASE_CORREGIDA", "SUCESO"]].groupby(["AÑO", "MES",
"DIA", "DIA_CODIFICADO","BASE_CORREGIDA"]).count()
data = data.reset_index()
data = data[(np.abs(stats.zscore(data)) < 3).all(axis=1)] #Remove outliers
X = data[["MES", "DIA", "DIA_CODIFICADO","BASE_CORREGIDA"]]
Y = data["SUCESO"]

#Define X and Y for training

data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO","BASE_CORREGIDA", "SUCESO"]].groupby(["AÑO",
"MES", "DIA", "DIA_CODIFICADO","BASE_CORREGIDA"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO","BASE_CORREGIDA"]]

data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO","BASE_CORREGIDA", "SUCESO"]].groupby(["AÑO", "M
ES", "DIA", "DIA_CODIFICADO","BASE_CORREGIDA"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO","BASE_CORREGIDA"]]
```

In [27]:

```
data_train.head(5)
```

Out[27]:

	AÑO	MES	DIA	DIA_CODIFICADO	BASE_CORREGIDA	SUCESO
0	2009	1	1	3	-1	28
1	2009	1	1	3	3	31
2	2009	1	1	3	4	18
3	2009	1	1	3	5	22
4	2009	1	1	3	6	35

In [28]:

```
print("#" * 50)
print(X_train[:5])
print("#" * 50)
print(X_test[:5])
```

```
##################################################
   MES  DIA  DIA_CODIFICADO  BASE_CORREGIDA
0    1    1               3              -1
1    1    1               3               3
2    1    1               3               4
3    1    1               3               5
4    1    1               3               6
##################################################
   MES  DIA  DIA_CODIFICADO  BASE_CORREGIDA
0    1    1               0               1
1    1    1               0               2
2    1    1               0               5
3    1    1               0               7
4    1    1               0               8
```

In [29]:

```
print(X_train.shape)
print(Y_train.shape)
```

```
(71074, 4)
(71074,)
```

Parte nº 2: Trasformamos los datos a array de numpy para poder utilziarlos en Keras

In [30]:

```
X_train = X_train.to_numpy()
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.to_numpy()
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

Parte nº 3: Buscamos el mejor modelo

In [37]:

```
draw_layer_size(X_train,X_test,Y_test,num_features=4,size=200)
```

```
i =50.0 , j=50.0 --> RMSE:8.644803654452534
i =50.0 , j=100.0 --> RMSE:8.74031823332328
i =50.0 , j=150.0 --> RMSE:8.637190612595127
i =50.0 , j=200.0 --> RMSE:8.758220009692815
i =100.0 , j=50.0 --> RMSE:8.563265684581305
i =100.0 , j=100.0 --> RMSE:8.757619908793322
i =100.0 , j=150.0 --> RMSE:8.862222989108943
i =100.0 , j=200.0 --> RMSE:8.792732302683733
i =150.0 , j=50.0 --> RMSE:8.404567842976636
i =150.0 , j=100.0 --> RMSE:8.684166113795058
i =150.0 , j=150.0 --> RMSE:8.765968550308495
i =150.0 , j=200.0 --> RMSE:8.596254826486339
i =200.0 , j=50.0 --> RMSE:8.746015272487424
i =200.0 , j=100.0 --> RMSE:8.575817949655514
i =200.0 , j=150.0 --> RMSE:8.740361674550856
i =200.0 , j=200.0 --> RMSE:8.663356999200522
La mejor red tiene dos capas de 150.0 y 50.0 nodos---> RMSE:8.404567842976636
```

Parte nº4: Creamos el modelo

In [38]:

```
model = lstm_model_two_layers(150,50, num_features=4)
```

Parte nº5: Entrenamos y predecimos los valores de test

In [0]:

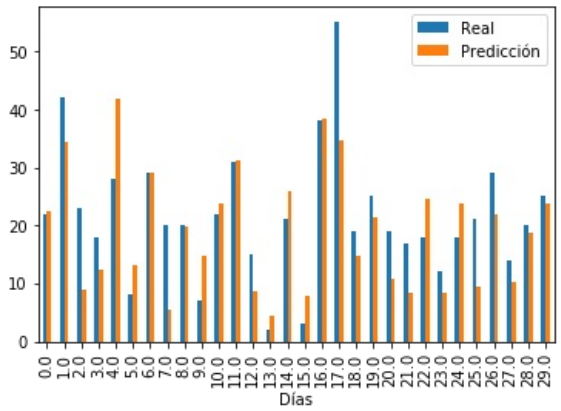
```
model.fit(X_train, Y_train, epochs=150, verbose=0)
Y_pred = model.predict(X_test, verbose=0)[:,0]
```

Parte nº5 : Obtenemos RMSE

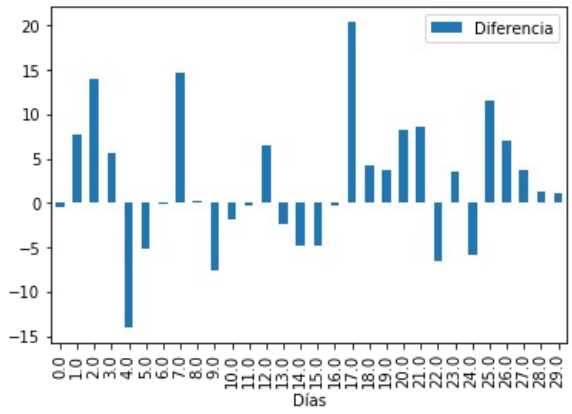
In [40]:

```
model.fit(X_train, Y_train, epochs=150, verbose=0)
Y_pred = model.predict(X_test, verbose=0)[:,0]
print_prediction(Y_test, Y_pred, 30)
```

Out[40]:



Out[40]:



Parte nº6: Guardamos el modelo creado

In [41]:

```
pickle.dump(model, open('NN/NN-3.sav', 'wb'))
```

Modelo nº 4: Nº de avisos para un Mes, día, día de la semana y distrito

Paso nº 1: Obtenemos variables

In [8]:

```
#Define X and Y
data = alerts_allyears[["AÑO", "MES", "DIA", "DIA_CODIFICADO","DISTRITO", "SUCESO"]].groupby(["AÑO", "MES", "DIA"
, "DIA_CODIFICADO","DISTRITO"]).count()
data = data.reset_index()
data = data[(np.abs(stats.zscore(data)) < 3).all(axis=1)] #Remove outliers
X = data[["MES", "DIA", "DIA_CODIFICADO","DISTRITO"]]
Y = data["SUCESO"]

#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO","DISTRITO", "SUCESO"]].groupby(["AÑO", "MES",
"DIA", "DIA_CODIFICADO","DISTRITO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO","DISTRITO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO","DISTRITO", "SUCESO"]].groupby(["AÑO", "MES", "
DIA", "DIA_CODIFICADO","DISTRITO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO","DISTRITO"]]
```

In [9]:

```
data_train.head(5)
```

Out[9]:

	AÑO	MES	DIA	DIA_CODIFICADO	DISTRITO	SUCESO
0	2009	1	1	3	0.0	3
1	2009	1	1	3	2.0	17
2	2009	1	1	3	3.0	9
3	2009	1	1	3	4.0	24
4	2009	1	1	3	5.0	26

In [10]:

```
print("#" * 50)
print(X_train[:5])
print("#" * 50)
print(X_test[:5])
```

```
##################################################
   MES  DIA  DIA_CODIFICADO  DISTRITO
0    1    1               3       0.0
1    1    1               3       2.0
2    1    1               3       3.0
3    1    1               3       4.0
4    1    1               3       5.0
##################################################
   MES  DIA  DIA_CODIFICADO  DISTRITO
0    1    1               0       2.0
1    1    1               0       3.0
2    1    1               0       4.0
3    1    1               0       5.0
4    1    1               0       6.0
```

In [11]:

```
print(X_train.shape)
print(Y_train.shape)
```

```
(75588, 4)
(75588,)
```

Parte nº 2: Trasformamos los datos a array de numpy para poder utilziarlos en Keras

In [12]:

```
X_train = X_train.to_numpy()
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.to_numpy()
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

Parte nº 3: Buscamos el mejor modelo

In [13]:

```
draw_layer_size(X_train,X_test,Y_test,num_features=4,size=200)
```

```
i =50.0 , j=50.0 --> RMSE:6.432890565320542
i =50.0 , j=100.0 --> RMSE:6.267812111273419
i =50.0 , j=150.0 --> RMSE:6.328296649030823
i =50.0 , j=200.0 --> RMSE:6.4617795501511885
i =100.0 , j=50.0 --> RMSE:6.4960740891013256
i =100.0 , j=100.0 --> RMSE:6.363154814410237
i =100.0 , j=150.0 --> RMSE:6.456242669748052
i =100.0 , j=200.0 --> RMSE:6.393154743502923
i =150.0 , j=50.0 --> RMSE:6.261808826355144
i =150.0 , j=100.0 --> RMSE:6.781849622491193
i =150.0 , j=150.0 --> RMSE:6.43283139568369
i =150.0 , j=200.0 --> RMSE:6.420736481330873
i =200.0 , j=50.0 --> RMSE:6.281338099723022
i =200.0 , j=100.0 --> RMSE:6.2673772245034645
i =200.0 , j=150.0 --> RMSE:6.367352593812655
i =200.0 , j=200.0 --> RMSE:6.515748242714253
La mejor red tiene dos capas de 150.0 y 50.0 nodos---> RMSE:6.261808826355144
```

Parte nº4: Creamos el modelo

In [15]:

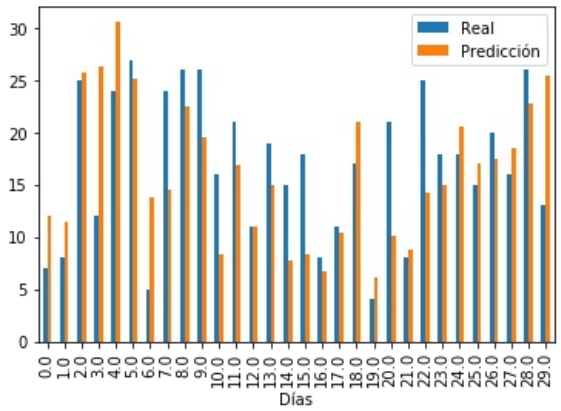
```
model = lstm_model_two_layers(150, 50, num_features=4)
```

Parte nº5: Entrenamos y predecimos los valores de test

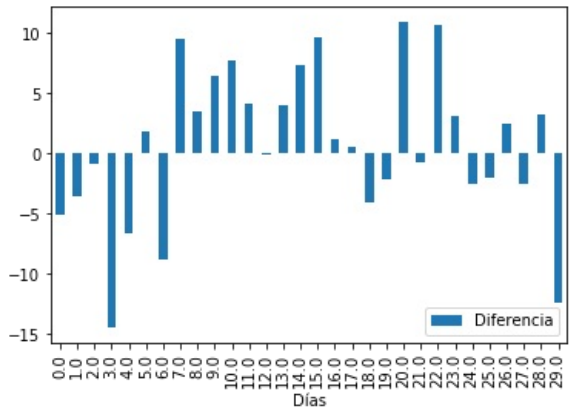
In [18]:

```
model.fit(X_train, Y_train, epochs=150, verbose=0)
Y_pred = model.predict(X_test, verbose=0)[:,0]
print_prediction(Y_test, Y_pred, 30)
```

Out[18]:



Out[18]:



Parte nº6: Guardamos el modelo creado

In [17]:

```
pickle.dump(model, open('NN/NN-4.sav', 'wb'))
```

Modelo nº 5 : Nº de avisos para un Mes, día, día de la semana y turno

Paso nº 1: Obtenemos variables

In [19]:

```
#Define X and Y
data = alerts_allyears[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO", "SUCESO"]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO"]).count()
data = data.reset_index()
data = data[(np.abs(stats.zscore(data)) < 3).all(axis=1)] #Remove outliers
X = data[["MES", "DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO"]]
Y = data["SUCESO"]

#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO", "SUCESO"]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO", "SUCESO"]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO"]]
```

In [20]:

```
data_train.head(5)
```

Out[20]:

	AÑO	MES	DIA	DIA_CODIFICADO	TURNO_CODIFICADO	SUCESO
0	2009	1	1	3	0	190
1	2009	1	1	3	1	116
2	2009	1	1	3	2	216
3	2009	1	2	4	0	147
4	2009	1	2	4	1	135

In [21]:

```
print("#" * 50)
print(X_train[:5])
print("#" * 50)
print(X_test[:5])
```

```
##################################################
   MES  DIA  DIA_CODIFICADO  TURNO_CODIFICADO
0    1    1               3                 0
1    1    1               3                 1
2    1    1               3                 2
3    1    2               4                 0
4    1    2               4                 1
##################################################
   MES  DIA  DIA_CODIFICADO  TURNO_CODIFICADO
0    1    1               0                 0
1    1    1               0                 1
2    1    1               0                 2
3    1    2               1                 0
4    1    2               1                 1
```

In [22]:

```
print(X_train.shape)
print(Y_train.shape)
```

```
(10988, 4)
(10988,)
```

Parte nº 2: Trasformamos los datos a array de numpy para poder utilziarlos en Keras

In [23]:

```
X_train = X_train.to_numpy()
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.to_numpy()
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

Parte nº 3: Buscamos el mejor modeloº

In [24]:

```
draw_layer_size(X_train,X_test,Y_test,num_features=4,size=200)
```

```
i =50.0 , j=50.0 --> RMSE:27.49552858574688
i =50.0 , j=100.0 --> RMSE:29.853787623427895
i =50.0 , j=150.0 --> RMSE:27.41785882284375
i =50.0 , j=200.0 --> RMSE:28.226532978525054
i =100.0 , j=50.0 --> RMSE:28.2563707793226
i =100.0 , j=100.0 --> RMSE:28.07883082503782
i =100.0 , j=150.0 --> RMSE:26.951099848724617
i =100.0 , j=200.0 --> RMSE:30.047048484800047
i =150.0 , j=50.0 --> RMSE:28.63979216453456
i =150.0 , j=100.0 --> RMSE:27.96004215288821
i =150.0 , j=150.0 --> RMSE:27.810044335785864
i =150.0 , j=200.0 --> RMSE:27.26512599411937
i =200.0 , j=50.0 --> RMSE:26.411171205897965
i =200.0 , j=100.0 --> RMSE:27.233628554857773
i =200.0 , j=150.0 --> RMSE:28.576163940178983
i =200.0 , j=200.0 --> RMSE:28.587666578843738
La mejor red tiene dos capas de 200.0 y 50.0 nodos---> RMSE:26.411171205897965
```

Parte nº4: Creamos el modelo

In [26]:

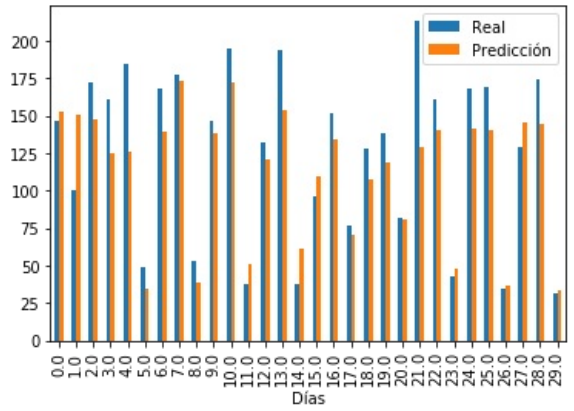
```
model = lstm_model_two_layers(200, 50, num_features=4)
```

Parte nº5: Entrenamos y predecimos los valores de test

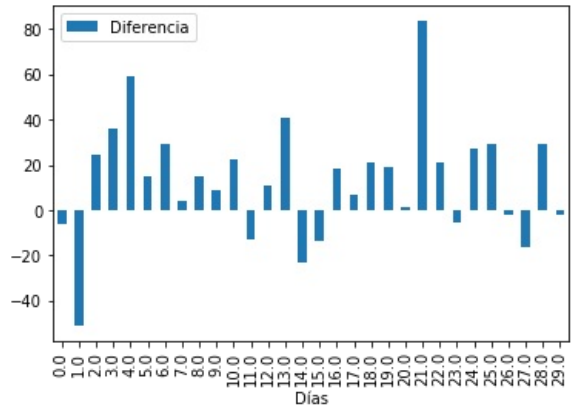
In [27]:

```
model.fit(X_train, Y_train, epochs=150, verbose=0)
Y_pred = model.predict(X_test, verbose=0)[:,0]
print_prediction(Y_test, Y_pred, 30)
```

Out[27]:



Out[27]:



Parte nº6: Guardamos el modelo creado

In [28]:

```
pickle.dump(model, open('NN/NN-5.sav', 'wb'))
```

Modelo nº 6: Nº de avisos para un Mes, día, dia de la semana, base y turno

Paso nº 1: Obtenemos variables

In [9]:

```
#Define X and Y
data = alerts_allyears[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "TURNO_CODIFICADO", "SUCESO"]].g
roupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "TURNO_CODIFICADO"]).count()
data = data.reset_index()
data = data[(np.abs(stats.zscore(data)) < 3).all(axis=1)] #Remove outliers
X = data[["MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "TURNO_CODIFICADO"]]
Y = data["SUCESO"]

#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "TURNO_CODIFICADO", "SUCES
O"]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "TURNO_CODIFICADO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "TURNO_CODIFICADO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "TURNO_CODIFICADO", "SUCESO"
]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "TURNO_CODIFICADO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "TURNO_CODIFICADO"]]
```

In [10]:

```
print("#" * 50)
print(X_train[:5])
print("#" * 50)
print(X_test[:5])
```

```
##################################################
   MES  DIA  DIA_CODIFICADO  BASE_CORREGIDA  TURNO_CODIFICADO
0    1    1               3              -1                 0
1    1    1               3              -1                 1
2    1    1               3              -1                 2
3    1    1               3               1                 0
4    1    1               3               1                 1
##################################################
   MES  DIA  DIA_CODIFICADO  BASE_CORREGIDA  TURNO_CODIFICADO
0    1    1               0              -1                 0
1    1    1               0              -1                 1
2    1    1               0               1                 0
3    1    1               0               1                 1
4    1    1               0               1                 2
```

In [11]:

```
print(X_train.shape)
print(Y_train.shape)
```

```
(181943, 5)
(181943,)
```

Parte nº 2: Trasformamos los datos a array de numpy para poder utilziarlos en Keras

In [12]:

```
X_train = X_train.to_numpy()
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.to_numpy()
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

Parte nº 3: Buscamos el mejor modeloº

In [0]:

```
draw_layer_size(X_train,X_test,Y_test,num_features=5,size=200)
```

```
i =50.0 , j=50.0 --> RMSE:3.7166667412056174
i =50.0 , j=100.0 --> RMSE:3.85940991933275
i =50.0 , j=150.0 --> RMSE:3.8255899358096026
i =50.0 , j=200.0 --> RMSE:3.874599092646378
i =100.0 , j=50.0 --> RMSE:3.7766234692489613
i =100.0 , j=100.0 --> RMSE:3.783455297396447
i =100.0 , j=150.0 --> RMSE:3.860628288310095
i =100.0 , j=200.0 --> RMSE:3.8933444139019486
i =150.0 , j=50.0 --> RMSE:3.826222804230881
i =150.0 , j=100.0 --> RMSE:3.737525709722189
i =150.0 , j=150.0 --> RMSE:3.861517370752199
i =150.0 , j=200.0 --> RMSE:3.8426572424512626
i =200.0 , j=50.0 --> RMSE:3.7681184282204723
i =200.0 , j=100.0 --> RMSE:3.7980196635310457
i =200.0 , j=150.0 --> RMSE:3.8030727515605136
```

Parte nº4: Creamos el modelo

In [13]:

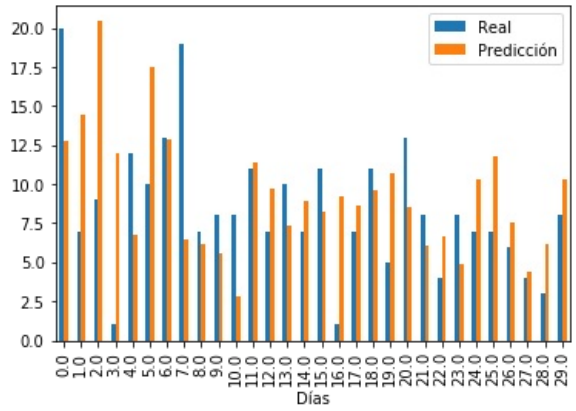
```
model = lstm_model_two_layers(50, 50, num_features=5)
```

Parte nº5: Entrenamos y predecimos los valores de test

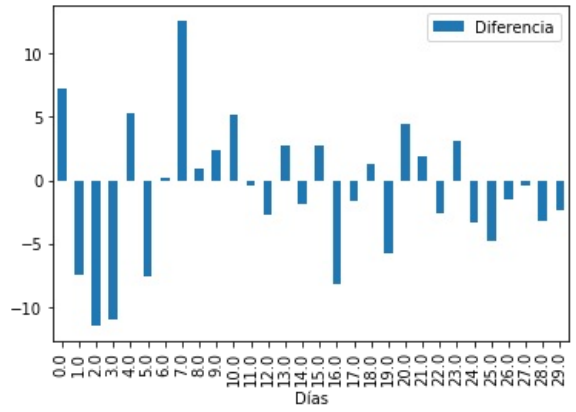
In [14]:

```
model.fit(X_train, Y_train, epochs=150, verbose=0)
Y_pred = model.predict(X_test, verbose=0)[:,0]
print_prediction(Y_test, Y_pred, 30)
```

Out[14]:



Out[14]:



Parte nº6: Guardamos el modelo creado

In [15]:

```
pickle.dump(model, open('NN/NN-6.sav', 'wb'))
```

Redes Neuronales Fase I (II) SAMUR - Protección Civil

Propósito del documento

El objetivo de este documento es la creación de modelos utilizando como técnica de aprendizaje Redes neuronales, para predecir el número de avisos o activaciones que recibirá SAMUR-PC para un determinado día y unas determinadas cirucnstancias en la ciudad de Madrid.

Este documento contiene el código desarrollado para la primera Fase de Redes Neuronales, es decir, el desarrollo se realiza de forma en local utilizando los equipos de los estudiantes, por ello, en esta fase, se crean redes neuronales de cuatroo capas (capa de entrada, dos capas ocultas y capa de salida.

Esta primera fase esta formada por dos documentos:

- `Redes Neuronales Fase I`
- `Redes Neuronales Fase I (II)`

Para cada uno de los modelos se desarrollarán los siguientes pasos:

1. Obtención de variables
2. Transformación de datos para prerparalos para modelos Deep Learning utilizando la librería Keras
3. Búsqueda del mejor modelo
4. Creación del mejor modelo
5. Búsqueda del mejor modelo
6. Exportación del modelo

1.3.1 Modelo nº 7: nº de Avisos para un mes, día , día de la semana, base y dispositivo

Paso nº 1: Obtenemos variables

[8]:
```
#Define X and Y
data = alerts_allyears[["AÑO", "MES", "DIA", "DIA_CODIFICADO",␣
↪"BASE_CORREGIDA", "ABREV_CODIFICADO", "SUCESO"]].groupby(["AÑO", "MES",␣
↪"DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "ABREV_CODIFICADO"]).count()
data = data.reset_index()
data = data[(np.abs(stats.zscore(data)) < 3).all(axis=1)] #Remove outliers
X = data[["MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "ABREV_CODIFICADO"]]
Y = data["SUCESO"]

#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO",␣
↪"BASE_CORREGIDA", "ABREV_CODIFICADO", "SUCESO"]].groupby(["AÑO", "MES",␣
↪"DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "ABREV_CODIFICADO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)]␣
↪#Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA",␣
↪"ABREV_CODIFICADO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO",␣
↪"BASE_CORREGIDA", "ABREV_CODIFICADO", "SUCESO"]].groupby(["AÑO", "MES",␣
↪"DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "ABREV_CODIFICADO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)]␣
↪#Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA",␣
↪"ABREV_CODIFICADO"]]
```

[9]:
```
data_train.head(5)
```

[9]:

	AÑO	MES	DIA	DIA_CODIFICADO	BASE_CORREGIDA	ABREV_CODIFICADO	SUCESO
0	2009	1	1	3	-1	0.0	4
1	2009	1	1	3	-1	2.0	1
2	2009	1	1	3	-1	20.0	15
3	2009	1	1	3	-1	28.0	1
4	2009	1	1	3	-1	30.0	7

[10]:
```
print("#" * 50)
print(X_train[:5])
print("#" * 50)
print(X_test[:5])
```

```
##################################################
   MES  DIA  DIA_CODIFICADO  BASE_CORREGIDA  ABREV_CODIFICADO
0    1    1               3              -1               0.0
1    1    1               3              -1               2.0
2    1    1               3              -1              20.0
3    1    1               3              -1              28.0
4    1    1               3              -1              30.0
##################################################
   MES  DIA  DIA_CODIFICADO  BASE_CORREGIDA  ABREV_CODIFICADO
0    1    1               0              -1               1.0
1    1    1               0              -1               2.0
2    1    1               0              -1              17.0
3    1    1               0              -1              28.0
4    1    1               0              -1              30.0
```

[11]:
```
print(X_train.shape)
print(Y_train.shape)
```

```
(153853, 5)
(153853,)
```

Parte nº 2: Trasformamos los datos a array de numpy para poder utilziarlos en Keras

[12]:
```
X_train = X_train.to_numpy()
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.to_numpy()
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

Parte nº 3: Buscamos el mejor modelo

[ ]:
```
draw_layer_size(X_train,X_test,Y_test,5,300)
```

```
WARNING:tensorflow:From /home/rvindel/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/resource_variable_ops.py:435: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future
version.
```

```
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /home/rvindel/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.cast instead.
i =50.0 , j=50.0 --> RMSE:4.52986147136404
i =50.0 , j=100.0 --> RMSE:4.576415944163831
i =50.0 , j=150.0 --> RMSE:4.64160094496473
```

Nos quedamos con la red que minimza el RMSE y creamos el modelo

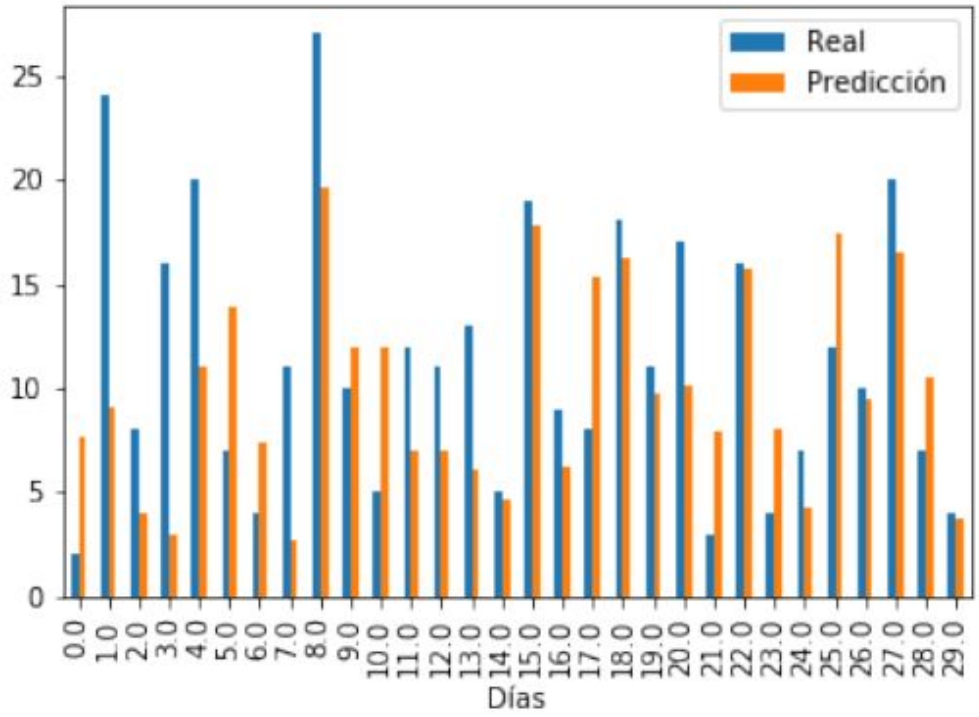
Parte nº4: Creamos el modelo

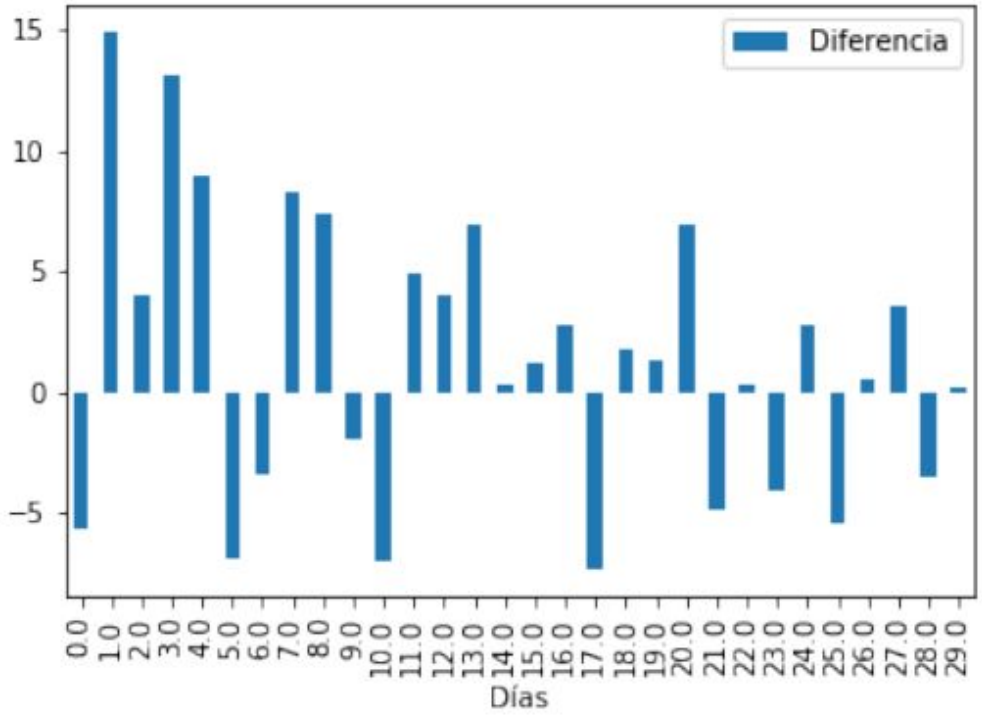
[15]:
```
model = lstm_model_two_layers(50, 50,num_features=5)
```

Parte nº5: Entrenamos y predecimos los valores de test

[16]:
```
model.fit(X_train, Y_train, epochs=150, verbose=0)
Y_pred = model.predict(X_test, verbose=0)[:,0]
print_prediction(Y_test, Y_pred, 30)
```

```
WARNING:tensorflow:From /home/rvindel/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.cast instead.
```


Real
Predicción
25
20
15
10
5
0
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
11.0
12.0
13.0
14.0
15.0
16.0
17.0
18.0
19.0
20.0
21.0
22.0
23.0
24.0
25.0
26.0
27.0
28.0
29.0
Días


Diferencia
15
10
5
0
−5
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
11.0
12.0
13.0
14.0
15.0
16.0
17.0
18.0
19.0
20.0
21.0
22.0
23.0
24.0
25.0
26.0
27.0
28.0
29.0
Días

Parte nº6: Guardamos el modelo creado

[17]:
```
pickle.dump(model, open('NN/NN-7.sav', 'wb'))
```

1.3.2 Modelo nº 8: nº de Avisos para un mes, día , día de la semana, distrito y dispositivo

Paso nº 1: Obtenemos variables

[10]:
```
#Define X and Y
data = alerts_allyears[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DISTRITO",␣
 ↪"ABREV_CODIFICADO", "SUCESO"]].groupby(["AÑO", "MES", "DIA",␣
 ↪"DIA_CODIFICADO", "DISTRITO", "ABREV_CODIFICADO"]).count()
data = data.reset_index()
data = data[(np.abs(stats.zscore(data)) < 3).all(axis=1)] #Remove outliers
X = data[["MES", "DIA", "DIA_CODIFICADO", "DISTRITO", "ABREV_CODIFICADO"]]
Y = data["SUCESO"]

#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO",␣
 ↪"DISTRITO", "ABREV_CODIFICADO", "SUCESO"]].groupby(["AÑO", "MES", "DIA",␣
 ↪"DIA_CODIFICADO", "DISTRITO", "ABREV_CODIFICADO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)]␣
 ↪#Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO", "DISTRITO",␣
 ↪"ABREV_CODIFICADO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DISTRITO",␣
 ↪"ABREV_CODIFICADO", "SUCESO"]].groupby(["AÑO", "MES", "DIA",␣
 ↪"DIA_CODIFICADO", "DISTRITO", "ABREV_CODIFICADO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)]␣
 ↪#Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO", "DISTRITO",␣
 ↪"ABREV_CODIFICADO"]]
```

[11]:
```
data_train.head(5)
```

[11]:

	AÑO	MES	DIA	DIA_CODIFICADO	DISTRITO	ABREV_CODIFICADO	SUCESO
0	2009	1	1	3	0.0	0.0	1
1	2009	1	1	3	0.0	1.0	2
2	2009	1	1	3	1.0	20.0	2
3	2009	1	1	3	1.0	30.0	1
4	2009	1	1	3	2.0	0.0	9

[12]:
```
print("#" * 50)
print(X_train[:5])
print("#" * 50)
print(X_test[:5])
```

```
##################################################
   MES  DIA  DIA_CODIFICADO  DISTRITO  ABREV_CODIFICADO
0    1    1               3       0.0               0.0
1    1    1               3       0.0               1.0
2    1    1               3       1.0              20.0
3    1    1               3       1.0              30.0
4    1    1               3       2.0               0.0
##################################################
   MES  DIA  DIA_CODIFICADO  DISTRITO  ABREV_CODIFICADO
0    1    1               0       1.0               1.0
1    1    1               0       1.0               2.0
2    1    1               0       1.0               4.0
3    1    1               0       1.0              17.0
4    1    1               0       1.0              28.0
```

[13]:
```
print(X_train.shape)
print(Y_train.shape)
```

```
(292179, 5)
(292179,)
```

Parte nº 2: Trasformamos los datos a array de numpy para poder utilziarlos en Keras

[14]:
```
X_train = X_train.to_numpy()
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.to_numpy()
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

Parte nº 3: Buscamos el mejor modelo

[ ]:
```
draw_layer_size(X_train,X_test,Y_test,5,300)
```

```
i =50.0 , j=50.0 --> RMSE:2.3745710234827673
i =50.0 , j=100.0 --> RMSE:2.3894411028720874
i =50.0 , j=150.0 --> RMSE:2.417859942699866
```

Nos quedamos con la red que minimza el RMSE y creamos el modelo

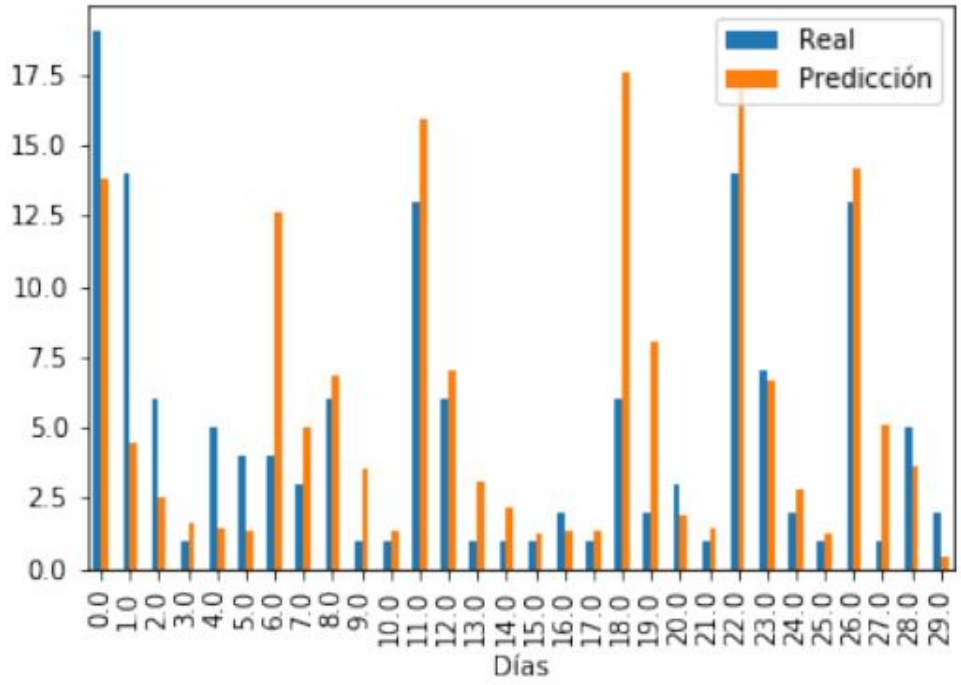
Parte nº4: Creamos el modelo

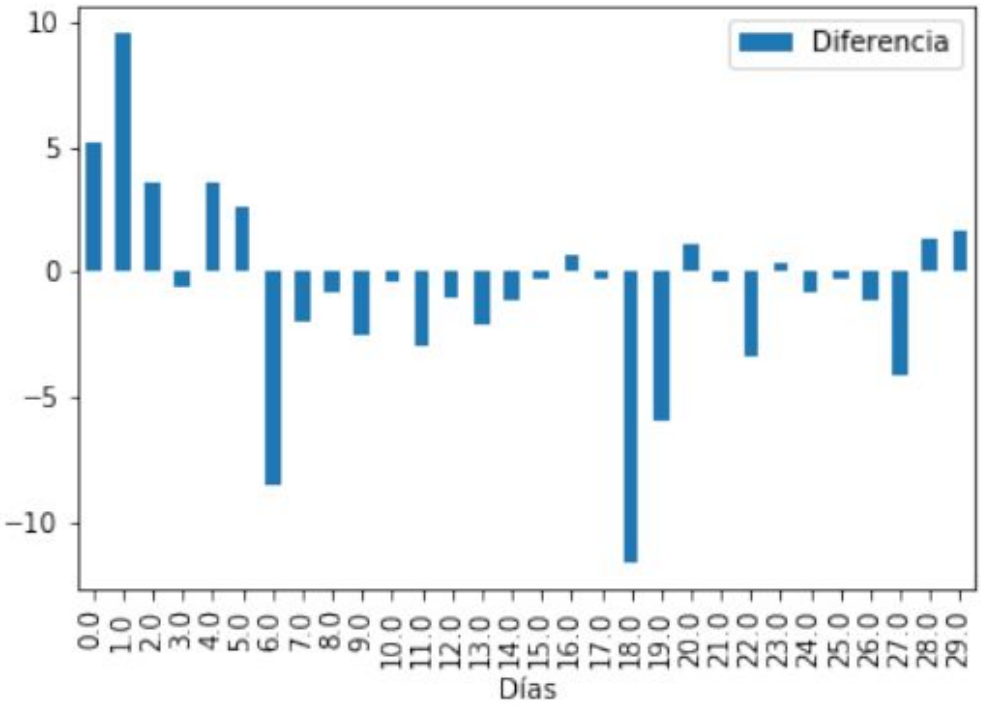
[17]:
```
model = lstm_model_two_layers(50, 50,num_features=5)
```

Parte nº5: Entrenamos y predecimos los valores de test

[18]:
```
model.fit(X_train, Y_train, epochs=150, verbose=0)
Y_pred = model.predict(X_test, verbose=0)[:,0]
print_prediction(Y_test, Y_pred, 30)
```

```
WARNING:tensorflow:From /home/rvindel/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.cast instead.
```





Parte nº6: Guardamos el modelo creado

[19]:
```
pickle.dump(model, open('NN/NN-8.sav', 'wb'))
```

1.3.3 Modelo nº 9: nº de Avisos para un mes, día , día de la semana y dispositivo

Paso nº 1: Obtenemos variables

[8]:
```
#Define X and Y
data = alerts_allyears[["AÑO", "MES", "DIA", "DIA_CODIFICADO",␣
 ↪"ABREV_CODIFICADO", "SUCESO"]].groupby(["AÑO", "MES", "DIA",␣
 ↪"DIA_CODIFICADO", "ABREV_CODIFICADO"]).count()
data = data.reset_index()
data = data[(np.abs(stats.zscore(data)) < 3).all(axis=1)] #Remove outliers
X = data[["MES", "DIA", "DIA_CODIFICADO", "ABREV_CODIFICADO"]]
Y = data["SUCESO"]

#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO",␣
 ↪"ABREV_CODIFICADO", "SUCESO"]].groupby(["AÑO", "MES", "DIA",␣
 ↪"DIA_CODIFICADO", "ABREV_CODIFICADO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)]␣
 ↪#Eliminamos outliers
```

```
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO", "ABREV_CODIFICADO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO",␣
 ↪"ABREV_CODIFICADO", "SUCESO"]].groupby(["AÑO", "MES", "DIA",␣
 ↪"DIA_CODIFICADO", "ABREV_CODIFICADO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)]␣
 ↪#Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO", "ABREV_CODIFICADO"]]
```

[9]:
```
data_train.head(5)
```

[9]:

	AÑO	MES	DIA	DIA_CODIFICADO	ABREV_CODIFICADO	SUCESO
0	2009	1	1	3	1.0	126
1	2009	1	1	3	2.0	87
2	2009	1	1	3	18.0	7
3	2009	1	1	3	20.0	15
4	2009	1	1	3	28.0	1

[10]:
```
print("#" * 50)
print(X_train[:5])
print("#" * 50)
print(X_test[:5])
```

```
##################################################
   MES  DIA  DIA_CODIFICADO  ABREV_CODIFICADO
0    1    1               3               1.0
1    1    1               3               2.0
2    1    1               3              18.0
3    1    1               3              20.0
4    1    1               3              28.0
##################################################
   MES  DIA  DIA_CODIFICADO  ABREV_CODIFICADO
0    1    1               0               0.0
1    1    1               0               1.0
2    1    1               0               2.0
3    1    1               0               4.0
4    1    1               0              17.0
```

[11]:
```
print(X_train.shape)
print(Y_train.shape)
```

```
(36447, 4)


(36447,)
```

Parte nº 2: Trasformamos los datos a array de numpy para poder utilziarlos en Keras

[12]:
```
X_train = X_train.to_numpy()
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.to_numpy()
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

Parte nº 3: Buscamos el mejor modelo

[ ]:
```
draw_layer_size(X_train,X_test,Y_test,4,300)
```

```
WARNING:tensorflow:From /home/rvindel/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/resource_variable_ops.py:435: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /home/rvindel/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.cast instead.
i =50.0 , j=50.0 --> RMSE:16.158758364056116
i =50.0 , j=100.0 --> RMSE:15.981843361560585
i =50.0 , j=150.0 --> RMSE:16.392084702118705
i =50.0 , j=200.0 --> RMSE:16.868638368227792
```

Nos quedamos con la red que minimza el RMSE y creamos el modelo

Parte nº4: Creamos el modelo

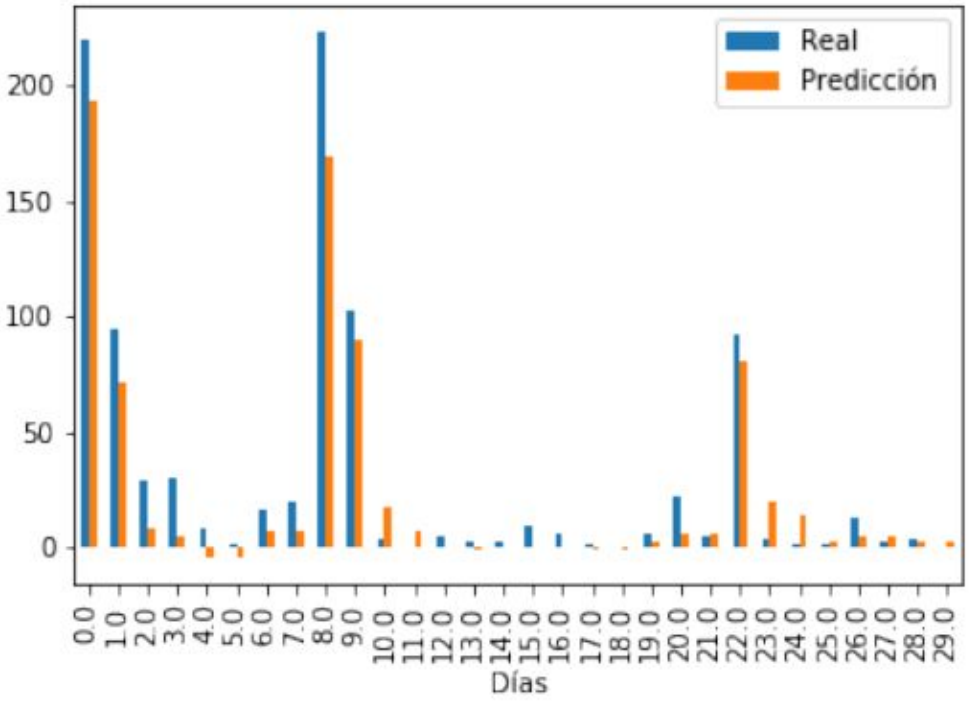
[13]:
```
model = lstm_model_two_layers(50, 100,num_features=4)
```

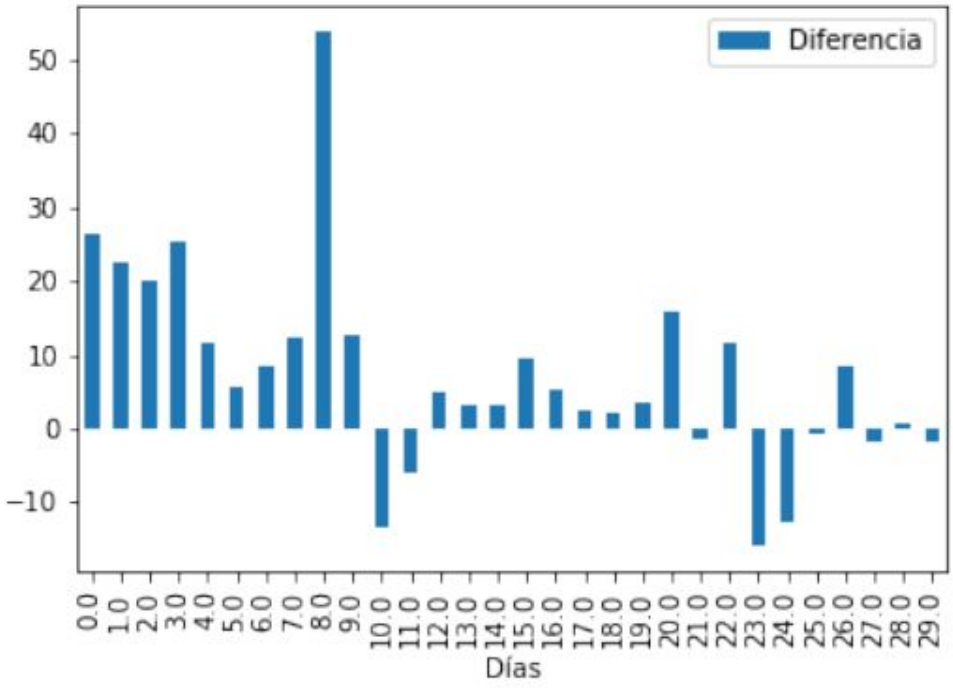
```
WARNING:tensorflow:From /home/rvindel/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/resource_variable_ops.py:435: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Colocations handled automatically by placer.
```

Parte nº5: Entrenamos y predecimos los valores de test

[14]:
```
model.fit(X_train, Y_train, epochs=150, verbose=0)
Y_pred = model.predict(X_test, verbose=0)[:,0]
print_prediction(Y_test, Y_pred, 30)
```

```
WARNING:tensorflow:From /home/rvindel/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.cast instead.
```





Parte nº6: Guardamos el modelo creado

[15]:
```
pickle.dump(model, open('NN/NN-9.sav', 'wb'))
```

1.3.4 Modelo nº 10: nº de Avisos para un mes, día , día de la semana, base, turno y dispositivo

Paso nº 1: Obtenemos variables

[10]:
```
#Define X and Y
data = alerts_allyears[["AÑO", "MES", "DIA", "DIA_CODIFICADO",␣
 ↪"BASE_CORREGIDA", "TURNO_CODIFICADO", "ABREV_CODIFICADO", "SUCESO"]].
 ↪groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA",␣
 ↪"TURNO_CODIFICADO", "ABREV_CODIFICADO"]).count()
data = data.reset_index()
data = data[(np.abs(stats.zscore(data)) < 3).all(axis=1)] #Remove outliers
X = data[["MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA", "TURNO_CODIFICADO",␣
 ↪"ABREV_CODIFICADO"]]
Y = data["SUCESO"]

#Define X and Y for training
```

```
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO",␣
 ↪"BASE_CORREGIDA", "TURNO_CODIFICADO", "ABREV_CODIFICADO", "SUCESO"]].
 ↪groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA",␣
 ↪"TURNO_CODIFICADO", "ABREV_CODIFICADO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)]␣
 ↪#Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA",␣
 ↪"TURNO_CODIFICADO", "ABREV_CODIFICADO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO",␣
 ↪"BASE_CORREGIDA", "TURNO_CODIFICADO", "ABREV_CODIFICADO", "SUCESO"]].
 ↪groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA",␣
 ↪"TURNO_CODIFICADO", "ABREV_CODIFICADO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)]␣
 ↪#Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO", "BASE_CORREGIDA",␣
 ↪"TURNO_CODIFICADO", "ABREV_CODIFICADO"]]
```

[11]:
```
data_train.head(5)
```

[11]:

	AÑO	MES	DIA	DIA_CODIFICADO	BASE_CORREGIDA	TURNO_CODIFICADO	ABREV_CODIFICADO	SUCESO
0	2009	1	1	3	-1	0	0.0	4
1	2009	1	1	3	-1	0	20.0	2
2	2009	1	1	3	-1	0	28.0	1
3	2009	1	1	3	-1	0	30.0	1
4	2009	1	1	3	-1	1	20.0	3

[12]:
```
print("#" * 50)
print(X_train[:5])
print("#" * 50)
print(X_test[:5])
```

```
##################################################
   MES  DIA  DIA_CODIFICADO  BASE_CORREGIDA  TURNO_CODIFICADO  \
0    1    1               3              -1                 0
```

```
1    1    1              3              -1                0
2    1    1              3              -1                0
3    1    1              3              -1                0
4    1    1              3              -1                1

   ABREV_CODIFICADO
0               0.0
1              20.0
2              28.0
3              30.0
4              20.0
##################################################
   MES  DIA  DIA_CODIFICADO  BASE_CORREGIDA  TURNO_CODIFICADO  \
0    1    1               0              -1                 0
1    1    1               0              -1                 0
2    1    1               0              -1                 0
3    1    1               0              -1                 0
4    1    1               0              -1                 1

   ABREV_CODIFICADO
0               2.0
1              17.0
2              28.0
3              30.0
4              17.0
```

[13]:
```
print(X_train.shape)
print(Y_train.shape)
```

```
(328360, 6)
(328360,)
```

Parte nº 2: Trasformamos los datos a array de numpy para poder utilziarlos en Keras

[14]:
```
X_train = X_train.to_numpy()
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.to_numpy()
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

Parte nº 3: Buscamos el mejor modelo

[ ]:
```
draw_layer_size(X_train,X_test,Y_test,6,300)
```

```
WARNING:tensorflow:From /home/rvindel/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/resource_variable_ops.py:435: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future
version.
Instructions for updating:
```

```
Colocations handled automatically by placer.
WARNING:tensorflow:From /home/rvindel/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.cast instead.
i =50.0 , j=50.0 --> RMSE:2.0707067131543893
i =50.0 , j=100.0 --> RMSE:2.111064540123367
i =50.0 , j=150.0 --> RMSE:2.148737572597092
```

Nos quedamos con la red que minimza el RMSE y creamos el modelo

Parte nº4: Creamos el modelo

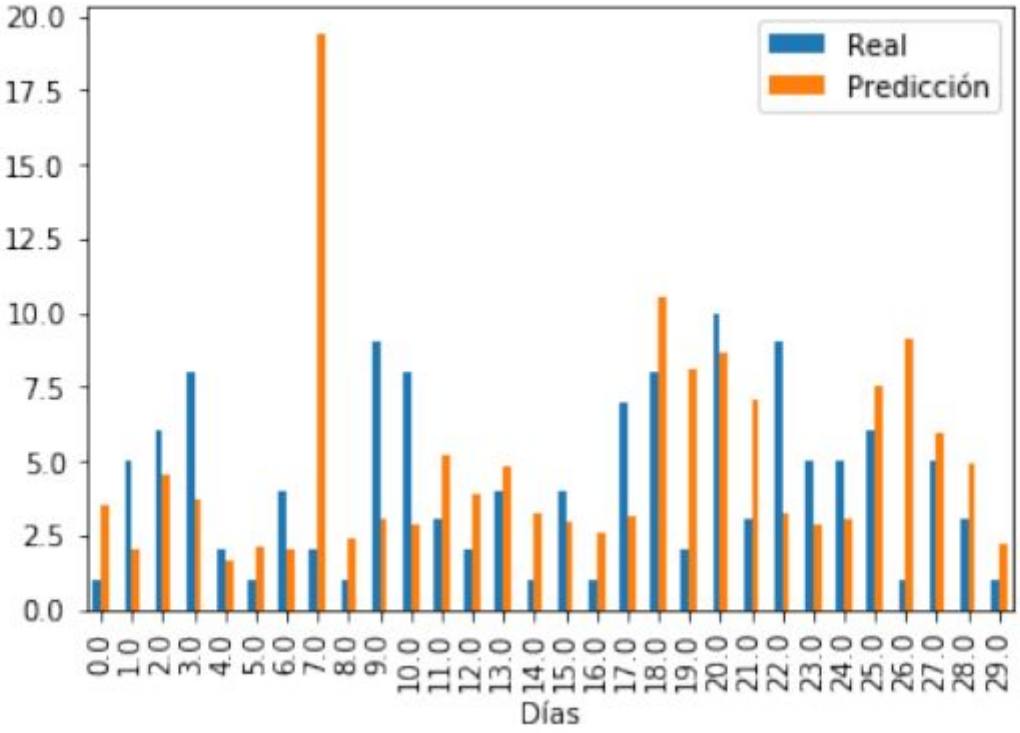
[8]:
```
model = lstm_model_two_layers(50, 50,num_features=6)
```

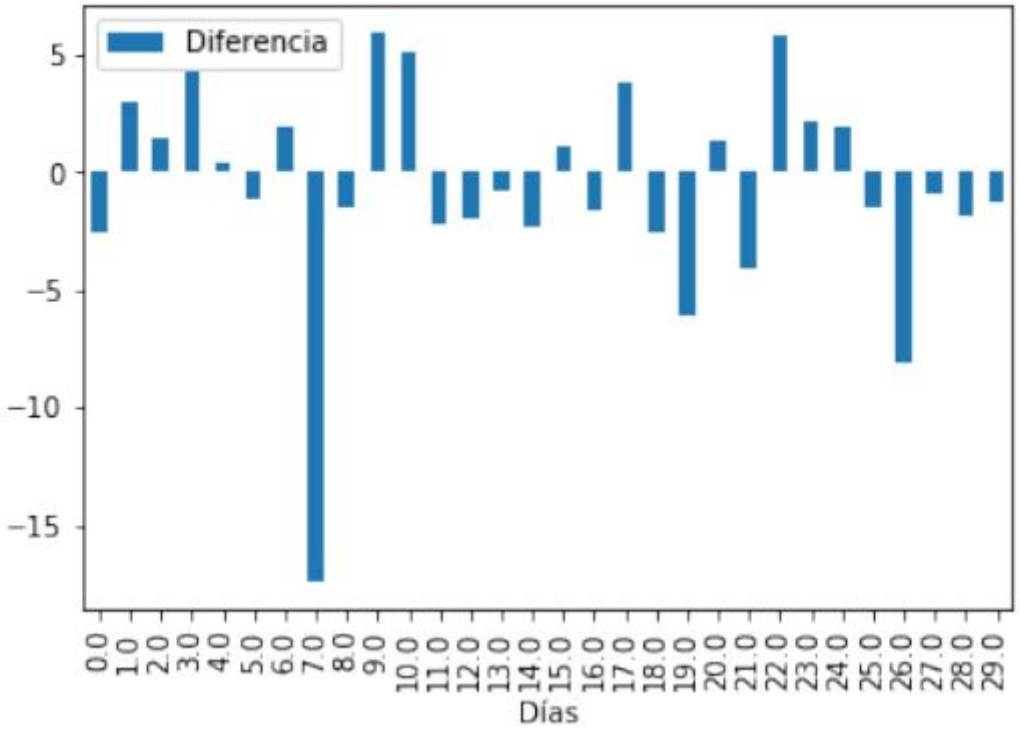
```
WARNING:tensorflow:From /home/rvindel/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/resource_variable_ops.py:435: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Colocations handled automatically by placer.
```

Parte nº5: Entrenamos y predecimos los valores de test

[15]:
```
model.fit(X_train, Y_train, epochs=150, verbose=0)
Y_pred = model.predict(X_test, verbose=0)[:,0]
print_prediction(Y_test, Y_pred, 30)
```

```
WARNING:tensorflow:From /home/rvindel/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.cast instead.
```


Real
Predicción
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 13.0 14.0 15.0 16.0 17.0 18.0 19.0 20.0 21.0 22.0 23.0 24.0 25.0 26.0 27.0 28.0 29.0
Días


Diferencia
5
0
−5
−10
−15
0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 13.0 14.0 15.0 16.0 17.0 18.0 19.0 20.0 21.0 22.0 23.0 24.0 25.0 26.0 27.0 28.0 29.0
Días

Parte nº6: Guardamos el modelo creado

[16]:
```
pickle.dump(model, open('NN/NN-10.sav', 'wb'))
```

1.3.5 Modelo nº 11: nº de Avisos para un mes, día , día de la semana, distrito, turno y dispositivo

Paso nº 1: Obtenemos variables

[8]:
```
#Define X and Y
data = alerts_allyears[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DISTRITO",␣
 ↪"TURNO_CODIFICADO", "ABREV_CODIFICADO", "SUCESO"]].groupby(["AÑO", "MES",␣
 ↪"DIA", "DIA_CODIFICADO", "DISTRITO", "TURNO_CODIFICADO",␣
 ↪"ABREV_CODIFICADO"]).count()
data = data.reset_index()
data = data[(np.abs(stats.zscore(data)) < 3).all(axis=1)] #Remove outliers
X = data[["MES", "DIA", "DIA_CODIFICADO", "DISTRITO", "TURNO_CODIFICADO",␣
 ↪"ABREV_CODIFICADO"]]
Y = data["SUCESO"]

#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO",␣
 ↪"DISTRITO", "TURNO_CODIFICADO", "ABREV_CODIFICADO", "SUCESO"]].
 ↪groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DISTRITO",␣
 ↪"TURNO_CODIFICADO", "ABREV_CODIFICADO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)]␣
 ↪#Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO", "DISTRITO",␣
 ↪"TURNO_CODIFICADO", "ABREV_CODIFICADO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DISTRITO",␣
 ↪"TURNO_CODIFICADO", "ABREV_CODIFICADO", "SUCESO"]].groupby(["AÑO", "MES",␣
 ↪"DIA", "DIA_CODIFICADO", "DISTRITO", "TURNO_CODIFICADO",␣
 ↪"ABREV_CODIFICADO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)]␣
 ↪#Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO", "DISTRITO",␣
 ↪"TURNO_CODIFICADO", "ABREV_CODIFICADO"]]
```

[9]:
```
data_train.head(5)
```

[9]:

	AÑO	MES	DIA	DIA_CODIFICADO	DISTRITO	TURNO_CODIFICADO	\
0	2009	1	1	3	0.0	1	
1	2009	1	1	3	0.0	1	
2	2009	1	1	3	1.0	0	
3	2009	1	1	3	1.0	0	
4	2009	1	1	3	1.0	1	

	ABREV_CODIFICADO	SUCESO
0	0.0	1
1	1.0	2
2	1.0	7
3	2.0	3
4	1.0	7

[10]:
```
print("#" * 50)
print(X_train[:5])
print("#" * 50)
print(X_test[:5])
```

```
##################################################
   MES  DIA  DIA_CODIFICADO  DISTRITO  TURNO_CODIFICADO  ABREV_CODIFICADO
0    1    1               3       0.0                 1               0.0
1    1    1               3       0.0                 1               1.0
2    1    1               3       1.0                 0               1.0
3    1    1               3       1.0                 0               2.0
4    1    1               3       1.0                 1               1.0
##################################################
   MES  DIA  DIA_CODIFICADO  DISTRITO  TURNO_CODIFICADO  ABREV_CODIFICADO
0    1    1               0       1.0                 0               1.0
1    1    1               0       1.0                 0               4.0
2    1    1               0       1.0                 0              17.0
3    1    1               0       1.0                 0              28.0
4    1    1               0       1.0                 0              30.0
```

[11]:
```
print(X_train.shape)
print(Y_train.shape)
```

```
(507875, 6)
(507875,)
```

Parte nº 2: Trasformamos los datos a array de numpy para poder utilziarlos en Keras

[12]:
```
X_train = X_train.to_numpy()
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.to_numpy()
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

Parte nº 3: Buscamos el mejor modelo

[13]:
```
draw_layer_size(X_train,X_test,Y_test,6,300)
```

```
WARNING:tensorflow:From /home/rvindel/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/resource_variable_ops.py:435: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /home/rvindel/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.cast instead.
i =50.0 , j=50.0 --> RMSE:1.4853265972344554
i =50.0 , j=100.0 --> RMSE:1.4846118570820994
```

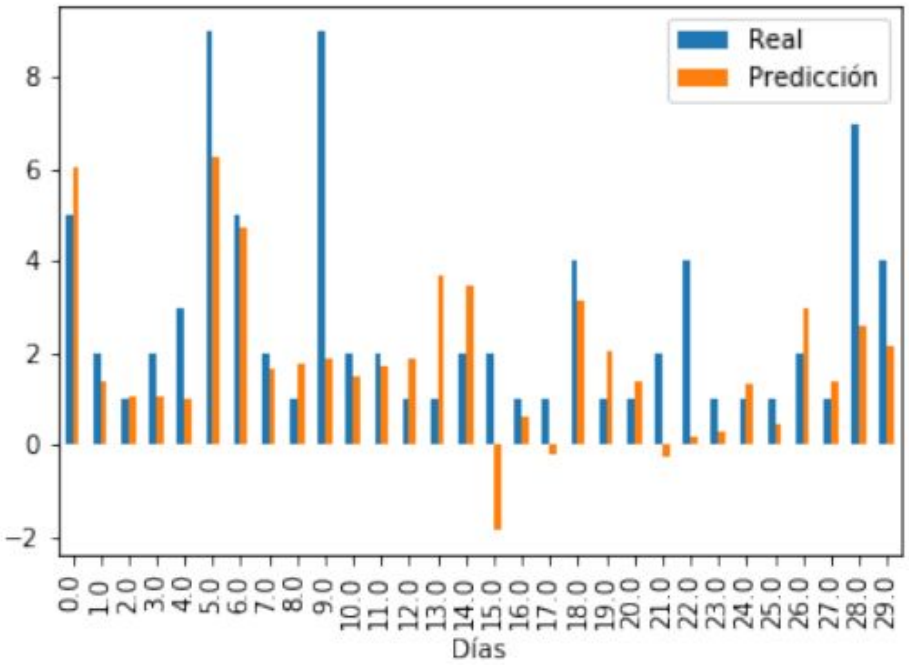
Nos quedamos con la red que minimza el RMSE y creamos el modelo

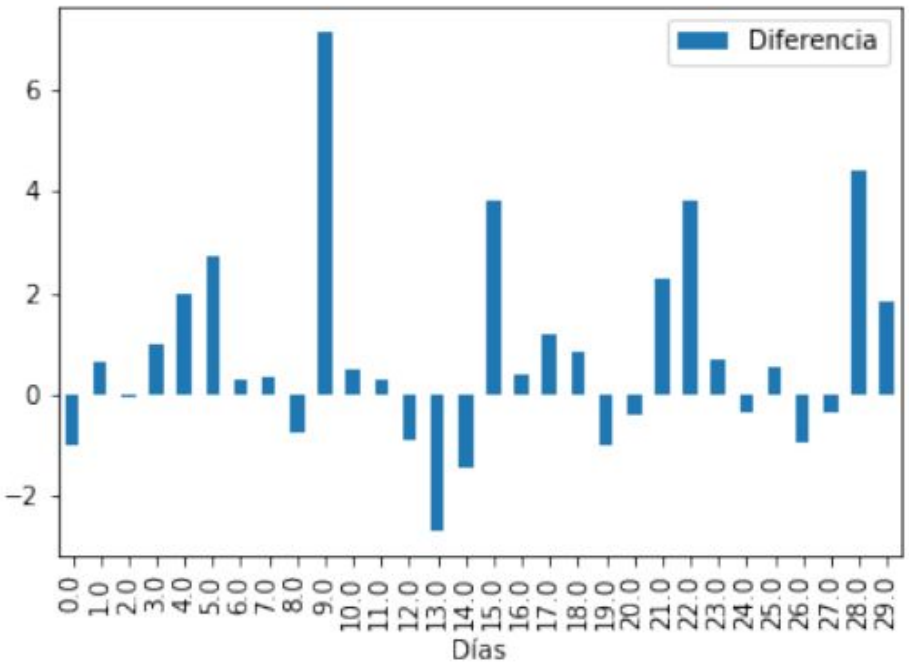
Parte nº4: Creamos el modelo

[14]:
```
model = lstm_model_two_layers(50, 100,num_features=6)
```

Parte nº5: Entrenamos y predecimos los valores de test

[15]:
```
model.fit(X_train, Y_train, epochs=150, verbose=0)
Y_pred = model.predict(X_test, verbose=0)[:,0]
print_prediction(Y_test, Y_pred, 30)
```


Real
Predicción
8
6
4
2
0
−2
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
11.0
12.0
13.0
14.0
15.0
16.0
17.0
18.0
19.0
20.0
21.0
22.0
23.0
24.0
25.0
26.0
27.0
28.0
29.0
Días


Diferencia
6
4
2
0
−2
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
11.0
12.0
13.0
14.0
15.0
16.0
17.0
18.0
19.0
20.0
21.0
22.0
23.0
24.0
25.0
26.0
27.0
28.0
29.0
Días

Parte nº6: Guardamos el modelo creado

[16]:
```
pickle.dump(model, open('NN/NN-11.sav', 'wb'))
```

1.3.6 Modelo nº 12: nº de Avisos para un mes, día , día de la semana, dispositivo y turno

Paso nº 1: Obtenemos variables

[11]:
```
#Define X and Y
data = alerts_allyears[["AÑO", "MES", "DIA", "DIA_CODIFICADO",␣
 ↪"TURNO_CODIFICADO", "ABREV_CODIFICADO", "SUCESO"]].groupby(["AÑO", "MES",␣
 ↪"DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO", "ABREV_CODIFICADO"]).count()
data = data.reset_index()
data = data[(np.abs(stats.zscore(data)) < 3).all(axis=1)] #Remove outliers
X = data[["MES", "DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO",␣
 ↪"ABREV_CODIFICADO"]]
Y = data["SUCESO"]

#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO",␣
 ↪"TURNO_CODIFICADO", "ABREV_CODIFICADO", "SUCESO"]].groupby(["AÑO", "MES",␣
 ↪"DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO", "ABREV_CODIFICADO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)]␣
 ↪#Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO",␣
 ↪"ABREV_CODIFICADO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO",␣
 ↪"TURNO_CODIFICADO", "ABREV_CODIFICADO", "SUCESO"]].groupby(["AÑO", "MES",␣
 ↪"DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO", "ABREV_CODIFICADO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)]␣
 ↪#Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO", "TURNO_CODIFICADO",␣
 ↪"ABREV_CODIFICADO"]]
```

[12]:
```
data_train.head(5)
```

[12]:

	AÑO	MES	DIA	DIA_CODIFICADO	TURNO_CODIFICADO	ABREV_CODIFICADO	SUCESO
0	2009	1	1	3	0	1.0	52
1	2009	1	1	3	0	2.0	7
2	2009	1	1	3	0	18.0	3
3	2009	1	1	3	0	20.0	2
4	2009	1	1	3	0	28.0	1

[13]:
```
print("#" * 50)
print(X_train[:5])
print("#" * 50)
print(X_test[:5])
```

```
##################################################
   MES  DIA  DIA_CODIFICADO  TURNO_CODIFICADO  ABREV_CODIFICADO
0    1    1               3                 0               1.0
1    1    1               3                 0               2.0
2    1    1               3                 0              18.0
3    1    1               3                 0              20.0
4    1    1               3                 0              28.0
##################################################
   MES  DIA  DIA_CODIFICADO  TURNO_CODIFICADO  ABREV_CODIFICADO
0    1    1               0                 0               0.0
1    1    1               0                 0               1.0
2    1    1               0                 0               2.0
3    1    1               0                 0               4.0
4    1    1               0                 0              17.0
```

[14]:
```
print(X_train.shape)
print(Y_train.shape)
```

```
(82190, 5)
(82190,)
```

Parte nº 2: Trasformamos los datos a array de numpy para poder utilziarlos en Keras

[15]:
```
X_train = X_train.to_numpy()
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.to_numpy()
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

Parte nº 3: Buscamos el mejor modelo

[16]:
```
draw_layer_size(X_train,X_test,Y_test,5,300)
```

```
WARNING:tensorflow:From /home/rvindel/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/resource_variable_ops.py:435: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future
version.
```

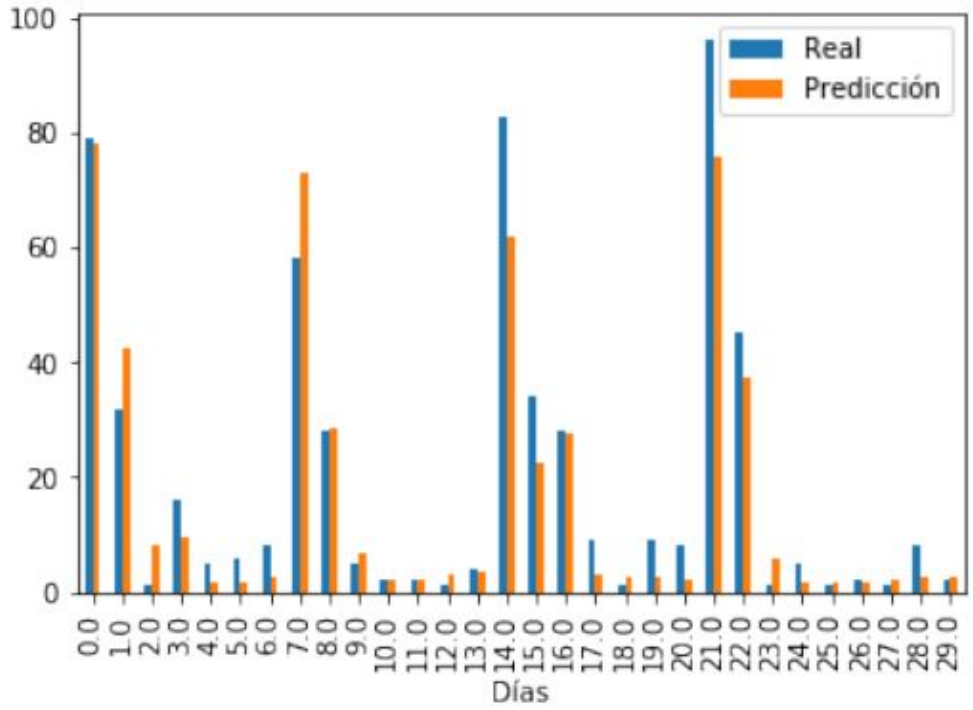
```
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /home/rvindel/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.cast instead.
i =50.0 , j=50.0 --> RMSE:6.842630171541637
i =50.0 , j=100.0 --> RMSE:6.643299267253301
i =50.0 , j=150.0 --> RMSE:6.709005680153764
i =50.0 , j=200.0 --> RMSE:6.9139361883249615
```

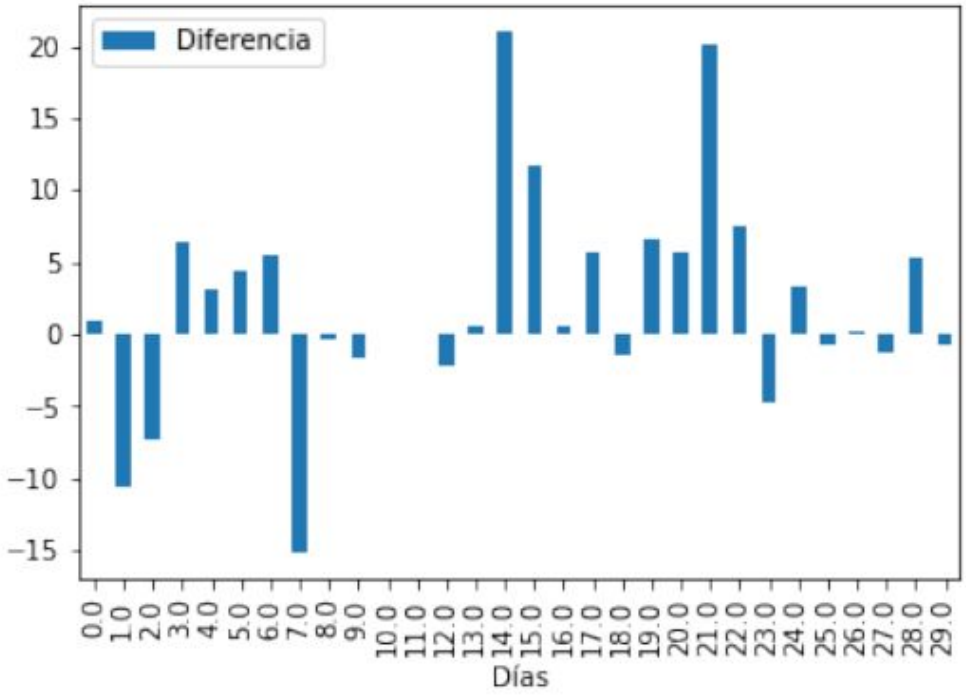
Parte nº4: Creamos el modelo

[17]:
```
model = lstm_model_two_layers(50, 100,num_features=5)
```

Parte nº5: Entrenamos y predecimos los valores de test

[18]:
```
model.fit(X_train, Y_train, epochs=150, verbose=0)
Y_pred = model.predict(X_test, verbose=0)[:,0]
print_prediction(Y_test, Y_pred, 30)
```





Parte nº6: Guardamos el modelo creado

[19]:
```
pickle.dump(model, open('NN/NN-12.sav', 'wb'))
```

1.3.7 Modelo nº 13: Nº de avisos para un mes, día , día de la semana, distrito, turno

Paso nº 1: Obtenemos variables

[17]:
```
#Define X and Y
data = alerts_allyears[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DISTRITO",␣
 ↪"TURNO_CODIFICADO", "SUCESO"]].groupby(["AÑO", "MES", "DIA",␣
 ↪"DIA_CODIFICADO", "DISTRITO", "TURNO_CODIFICADO"]).count()
data = data.reset_index()
data = data[(np.abs(stats.zscore(data)) < 3).all(axis=1)] #Remove outliers
X = data[["MES", "DIA", "DIA_CODIFICADO", "DISTRITO", "TURNO_CODIFICADO"]]
Y = data["SUCESO"]

#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO",␣
 ↪"DISTRITO", "TURNO_CODIFICADO", "SUCESO"]].groupby(["AÑO", "MES", "DIA",␣
 ↪"DIA_CODIFICADO", "DISTRITO", "TURNO_CODIFICADO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)]␣
 ↪#Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO", "DISTRITO",␣
 ↪"TURNO_CODIFICADO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DISTRITO",␣
 ↪"TURNO_CODIFICADO", "SUCESO"]].groupby(["AÑO", "MES", "DIA",␣
 ↪"DIA_CODIFICADO", "DISTRITO", "TURNO_CODIFICADO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)]␣
 ↪#Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO", "DISTRITO",␣
 ↪"TURNO_CODIFICADO"]]
```

[18]:
```
data_train.head(5)
```

[18]:

	AÑO	MES	DIA	DIA_CODIFICADO	DISTRITO	TURNO_CODIFICADO	SUCESO
0	2009	1	1	3	0.0	1	3
1	2009	1	1	3	2.0	0	6
2	2009	1	1	3	2.0	1	3
3	2009	1	1	3	2.0	2	8
4	2009	1	1	3	3.0	0	2

[19]:
```
print("#" * 50)
print(X_train[:5])
print("#" * 50)
print(X_test[:5])
```

```
##################################################
   MES  DIA  DIA_CODIFICADO  DISTRITO  TURNO_CODIFICADO
0    1    1               3       0.0                 1
1    1    1               3       2.0                 0
2    1    1               3       2.0                 1
3    1    1               3       2.0                 2
4    1    1               3       3.0                 0
##################################################
   MES  DIA  DIA_CODIFICADO  DISTRITO  TURNO_CODIFICADO
0    1    1               0       1.0                 1
1    1    1               0       2.0                 0
2    1    1               0       2.0                 1
3    1    1               0       2.0                 2
4    1    1               0       3.0                 0
```

[20]:
```
print(X_train.shape)
print(Y_train.shape)
```

```
(202443, 5)
(202443,)
```

Parte nº 2: Trasformamos los datos a array de numpy para poder utilziarlos en Keras

[21]:
```
X_train = X_train.to_numpy()
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.to_numpy()
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

Parte nº 3: Buscamos el mejor modelo

[22]:
```
draw_layer_size(X_train,X_test,Y_test,5,300)
```

```
WARNING:tensorflow:From /home/rvindel/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/resource_variable_ops.py:435: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /home/rvindel/anaconda3/lib/python3.7/site-
packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
```

```
Use tf.cast instead.
i =50.0 , j=50.0 --> RMSE:3.3338765044028458
i =50.0 , j=100.0 --> RMSE:3.3807466045993055
i =50.0 , j=150.0 --> RMSE:3.434255356697046
i =50.0 , j=200.0 --> RMSE:3.4849694574006036
i =50.0 , j=250.0 --> RMSE:3.5438369811003647
```

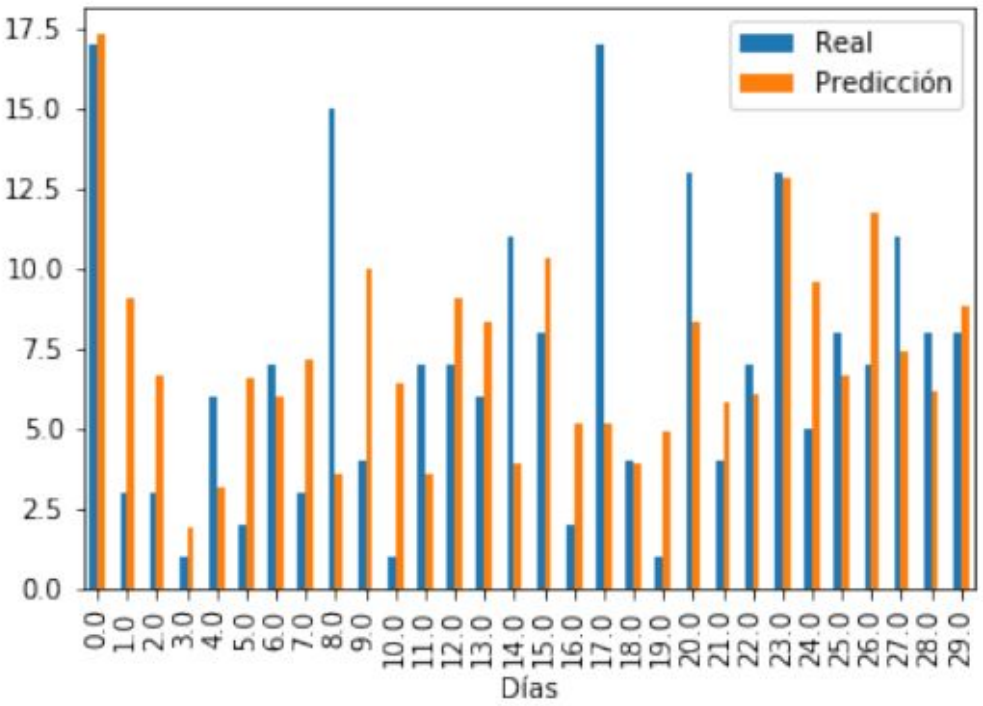
Nos quedamos con la red que minimza el RMSE y creamos el modelo

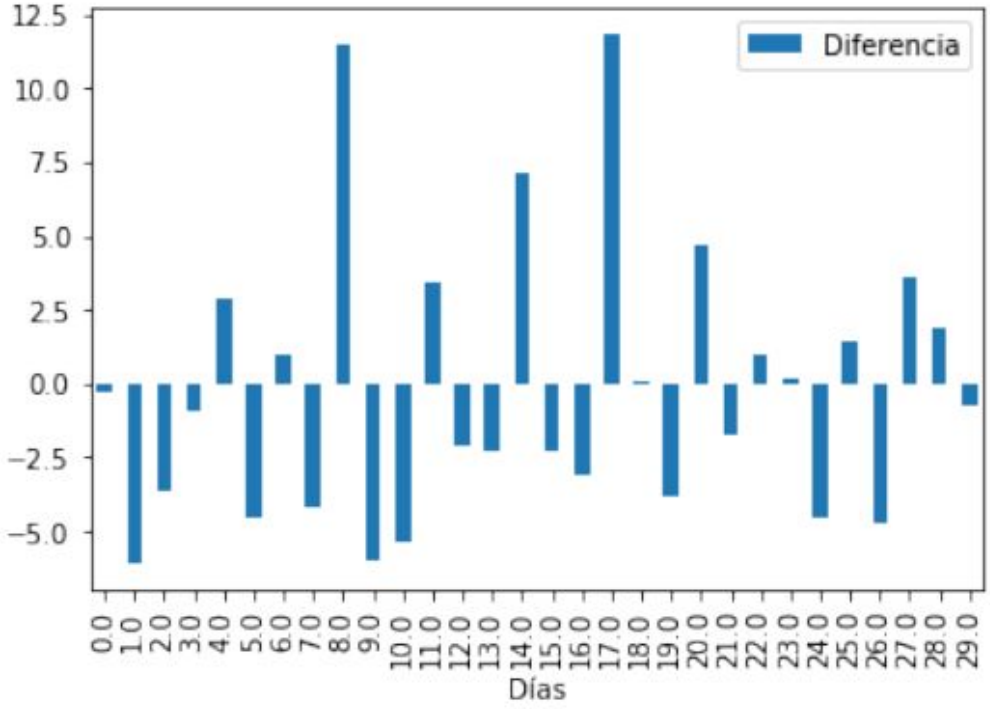
Parte nº4: Creamos el modelo

[26]:
```
model = lstm_model_two_layers(50, 50,num_features=5)
```

Parte nº5: Entrenamos y predecimos los valores de test

[27]:
```
model.fit(X_train, Y_train, epochs=150, verbose=0)
Y_pred = model.predict(X_test, verbose=0)[:,0]
print_prediction(Y_test, Y_pred, 30)
```





Parte nº6: Guardamos el modelo creado

[28]:
```
pickle.dump(model, open('NN/NN-13.sav', 'wb'))
```

Redes Neuronales Fase II SAMUR - Protección Civil

Propósito del documento

El objetivo de este documento es la creación de modelos utilizando como técnica de aprendizaje Redes neuronales, para predecir el número de avisos o activaciones que recibirá SAMUR-PC para un determinado día y unas determinadas cirucnstancias en la ciudad de Madrid. Este documento contiene el código desarrollado para la segunda Fase de Redes Neuronales, es decir, el desarrollo se realiza utilizando la CPU de Google Collaboratory.

Esta segunda fase esta formada por dos documentos:

- Redes Neuronales Fase II
- Redes Neuronales Fase II (II)

Para cada uno de los modelos se desarrollarán los siguientes pasos:

1. Obtención de variables
2. Transformación de datos para prepararlos para modelos Deep Learning utilizando la librería Keras
3. Búsqueda del mejor número de capas del modelo
4. Búsqueda del mejor número de nodos en cada capa del modelo
5. Exportación del modelo

Librerías

In [0]:

```
#general libraries
import pandas as pd
from matplotlib import pyplot as plt
import numpy as np
from math import sqrt
from scipy import stats
import pickle
import torch
import math
import warnings
import timeit
import os
from random import randint
from IPython.display import Image
warnings.filterwarnings("ignore")
from numpy.random import RandomState

#google libraries
from google.colab import auth
import gspread
from oauth2client.client import GoogleCredentials
from google.colab import drive
from google.colab import files

#deep learning libraries
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.utils.vis_utils import plot_model
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn import metrics
from keras.layers import Flatten
from keras.layers import LSTM
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
tf.random.set_seed(2)
```

```
Using TensorFlow backend.
```

In [0]:

```
auth.authenticate_user()
gc = gspread.authorize(GoogleCredentials.get_application_default())
```

In [0]:

```
os.environ['PYTHONHASHSEED'] = '1'
np.random.seed(1)
tf.random.set_seed(1)
```

In [0]:

```
#create woorkbooks
wb2009 = gc.open_by_url('https://docs.google.com/spreadsheets/d/1uRXslPOaa99Jj9xZeaXIZk546lgl7BbozpCTzIg-5Xk/edit
#gid=340244330')
wb2010 = gc.open_by_url('https://docs.google.com/spreadsheets/d/12zcTH0VL57AptiXjpXOHKXehGpIu2W7vKhrkw-liQFs/edit
#gid=825218595')
wb2011 = gc.open_by_url('https://docs.google.com/spreadsheets/d/1az4qe5Pa1-dNHIZIlkt8cTDk0itJRsl5Qs5RYrJNlo0/edit
#gid=50080999')
wb2012 = gc.open_by_url('https://docs.google.com/spreadsheets/d/1kH5h9NMtgOHYizGpAMcyTCwsuflf-CRrGmaQwc1uY0c/edit
#gid=1733288549')
wb2013 = gc.open_by_url('https://docs.google.com/spreadsheets/d/1Ic1PBb8mJAHogjvnCJCVcyydcEtWPvTVWgMh_xxfvXI/edit
#gid=1474377252')
wb2014 = gc.open_by_url('https://docs.google.com/spreadsheets/d/1fp5hHXUcb6WR4nKSh5K1rnvb0tV50PHKPlTVau11rIQ/edit
#gid=1344814184')
wb2015 = gc.open_by_url('https://docs.google.com/spreadsheets/d/1FawfEvToC2i3-FlxAuXDoiteonvdKNpd7qPH8OtGzLA/edit
#gid=158132632')
wb2016 = gc.open_by_url('https://docs.google.com/spreadsheets/d/1gh3FSz32hhoaBm6Ez0xrkN3iYIV-xN2XvlJNJqIMrPY/edit
#gid=1268380804')
wb2017 = gc.open_by_url('https://docs.google.com/spreadsheets/d/1bNF2LsY75tUbhfZm448zG7dUxzehHtP2KCy0GD1biPE/edit
#gid=1309243119')
wb2018 = gc.open_by_url('https://docs.google.com/spreadsheets/d/1qQszEZUi-URftAtBpyruqvXQKFU06UyNz4DmKkewmn8/edit
#gid=267435973')
wb2019 = gc.open_by_url('https://docs.google.com/spreadsheets/d/1T5dWfsEbUO2lxJzRyNyuIIPbCrjZbt3lR6OsVXr0yN8/edit
#gid=2122785558')

#get sheet
sheet2009 = wb2009.worksheet('Sheet1')
sheet2010 = wb2010.worksheet('Sheet1')
sheet2011 = wb2011.worksheet('Sheet1')
sheet2012 = wb2012.worksheet('Sheet1')
sheet2013 = wb2013.worksheet('Sheet1')
sheet2014 = wb2014.worksheet('Sheet1')
sheet2015 = wb2015.worksheet('Sheet1')
sheet2016 = wb2016.worksheet('Sheet1')
sheet2017 = wb2017.worksheet('Sheet1')
sheet2018 = wb2018.worksheet('Sheet1')
sheet2019 = wb2019.worksheet('Sheet1')

#get data
alerts_2009 = pd.DataFrame(sheet2009.get_all_values())
alerts_2009.columns = alerts_2009.iloc[0]
alerts_2009 = alerts_2009.iloc[1:]
alerts_2010 = pd.DataFrame(sheet2010.get_all_values())
alerts_2010.columns = alerts_2010.iloc[0]
alerts_2010 = alerts_2010.iloc[1:]
alerts_2011 = pd.DataFrame(sheet2011.get_all_values())
alerts_2011.columns = alerts_2011.iloc[0]
alerts_2011 = alerts_2011.iloc[1:]
alerts_2012 = pd.DataFrame(sheet2012.get_all_values())
alerts_2012.columns = alerts_2012.iloc[0]
alerts_2012 = alerts_2012.iloc[1:]
alerts_2013 = pd.DataFrame(sheet2013.get_all_values())
alerts_2013.columns = alerts_2013.iloc[0]
alerts_2013 = alerts_2013.iloc[1:]
alerts_2014 = pd.DataFrame(sheet2014.get_all_values())
alerts_2014.columns = alerts_2014.iloc[0]
alerts_2014 = alerts_2014.iloc[1:]
alerts_2015 = pd.DataFrame(sheet2015.get_all_values())
alerts_2015.columns = alerts_2015.iloc[0]
alerts_2015 = alerts_2015.iloc[1:]
alerts_2016 = pd.DataFrame(sheet2016.get_all_values())
alerts_2016.columns = alerts_2016.iloc[0]
alerts_2016 = alerts_2016.iloc[1:]
alerts_2017 = pd.DataFrame(sheet2017.get_all_values())
alerts_2017.columns = alerts_2017.iloc[0]
alerts_2017 = alerts_2017.iloc[1:]
alerts_2018 = pd.DataFrame(sheet2018.get_all_values())
alerts_2018.columns = alerts_2018.iloc[0]
alerts_2018 = alerts_2018.iloc[1:]
alerts_2019 = pd.DataFrame(sheet2019.get_all_values())
alerts_2019.columns = alerts_2019.iloc[0]
alerts_2019 = alerts_2019.iloc[1:]
alerts_allyears = pd.concat([alerts_2009,alerts_2010, alerts_2011, alerts_2012, alerts_2013, alerts_2014,
                             alerts_2015, alerts_2016, alerts_2017, alerts_2018, alerts_2019], ignore_index=True)
```

In [0]:

```
train_dataframe = pd.concat([alerts_2009,alerts_2010, alerts_2011, alerts_2012, alerts_2013, alerts_2014,
                             alerts_2015, alerts_2016, alerts_2017, alerts_2019], ignore_index=True)

train_dataframe["AB_CO"] = np.where((train_dataframe.AB_CO == ''),'-2', train_dataframe.AB_CO)
train_dataframe["DI"] = np.where((train_dataframe.DI == ''),'-1', train_dataframe.DI)
train_dataframe = train_dataframe.astype('int64')

test_dataframe = alerts_2018
test_dataframe["AB_CO"] = np.where((test_dataframe.AB_CO == ''),'-2', test_dataframe.AB_CO)
test_dataframe["DI"] = np.where((test_dataframe.DI == ''),'-1', test_dataframe.DI)
test_dataframe = test_dataframe.astype('int64')
```

In [0]:

```
def lstm_model_X_layers(layers,num_features,dropout):
  """Create Neural Network with X (layers) lstm layers with random sizes between (25-100)"""
  model = Sequential()
  for i in range(0,len(layers)-1):
    model.add(LSTM(int(layers[i]), activation='relu',return_sequences=True, input_shape=(num_features,1)))
    model.add(Dropout(dropout))
  if len(layers) == 1 :
    model.add(LSTM(layers[-1], activation='relu', input_shape=(num_features,1)))
  else:
    model.add(LSTM(layers[-1], activation='relu'))
  model.add(Dropout(dropout))
  model.add(Dense(1))
  model.compile(optimizer='adam', loss='mse')
  print("Fit model with {} sizes and Dropout = {}".format(layers,dropout))
  return model
```

In [0]:

```
def draw_layer_size(X_train,X_test,Y_test,num_features,model_name,layers,attempts=10):
    """Get the model which minimizes the test error given the data and the number of layers in all attempts"""
    min_rmse = math.inf
    dropouts = [0.1]
    best_model = None
    for i in range(attempts):
      layers_array = [randint(25, 100) for p in range(0, layers)]
      for dropout in dropouts:
          model = lstm_model_X_layers(layers_array,num_features,dropout)
          model.fit(X_train, Y_train, epochs=500, verbose=0)
          Y_pred = model.predict(X_test, verbose=0)[:,0]
          new_rmse = np.sqrt(metrics.mean_squared_error(Y_test, Y_pred))
          print("Test RMSE = {}".format(new_rmse))
          if min_rmse > new_rmse and dropout !=0:
              best_model = model
              min_rmse = new_rmse
    print("Mejor RMSE: {}".format(min_rmse))
    plot_model(best_model, to_file='{}.png'.format(model_name), show_shapes=True, show_layer_names=True)
    return best_model
```

In [0]:

```
def print_prediction(real, pred, dias):
    """ Prints two plots: 1:a bar for real data and a bar for predicted data 2:the difference between real and pr
edicted data
    """
    #Print real and predicted data
    y = np.zeros((real.shape[0],3))
    y[:,0]= np.arange(real.shape[0])
    y[:,1] = real
    y[:,2] = pred
    df = pd.DataFrame(y, columns=["Días", "Real", "Predicción"]).head(dias)
    ax = df.plot(x="Días", y=["Real", "Predicción"], kind="bar")
    plt.show()


    #Print the difference
    y = np.zeros((real.shape[0],2))
    y[:,0] = np.arange(real.shape[0])
    y[:,1] = real - pred
    df = pd.DataFrame(y, columns=["Días", "Diferencia"]).head(dias)
    ax = df.plot(x="Días", y="Diferencia", kind="bar")
    plt.show()
```

In [0]:

```
def get_number_of_layers(X_train,Y_train,num_features):
  train_cost = []
  val_cost = []
  dots = np.linspace(1,5,5)
  for elem in dots:
      print("LAYER N° {}".format(elem))
      model = Sequential()
      for i in range(0,int(elem)-1):
        model.add(LSTM(32, activation='relu',return_sequences=True, input_shape=(num_features,1)))

      if elem == 1 :
        model.add(LSTM(32, activation='relu', input_shape=(num_features,1)))
      else:
        model.add(LSTM(32, activation='relu'))
      model.add(Dense(1))
      model.compile(optimizer='adam', loss='mse')
      history = model.fit(X_train, Y_train, nb_epoch=250, batch_size = 128, validation_split=0.2, shuffle=True, v
erbose=0)
      train_cost.append(np.sqrt(history.history['loss'][-1:]))
      val_cost.append(np.sqrt(history.history['val_loss'][-1:]))
  return train_cost,val_cost
```

Modelo nº 1: Nº de avisos en función del Mes y el día

Paso nº 1: Obtenemos variables

In [0]:

```
#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "SUCESO"]].groupby(["AÑO", "MES", "DIA"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Remove outliers
data_train = data_train.reset_index()
Y_train = data_train["SUCESO"]
X_train = data_train[["MES","DIA"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "SUCESO"]].groupby(["AÑO", "MES", "DIA"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Remove outliers
data_test = data_test.reset_index()
Y_test = data_test["SUCESO"]
X_test = data_test[["MES","DIA"]]
```

In [0]:

```
data_train.head(5)
```

Out[0]:

	AÑO	MES	DIA	SUCESO
0	2009	1	1	522
1	2009	1	2	329
2	2009	1	3	322
3	2009	1	4	314
4	2009	1	5	355

Parte nº 2: Transformamos los datos a array de numpy para poder utilizarlos en Keras

In [0]:

```
X_train = X_train.to_numpy()
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.to_numpy()
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

Parte nº 3: Buscamos a partir de que número de capas hay Overfitting

Es decir, buscamos un número de capas donde el coste de validación es mínimo

In [0]:

```
train_cost,val_cost = get_number_of_layers(X_train,Y_train,2)
```

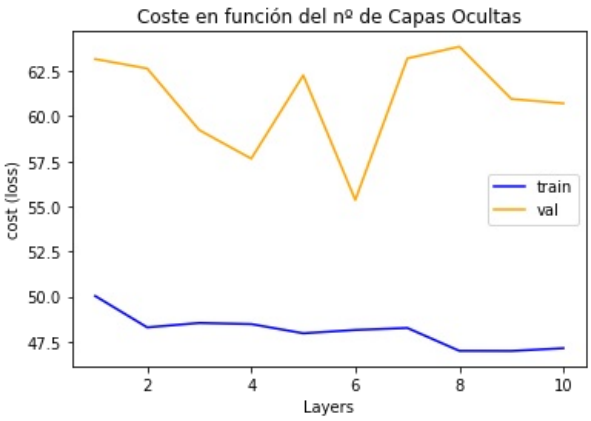
In [0]:

```
dots = np.linspace(1,10,10)
plt.plot(dots,train_cost, color="blue")
plt.plot(dots,val_cost, color="orange")
plt.legend(['train', 'val'])
plt.ylabel('cost (loss)')
plt.xlabel('Layers')
plt.title("Coste en función del n° de Capas Ocultas")
```

Out[0]:

```
Text(0.5, 1.0, 'Coste en función del n° de Capas Ocultas')
```

Out[0]:



Parte nº 4: Buscamos el mejor modelo con diferentes configuraciones de regularización

In [0]:

```
model = draw_layer_size(X_train,X_test,Y_test,2,"model_01",6,5)
```

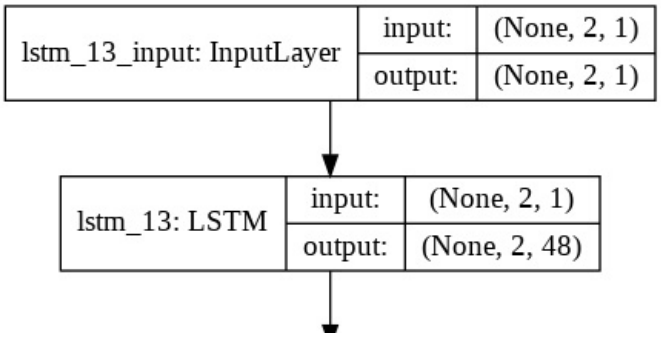
```
Fit model with [48, 81, 74, 64, 37, 85] sizes and Dropout = 0
Test RMSE = 57.20487237812411
Fit model with [48, 81, 74, 64, 37, 85] sizes and Dropout = 0.1
Test RMSE = 57.23402701319874
Fit model with [48, 81, 74, 64, 37, 85] sizes and Dropout = 0.2
Test RMSE = 74.2697102536277
Fit model with [48, 81, 74, 64, 37, 85] sizes and Dropout = 0.3
Test RMSE = 64.86294217989645
Fit model with [48, 35, 69, 92, 48, 91] sizes and Dropout = 0
Test RMSE = 62.41922250485678
Fit model with [48, 35, 69, 92, 48, 91] sizes and Dropout = 0.1
Test RMSE = 62.433989628265536
Fit model with [48, 35, 69, 92, 48, 91] sizes and Dropout = 0.2
Test RMSE = 57.41124861141172
Fit model with [48, 35, 69, 92, 48, 91] sizes and Dropout = 0.3
Test RMSE = 66.02760779374813
Fit model with [62, 88, 96, 97, 73, 64] sizes and Dropout = 0
Test RMSE = 55.53076175288674
Fit model with [62, 88, 96, 97, 73, 64] sizes and Dropout = 0.1
Test RMSE = 58.92704097218215
Fit model with [62, 88, 96, 97, 73, 64] sizes and Dropout = 0.2
Test RMSE = 60.91245399093475
Fit model with [62, 88, 96, 97, 73, 64] sizes and Dropout = 0.3
```

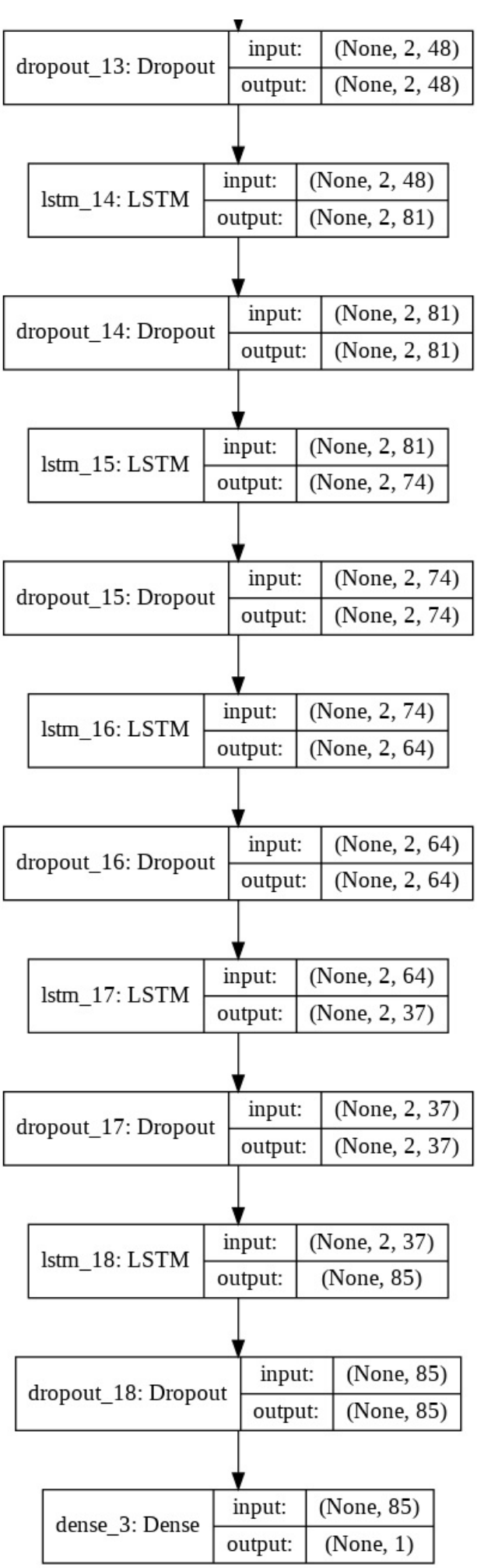
In [0]:

```
plot_model(model, to_file='model_01.png', show_shapes=True, show_layer_names=True)
```

```
Fit model with [48, 81, 74, 64, 37, 85] sizes and Dropout = 0.1
```

Out[0]:

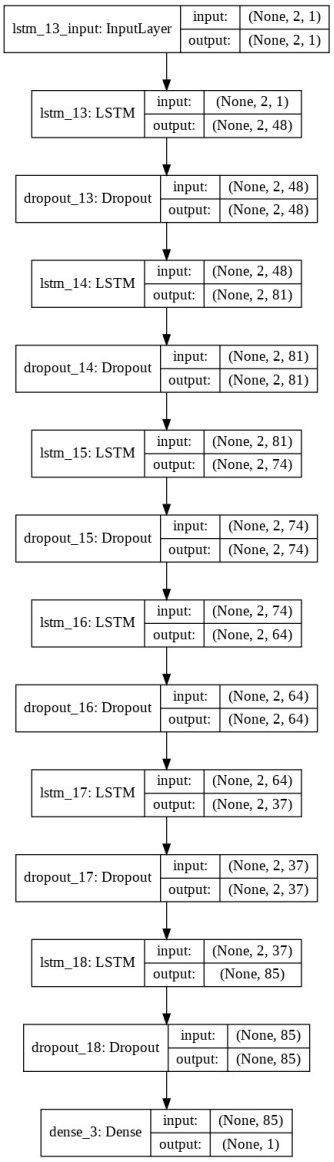



dropout_13: Dropout
input: (None, 2, 48)
output: (None, 2, 48)
lstm_14: LSTM
input: (None, 2, 48)
output: (None, 2, 81)
dropout_14: Dropout
input: (None, 2, 81)
output: (None, 2, 81)
lstm_15: LSTM
input: (None, 2, 81)
output: (None, 2, 74)
dropout_15: Dropout
input: (None, 2, 74)
output: (None, 2, 74)
lstm_16: LSTM
input: (None, 2, 74)
output: (None, 2, 64)
dropout_16: Dropout
input: (None, 2, 64)
output: (None, 2, 64)
lstm_17: LSTM
input: (None, 2, 64)
output: (None, 2, 37)
dropout_17: Dropout
input: (None, 2, 37)
output: (None, 2, 37)
lstm_18: LSTM
input: (None, 2, 37)
output: (None, 85)
dropout_18: Dropout
input: (None, 85)
output: (None, 85)
dense_3: Dense
input: (None, 85)
output: (None, 1)

In [0]:

```
Image(retina=True, filename='model_01.png')
```

Out[0]:



Parte nº5: Guardamos el modelo creado

In [0]:

```
pickle.dump(model, open('NN_cloud_01.sav', 'wb'))
files.download('NN_cloud_01.sav')
```

Modelo nº 2: Nº de avisos para un Mes, un día, un día de la semana¶

Paso nº 1: Obtenemos variables

In [0]:

```
#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA_CODIFICADO", "DIA", "SUCESO"]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA_CODIFICADO", "DIA", "SUCESO"]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO"]]
```

In [0]:

```
data_train.head(5)
```

Out[0]:

	AÑO	MES	DIA	DIA_CODIFICADO	SUCESO
0	2009	1	1	3	522
1	2009	1	2	4	329
2	2009	1	3	5	322
3	2009	1	4	6	314
4	2009	1	5	0	355

Parte nº 2: Transformamos los datos a array de numpy para poder utilizarlos en Keras

In [0]:

```
X_train = X_train.to_numpy()
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.to_numpy()
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

Parte nº 3: Buscamos a partir de que número de capas hay Overfitting

Es decir, buscamos un número de capas donde el coste de validación es mínimo

In [0]:

```
train_cost,val_cost = get_number_of_layers(X_train,Y_train,3)
```

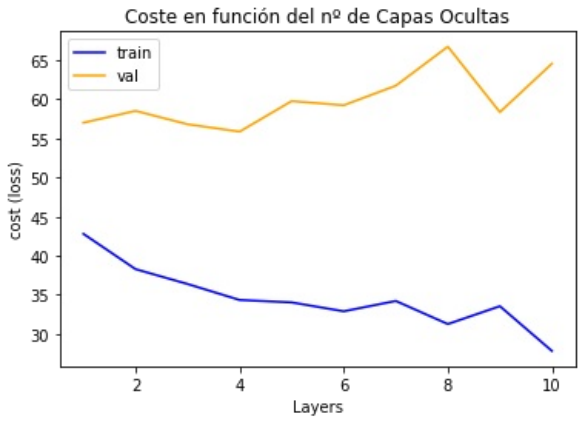
In [0]:

```
dots = np.linspace(1,10,10)
plt.plot(dots,train_cost, color="blue")
plt.plot(dots,val_cost, color="orange")
plt.legend(['train', 'val'])
plt.ylabel('cost (loss)')
plt.xlabel('Layers')
plt.title("Coste en función del n° de Capas Ocultas")
```

Out[0]:

```
Text(0.5, 1.0, 'Coste en función del n° de Capas Ocultas')
```

Out[0]:



Parte nº 4: Buscamos el mejor modelo con diferentes configuraciones de regularización

In [0]:

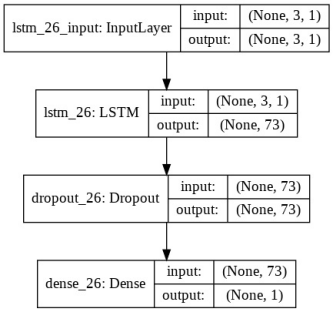
```
model = draw_layer_size(X_train,X_test,Y_test,3,"model_02",1,10)
```

```
Fit model with [55] sizes and Dropout = 0.1
Test RMSE = 47.124120182742054
Fit model with [55] sizes and Dropout = 0.2
Test RMSE = 53.4624902871773
Fit model with [55] sizes and Dropout = 0.3
Test RMSE = 54.005841075684025
Fit model with [56] sizes and Dropout = 0.1
Test RMSE = 53.66294303759715
Fit model with [56] sizes and Dropout = 0.2
Test RMSE = 50.40746388212477
Fit model with [56] sizes and Dropout = 0.3
Test RMSE = 60.87800945558532
Fit model with [58] sizes and Dropout = 0.1
Test RMSE = 49.24539341433855
Fit model with [58] sizes and Dropout = 0.2
Test RMSE = 55.763371291878116
Fit model with [58] sizes and Dropout = 0.3
Test RMSE = 57.39919090877928
Fit model with [83] sizes and Dropout = 0.1
Test RMSE = 47.31966495801113
Fit model with [83] sizes and Dropout = 0.2
Test RMSE = 50.99635760988677
Fit model with [83] sizes and Dropout = 0.3
Test RMSE = 51.49574830936919
Fit model with [34] sizes and Dropout = 0.1
Test RMSE = 54.045301978148586
Fit model with [34] sizes and Dropout = 0.2
Test RMSE = 56.229978619617455
Fit model with [34] sizes and Dropout = 0.3
Test RMSE = 59.752848791475955
Fit model with [67] sizes and Dropout = 0.1
Test RMSE = 49.02043761816737
Fit model with [67] sizes and Dropout = 0.2
Test RMSE = 51.5275129835329
Fit model with [67] sizes and Dropout = 0.3
Test RMSE = 57.13862461410709
Fit model with [53] sizes and Dropout = 0.1
Test RMSE = 58.40781029963219
Fit model with [53] sizes and Dropout = 0.2
Test RMSE = 47.20334139084302
Fit model with [53] sizes and Dropout = 0.3
Test RMSE = 55.8249342884247 14
Fit model with [73] sizes and Dropout = 0.1
Test RMSE = 51.652820953152265
Fit model with [73] sizes and Dropout = 0.2
Test RMSE = 48.448419014146964
Fit model with [73] sizes and Dropout = 0.3
Test RMSE = 45.192252688904496
Fit model with [40] sizes and Dropout = 0.1
Test RMSE = 53.75480920103496
Fit model with [40] sizes and Dropout = 0.2
Test RMSE = 53.19972852728811
Fit model with [40] sizes and Dropout = 0.3
Test RMSE = 53.46532553490279
Fit model with [93] sizes and Dropout = 0.1
Test RMSE = 51.06215963600691
Fit model with [93] sizes and Dropout = 0.2
Test RMSE = 47.9948799471379
Fit model with [93] sizes and Dropout = 0.3
Test RMSE = 51.41987380256345
Mejor RMSE: 45.192252688904496
```

In [0]:

```
Image(retina=True, filename='model_02.png')
```

Out[0]:



Parte nº5: Guardamos el modelo creado

In [0]:

```
pickle.dump(model, open('NN_cloud_02.sav', 'wb'))
files.download('NN_cloud_02.sav')
```

Modelo nº 3: Nº de avisos para un Mes, día, dia de la semana y base

Paso nº 1: Obtenemos variables

In [0]:

```
#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO","BA_CO", "SUCESO"]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO","BA_CO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO","BA_CO"]]

data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO","BA_CO", "SUCESO"]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO","BA_CO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO","BA_CO"]]
```

In [0]:

```
data_train.head(5)
```

Out[0]:

	AÑO	MES	DIA	DIA_CODIFICADO	BA_CO	SUCESO
0	2009	1	1	3	-1	28
1	2009	1	1	3	3	31
2	2009	1	1	3	4	18
3	2009	1	1	3	5	22
4	2009	1	1	3	6	35

Parte nº 2: Transformamos los datos a array de numpy para poder utilizarlos en Keras

In [0]:

```
X_train = X_train.to_numpy()
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.to_numpy()
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

Parte nº 3: Buscamos a partir de que número de capas hay Overfitting

Es decir, buscamos un número de capas a partir del cual el coste de validación se distancia del coste de train

In [0]:

```
train_cost,val_cost = get_number_of_layers(X_train,Y_train,4)
```

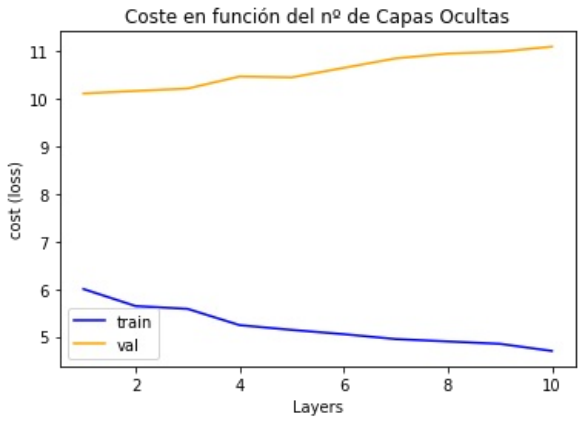
In [0]:

```
dots = np.linspace(1,10,10)
plt.plot(dots,train_cost, color="blue")
plt.plot(dots,val_cost, color="orange")
plt.legend(['train', 'val'])
plt.ylabel('cost (loss)')
plt.xlabel('Layers')
plt.title("Coste en función del n° de Capas Ocultas")
```

Out[0]:

```
Text(0.5, 1.0, 'Coste en función del n° de Capas Ocultas')
```

Out[0]:



Parte nº 4: Buscamos el mejor modelo con diferentes configuraciones de regularización

In [0]:

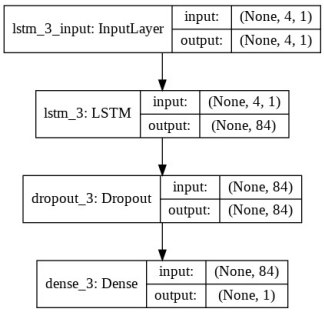
```
model = draw_layer_size(X_train,X_test,Y_test,4,"model_03",1,2)
```

```
Fit model with [84] sizes and Dropout = 0
Test RMSE = 8.559060488614975
Fit model with [84] sizes and Dropout = 0.1
Test RMSE = 8.234864262128973
Fit model with [84] sizes and Dropout = 0.2
Test RMSE = 8.511154171984892
Fit model with [84] sizes and Dropout = 0.3
Test RMSE = 8.602741240806912
Fit model with [93] sizes and Dropout = 0
Test RMSE = 8.635505435643596
Fit model with [93] sizes and Dropout = 0.1
```

In [0]:

```
Image(retina=True, filename='model_03.png')
```

Out[0]:



Parte nº5: Guardamos el modelo creado

In [0]:

```
pickle.dump(model, open('NN_cloud_03.sav', 'wb'))
files.download('NN_cloud_03.sav')
```

Modelo nº 4: Nº de avisos para un Mes, día, día de la semana y distrito

Paso nº 1: Obtenemos variables

In [0]:

```
#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO","DI", "SUCESO"]].groupby(["AÑO", "MES", "DIA"
, "DIA_CODIFICADO","DI"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO","DI"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO","DI", "SUCESO"]].groupby(["AÑO", "MES", "DIA",
"DIA_CODIFICADO","DI"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO","DI"]]
```

In [0]:

```
data_train.head(5)
```

Out[0]:

	AÑO	MES	DIA	DIA_CODIFICADO	DI	SUCESO
0	2009	1	1	3	-1	19
1	2009	1	1	3	0	3
2	2009	1	1	3	2	17
3	2009	1	1	3	3	9
4	2009	1	1	3	4	24

Parte nº 2: Transformamos los datos a array de numpy para poder utilizarlos en Keras

In [0]:

```
X_train = X_train.to_numpy()
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.to_numpy()
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

Parte nº 3: Buscamos a partir de que número de capas hay Overfitting

Es decir, buscamos un número de capas a partir del cual el coste de validación se distancia del coste de train

In [0]:

```
train_cost,val_cost = get_number_of_layers(X_train,Y_train,4)
```

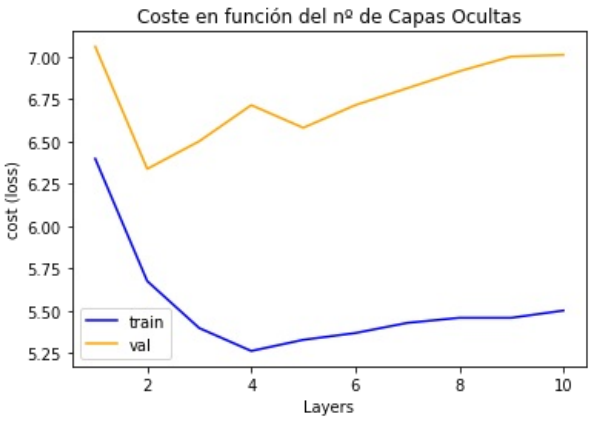
In [0]:

```
dots = np.linspace(1,10,10)
plt.plot(dots,train_cost, color="blue")
plt.plot(dots,val_cost, color="orange")
plt.legend(['train', 'val'])
plt.ylabel('cost (loss)')
plt.xlabel('Layers')
plt.title("Coste en función del nº de Capas Ocultas")
```

Out[0]:

```
Text(0.5, 1.0, 'Coste en función del nº de Capas Ocultas')
```

Out[0]:



Parte nº 4: Buscamos el mejor modelo con diferentes configuraciones de regularización

In [0]:

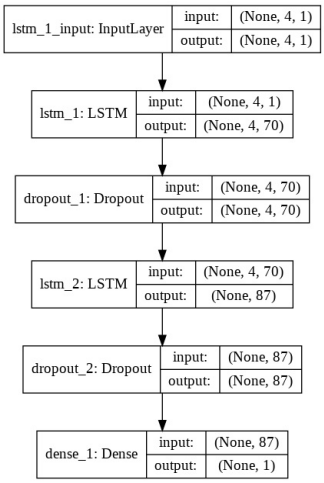
```
model = draw_layer_size(X_train,X_test,Y_test,4,"model_04",2,1)
```

```
Fit model with [70, 87] sizes and Dropout = 0
Test RMSE = 6.983352615798535
Fit model with [70, 87] sizes and Dropout = 0.1
Test RMSE = 6.520498589419127
Fit model with [70, 87] sizes and Dropout = 0.2
```

In [0]:

```
Image(retina=True, filename='model_04.png')
```

Out[0]:



Parte nº5: Guardamos el modelo creado

In [0]:

```
pickle.dump(model, open('NN_cloud_04.sav', 'wb'))
files.download('NN_cloud_04.sav')
```

Modelo nº 5 : Nº de avisos para un Mes, día, día de la semana y turno

Paso nº 1: Obtenemos variables¶

In [0]:

```
#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "T_CO", "SUCESO"]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "T_CO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO", "T_CO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "T_CO", "SUCESO"]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "T_CO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO", "T_CO"]]
```

In [0]:

```
data_train.head(5)
```

Out[0]:

	AÑO	MES	DIA	DIA_CODIFICADO	T_CO	SUCESO
0	2009	1	1	3	0	190
1	2009	1	1	3	1	116
2	2009	1	1	3	2	216
3	2009	1	2	4	0	147
4	2009	1	2	4	1	135

Parte nº 2: Transformamos los datos a array de numpy para poder utilizarlos en Keras

In [0]:

```
X_train = X_train.to_numpy()
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.to_numpy()
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

Parte nº 3: Buscamos a partir de que número de capas hay Overfitting

Es decir, buscamos un número de capas a partir del cual el coste de validación se distancia del coste de train

In [0]:

```
train_cost,val_cost = get_number_of_layers(X_train,Y_train,4)
```

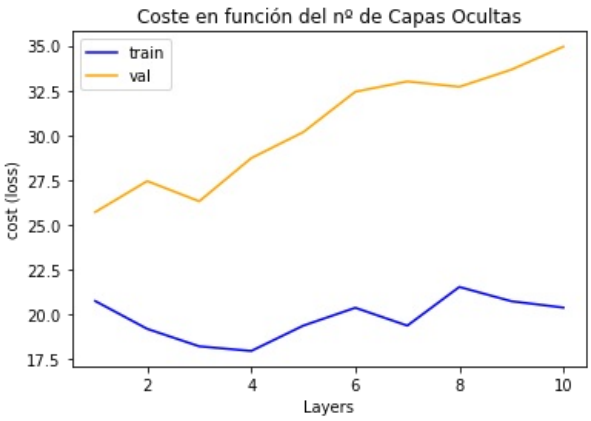
In [0]:

```
dots = np.linspace(1,10,10)
plt.plot(dots,train_cost, color="blue")
plt.plot(dots,val_cost, color="orange")
plt.legend(['train', 'val'])
plt.ylabel('cost (loss)')
plt.xlabel('Layers')
plt.title("Coste en función del n° de Capas Ocultas")
```

Out[0]:

```
Text(0.5, 1.0, 'Coste en función del n° de Capas Ocultas')
```

Out[0]:



Parte nº 4: Buscamos el mejor modelo con diferentes configuraciones de regularización

In [0]:

```
model = draw_layer_size(X_train,X_test,Y_test,4,"model_05",1,2)
```

```
Fit model with [35] sizes and Dropout = 0
Test RMSE = 24.992391995204336
Fit model with [35] sizes and Dropout = 0.1
Test RMSE = 25.641933475015957
Fit model with [35] sizes and Dropout = 0.2
Test RMSE = 27.159192132652525
Fit model with [35] sizes and Dropout = 0.3
Test RMSE = 25.68113871035248
Fit model with [99] sizes and Dropout = 0
Test RMSE = 26.252485980968142
Fit model with [99] sizes and Dropout = 0.1
Test RMSE = 23.92129252909358
Fit model with [99] sizes and Dropout = 0.2
Test RMSE = 26.246679349738862
Fit model with [99] sizes and Dropout = 0.3
Test RMSE = 25.66963928628731
Mejor RMSE: 23.92129252909358
```

In [0]:

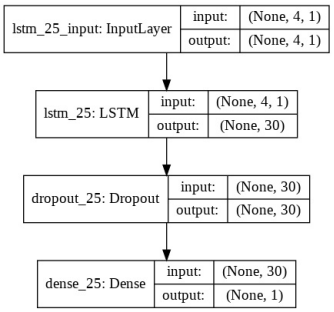
```
model = draw_layer_size(X_train,X_test,Y_test,4,"model_05",1,5)
```

```
Fit model with [73] sizes and Dropout = 0
Test RMSE = 24.981389824775544
Fit model with [73] sizes and Dropout = 0.1
Test RMSE = 25.661059778433312
Fit model with [73] sizes and Dropout = 0.2
Test RMSE = 24.73954290462975
Fit model with [73] sizes and Dropout = 0.3
Test RMSE = 25.119617575850334
Fit model with [80] sizes and Dropout = 0
Test RMSE = 25.08634305079478
Fit model with [80] sizes and Dropout = 0.1
Test RMSE = 26.532285620655507
Fit model with [80] sizes and Dropout = 0.2
Test RMSE = 25.317372378026175
Fit model with [80] sizes and Dropout = 0.3
Test RMSE = 24.88671535844075
Fit model with [73] sizes and Dropout = 0
Test RMSE = 25.54039876801032
Fit model with [73] sizes and Dropout = 0.1
Test RMSE = 24.12178185113044
Fit model with [73] sizes and Dropout = 0.2
Test RMSE = 24.811766818457254
Fit model with [73] sizes and Dropout = 0.3
Test RMSE = 25.743896459795288
Fit model with [30] sizes and Dropout = 0
Test RMSE = 23.797005361207006
Fit model with [30] sizes and Dropout = 0.1
Test RMSE = 23.93820839206829
Fit model with [30] sizes and Dropout = 0.2
Test RMSE = 26.113434435425823
Fit model with [30] sizes and Dropout = 0.3
Test RMSE = 25.933375673184237
Fit model with [27] sizes and Dropout = 0
Test RMSE = 24.3173524507971
Fit model with [27] sizes and Dropout = 0.1
Test RMSE = 25.616699630560138
Fit model with [27] sizes and Dropout = 0.2
Test RMSE = 25.719485237510526
Fit model with [27] sizes and Dropout = 0.3
Test RMSE = 26.06399171605239
Mejor RMSE: 23.93820839206829
```

In [0]:

```
Image(retina=True, filename='model_05.png')
```

Out[0]:



Parte nº5: Guardamos el modelo creado

In [0]:

```
pickle.dump(model, open('NN_cloud_05.sav', 'wb'))
files.download('NN_cloud_05.sav')
```

Modelo nº 6: Nº de avisos para un Mes, día, dia de la semana, base y turno

Paso nº 1: Obtenemos variables

In [0]:

```
#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BA_CO", "T_CO", "SUCESO"]].groupby(["AÑO",
"MES", "DIA", "DIA_CODIFICADO", "BA_CO", "T_CO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO", "BA_CO", "T_CO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BA_CO", "T_CO", "SUCESO"]].groupby(["AÑO", "M
ES", "DIA", "DIA_CODIFICADO", "BA_CO", "T_CO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO", "BA_CO", "T_CO"]]
```

In [0]:

```
data_train.head(5)
```

Out[0]:

	AÑO	MES	DIA	DIA_CODIFICADO	BA_CO	T_CO	SUCESO
0	2009	1	1	3	-1	0	8
1	2009	1	1	3	-1	1	6
2	2009	1	1	3	-1	2	14
3	2009	1	1	3	1	0	19
4	2009	1	1	3	1	1	13

Parte nº 2: Transformamos los datos a array de numpy para poder utilizarlos en Keras

In [0]:

```
X_train = X_train.to_numpy()
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.to_numpy()
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

Parte nº 3: Buscamos a partir de que número de capas hay Overfitting

Es decir, buscamos un número de capas a partir del cual el coste de validación se distancia del coste de train

In [0]:

```
train_cost,val_cost = get_number_of_layers(X_train,Y_train,5)
```

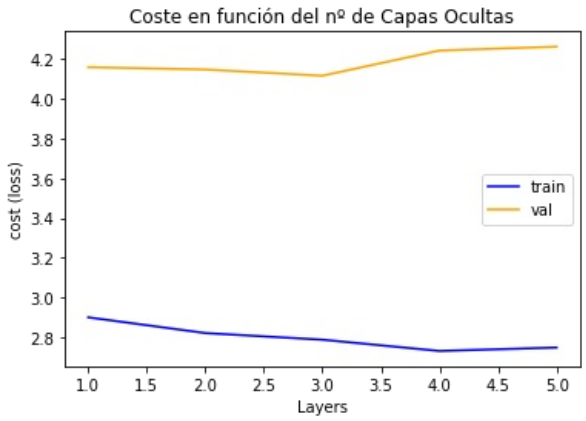
In [0]:

```
dots = np.linspace(1,5,5)
plt.plot(dots,train_cost, color="blue")
plt.plot(dots,val_cost, color="orange")
plt.legend(['train', 'val'])
plt.ylabel('cost (loss)')
plt.xlabel('Layers')
plt.title("Coste en función del n° de Capas Ocultas")
```

Out[0]:

```
Text(0.5, 1.0, 'Coste en función del n° de Capas Ocultas')
```

Out[0]:



Parte nº 4: Buscamos el mejor modelo con diferentes configuraciones de regularización

In [0]:

```
model = draw_layer_size(X_train,X_test,Y_test,5,"model_06",3,1)
```

```
Fit model with [26, 36, 54] sizes and Dropout = 0.1
Test RMSE = 4.165144041585944
Mejor RMSE: 4.165144041585944
```

In [0]:

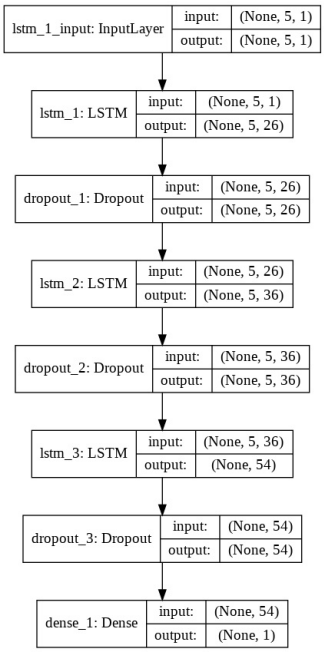
```
model = draw_layer_size(X_train,X_test,Y_test,5,"model_06",3,1)
```

```
Fit model with [90, 60, 95] sizes and Dropout = 0.1
```

In [0]:

```
Image(retina=True, filename='model_06.png')
```

Out[0]:



Parte nº5: Guardamos el modelo creado

In [0]:

```
pickle.dump(model, open('NN_cloud_06.sav', 'wb'))
files.download('NN_cloud_06.sav')
```

Modelo nº 11: nº de Avisos para un mes, día , día de la semana, distrito, turno y dispositivo

Paso nº 1: Obtenemos variables

In [0]:

```
#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DI", "T_CO", "AB_CO", "SUCESO"]].groupby(["
AÑO", "MES", "DIA", "DIA_CODIFICADO", "DI", "T_CO", "AB_CO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO", "DI", "T_CO", "AB_CO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DI", "T_CO", "AB_CO", "SUCESO"]].groupby(["AÑ
O", "MES", "DIA", "DIA_CODIFICADO", "DI", "T_CO", "AB_CO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO", "DI", "T_CO", "AB_CO"]]
```

In [0]:

```
data_train.head(5)
```

Out[0]:

	AÑO	MES	DIA	DIA_CODIFICADO	DI	T_CO	AB_CO	SUCESO
0	2009	1	1	3	-1	0	0	2
1	2009	1	1	3	-1	0	1	1
2	2009	1	1	3	-1	2	0	6
3	2009	1	1	3	-1	2	1	4
4	2009	1	1	3	-1	2	2	5

Parte nº 2: Transformamos los datos a array de numpy para poder utilizarlos en Keras

In [0]:

```
X_train = X_train.to_numpy()
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.to_numpy()
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

Parte nº 3: Buscamos a partir de que número de capas hay Overfitting

Es decir, buscamos un número de capas a partir del cual el coste de validación se distancia del coste de train

In [0]:

```
train_cost,val_cost = get_number_of_layers(X_train,Y_train,6)
```

```
LAYER N° 1.0
LAYER N° 2.0
LAYER N° 3.0
LAYER N° 4.0
LAYER N° 5.0
```

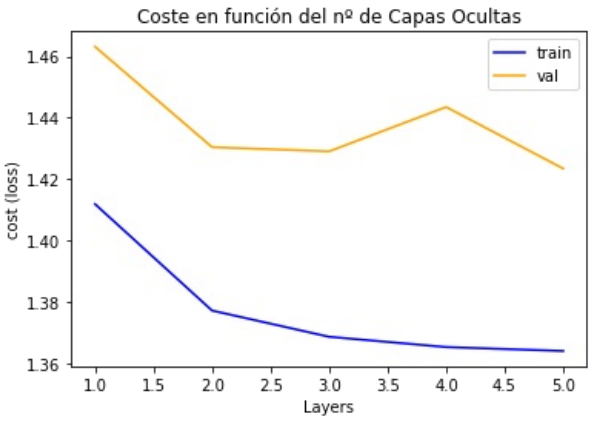
In [0]:

```
dots = np.linspace(1,5,5)
plt.plot(dots,train_cost, color="blue")
plt.plot(dots,val_cost, color="orange")
plt.legend(['train', 'val'])
plt.ylabel('cost (loss)')
plt.xlabel('Layers')
plt.title("Coste en función del n° de Capas Ocultas")
```

Out[0]:

```
Text(0.5, 1.0, 'Coste en función del n° de Capas Ocultas')
```

Out[0]:



In [0]:

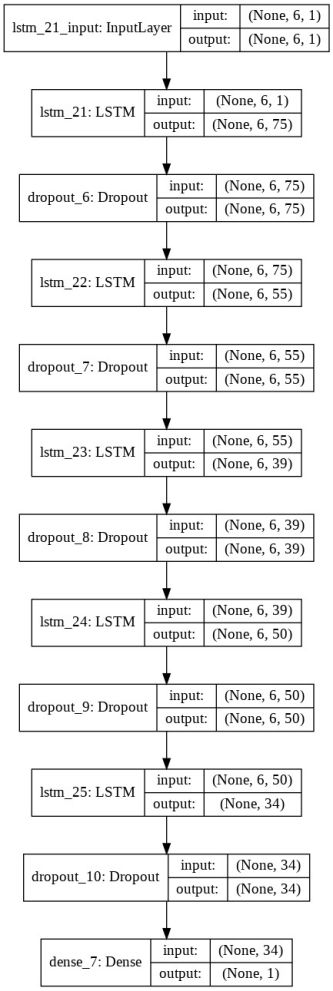
```
model = draw_layer_size(X_train,X_test,Y_test,6,"model_11",5,1)
```

```
Fit model with [75, 55, 39, 50, 34] sizes and Dropout = 0.1
Test RMSE = 1.4734456140732768
Mejor RMSE: 1.4734456140732768
```

In [0]:

```
Image(retina=True, filename='model_11.png')
```

Out[0]:



In [0]:

```
pickle.dump(model, open('NN_cloud_11.sav', 'wb'))
files.download('NN_cloud_11.sav')
```

Modelo nº 13: Nº de avisos para un mes, día , día de la semana, distrito, turno

Paso nº 1: Obtenemos variables

In [0]:

```
#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DI", "T_CO", "SUCESO"]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DI", "T_CO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO", "DI", "T_CO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DI", "T_CO", "SUCESO"]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DI", "T_CO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO", "DI", "T_CO"]]
```

In [0]:

```
data_train.head(5)
```

Out[0]:

	AÑO	MES	DIA	DIA_CODIFICADO	DI	T_CO	SUCESO
0	2009	1	1	3	-1	0	3
1	2009	1	1	3	-1	2	16
2	2009	1	1	3	0	1	3
3	2009	1	1	3	2	0	6
4	2009	1	1	3	2	1	3

Parte nº 2: Transformamos los datos a array de numpy para poder utilizarlos en Keras

In [0]:

```
X_train = X_train.to_numpy()
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.to_numpy()
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

Parte nº 3: Buscamos a partir de que número de capas hay Overfitting

Es decir, buscamos un número de capas a partir del cual el coste de validación se distancia del coste de train

In [0]:

```
train_cost,val_cost = get_number_of_layers(X_train,Y_train,5)
```

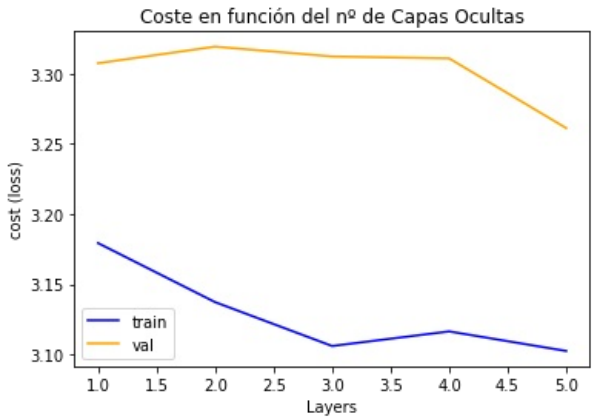
In [0]:

```
dots = np.linspace(1,5,5)
plt.plot(dots,train_cost, color="blue")
plt.plot(dots,val_cost, color="orange")
plt.legend(['train', 'val'])
plt.ylabel('cost (loss)')
plt.xlabel('Layers')
plt.title("Coste en función del nº de Capas Ocultas")
```

Out[0]:

```
Text(0.5, 1.0, 'Coste en función del nº de Capas Ocultas')
```

Out[0]:



Parte nº 4: Buscamos el mejor modelo con diferentes configuraciones de regularización

In [0]:

```
model = draw_layer_size(X_train,X_test,Y_test,5,"model_13",5,1)
```

```
Fit model with [33, 85, 78, 73, 52] sizes and Dropout = 0.1
Test RMSE = 3.559183655222119
Mejor RMSE: 3.559183655222119
```

In [0]:

```
model = draw_layer_size(X_train,X_test,Y_test,5,"model_13",5,1)
```

```
Fit model with [54, 46, 97, 47, 83] sizes and Dropout = 0.1
Test RMSE = 3.427683005979259
Mejor RMSE: 3.427683005979259
```

In [0]:

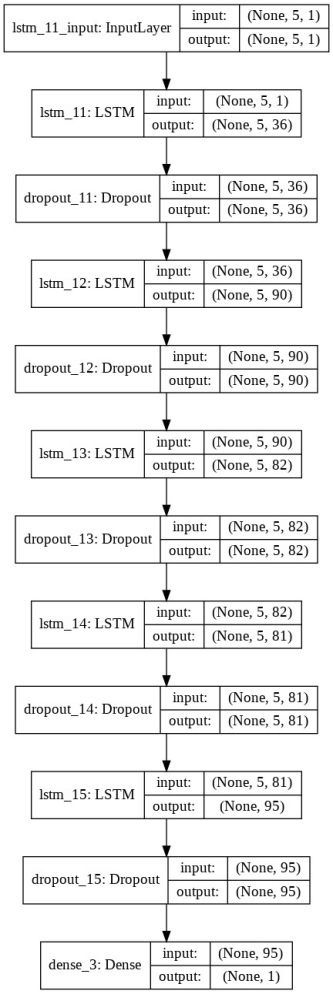
```
model = draw_layer_size(X_train,X_test,Y_test,5,"model_13",5,1)
```

```
Fit model with [36, 90, 82, 81, 95] sizes and Dropout = 0.1
Test RMSE = 3.38514239062216
```

In [0]:

```
Image(retina=True, filename='model_13.png')
```

Out[0]:



Parte nº5: Guardamos el modelo creado

In [0]:

```
pickle.dump(model, open('NN_cloud_13.sav', 'wb'))
files.download('NN_cloud_13.sav')
```

Redes Neuronales Fase II (II) SAMUR - Protección Civil

Propósito del documento

El objetivo de este documento es la creación de modelos utilizando como técnica de aprendizaje Redes neuronales, para predecir el número de avisos o activaciones que recibirá SAMUR-PC para un determinado día y unas determinadas cirucnstancias en la ciudad de Madrid. Este documento contiene el código desarrollado para la segunda Fase de Redes Neuronales, es decir, el desarrollo se realiza utilizando la CPU de Google Collaboratory.

Esta segunda fase esta formada por dos documentos:

- Redes Neuronales Fase II
- Redes Neuronales Fase II (II)

Para cada uno de los modelos se desarrollarán los siguientes pasos:

1. Obtención de variables
2. Transformación de datos para prepararlos para modelos Deep Learning utilizando la librería Keras
3. Búsqueda del mejor número de capas del modelo
4. Búsqueda del mejor número de nodos en cada capa del modelo
5. Exportación del modelo

Librerías

In [0]:

```
#general libraries
import pandas as pd
from matplotlib import pyplot as plt
import numpy as np
from math import sqrt
from scipy import stats
import pickle
import torch
import math
import warnings
import timeit
import os
from random import randint
from IPython.display import Image
warnings.filterwarnings("ignore")
from numpy.random import RandomState

#google libraries
from google.colab import auth
import gspread
from oauth2client.client import GoogleCredentials
from google.colab import drive
from google.colab import files

#deep learning libraries
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.utils.vis_utils import plot_model
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn import metrics
from keras.layers import Flatten
from keras.layers import LSTM
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
tf.random.set_seed(2)
```

In [0]:

```
auth.authenticate_user()
gc = gspread.authorize(GoogleCredentials.get_application_default())
```

In [0]:

```
#create woorkbooks
wb2009 = gc.open_by_url('https://docs.google.com/spreadsheets/d/1uRXslPOaa99Jj9xZeaXIZk546lgl7BbozpCTzIg-5Xk/edit
#gid=340244330')
wb2010 = gc.open_by_url('https://docs.google.com/spreadsheets/d/12zcTH0VL57AptiXjpXOHKXehGpIu2W7vKhrkw-liQFs/edit
#gid=825218595')
wb2011 = gc.open_by_url('https://docs.google.com/spreadsheets/d/1az4qe5Pa1-dNHIZIlkt8cTDk0itJRsl5Qs5RYrJNlo0/edit
#gid=50080999')
wb2012 = gc.open_by_url('https://docs.google.com/spreadsheets/d/1kH5h9NMtgOHYizGpAMcyTCwsuflf-CRrGmaQwc1uY0c/edit
#gid=1733288549')
wb2013 = gc.open_by_url('https://docs.google.com/spreadsheets/d/1Ic1PBb8mJAHogjvnCJCVcyydcEtWPvTVWgMh_xxfvXI/edit
#gid=1474377252')
wb2014 = gc.open_by_url('https://docs.google.com/spreadsheets/d/1fp5hHXUcb6WR4nKSh5K1rnvb0tV50PHKPlTVau11rIQ/edit
#gid=1344814184')
wb2015 = gc.open_by_url('https://docs.google.com/spreadsheets/d/1FawfEvToC2i3-FlxAuXDoiteonvdKNpd7qPH8OtGzLA/edit
#gid=158132632')
wb2016 = gc.open_by_url('https://docs.google.com/spreadsheets/d/1gh3FSz32hhoaBm6Ez0xrkN3iYIV-xN2XvlJNJqIMrPY/edit
#gid=1268380804')
wb2017 = gc.open_by_url('https://docs.google.com/spreadsheets/d/1bNF2LsY75tUbhfZm448zG7dUxzehHtP2KCy0GD1biPE/edit
#gid=1309243119')
wb2018 = gc.open_by_url('https://docs.google.com/spreadsheets/d/1qQszEZUi-URftAtBpyruqvXQKFU06UyNz4DmKkewmn8/edit
#gid=267435973')
wb2019 = gc.open_by_url('https://docs.google.com/spreadsheets/d/1T5dWfsEbUO2lxJzRyNyuIIPbCrjZbt3lR6OsVXr0yN8/edit
#gid=2122785558')

#get sheet
sheet2009 = wb2009.worksheet('Sheet1')
sheet2010 = wb2010.worksheet('Sheet1')
sheet2011 = wb2011.worksheet('Sheet1')
sheet2012 = wb2012.worksheet('Sheet1')
sheet2013 = wb2013.worksheet('Sheet1')
sheet2014 = wb2014.worksheet('Sheet1')
sheet2015 = wb2015.worksheet('Sheet1')
sheet2016 = wb2016.worksheet('Sheet1')
sheet2017 = wb2017.worksheet('Sheet1')
sheet2018 = wb2018.worksheet('Sheet1')
sheet2019 = wb2019.worksheet('Sheet1')

#get data
alerts_2009 = pd.DataFrame(sheet2009.get_all_values())
alerts_2009.columns = alerts_2009.iloc[0]
alerts_2009 = alerts_2009.iloc[1:]
alerts_2010 = pd.DataFrame(sheet2010.get_all_values())
alerts_2010.columns = alerts_2010.iloc[0]
alerts_2010 = alerts_2010.iloc[1:]
alerts_2011 = pd.DataFrame(sheet2011.get_all_values())
alerts_2011.columns = alerts_2011.iloc[0]
alerts_2011 = alerts_2011.iloc[1:]
alerts_2012 = pd.DataFrame(sheet2012.get_all_values())
alerts_2012.columns = alerts_2012.iloc[0]
alerts_2012 = alerts_2012.iloc[1:]
alerts_2013 = pd.DataFrame(sheet2013.get_all_values())
alerts_2013.columns = alerts_2013.iloc[0]
alerts_2013 = alerts_2013.iloc[1:]
alerts_2014 = pd.DataFrame(sheet2014.get_all_values())
alerts_2014.columns = alerts_2014.iloc[0]
alerts_2014 = alerts_2014.iloc[1:]
alerts_2015 = pd.DataFrame(sheet2015.get_all_values())
alerts_2015.columns = alerts_2015.iloc[0]
alerts_2015 = alerts_2015.iloc[1:]
alerts_2016 = pd.DataFrame(sheet2016.get_all_values())
alerts_2016.columns = alerts_2016.iloc[0]
alerts_2016 = alerts_2016.iloc[1:]
alerts_2017 = pd.DataFrame(sheet2017.get_all_values())
alerts_2017.columns = alerts_2017.iloc[0]
alerts_2017 = alerts_2017.iloc[1:]
alerts_2018 = pd.DataFrame(sheet2018.get_all_values())
alerts_2018.columns = alerts_2018.iloc[0]
alerts_2018 = alerts_2018.iloc[1:]
alerts_2019 = pd.DataFrame(sheet2019.get_all_values())
alerts_2019.columns = alerts_2019.iloc[0]
alerts_2019 = alerts_2019.iloc[1:]
alerts_allyears = pd.concat([alerts_2009,alerts_2010, alerts_2011, alerts_2012, alerts_2013, alerts_2014,
                             alerts_2015, alerts_2016, alerts_2017, alerts_2018, alerts_2019], ignore_index=True)
```

In [0]:

```
train_dataframe = pd.concat([alerts_2009,alerts_2010, alerts_2011, alerts_2012, alerts_2013, alerts_2014,
                             alerts_2015, alerts_2016, alerts_2017, alerts_2019], ignore_index=True)

train_dataframe["AB_CO"] = np.where((train_dataframe.AB_CO == ''),'-2', train_dataframe.AB_CO)
train_dataframe["DI"] = np.where((train_dataframe.DI == ''),'-1', train_dataframe.DI)
train_dataframe = train_dataframe.astype('int64')

test_dataframe = alerts_2018
test_dataframe["AB_CO"] = np.where((test_dataframe.AB_CO == ''),'-2', test_dataframe.AB_CO)
test_dataframe["DI"] = np.where((test_dataframe.DI == ''),'-1', test_dataframe.DI)
test_dataframe = test_dataframe.astype('int64')
```

In [0]:

```
def lstm_model_X_layers(layers,num_features,dropout):
  """Create Neural Network with X (layers) lstm layers with random sizes between (25-100)"""
  model = Sequential()
  for i in range(0,len(layers)-1):
    model.add(LSTM(int(layers[i]), activation='relu',return_sequences=True, input_shape=(num_features,1)))
    model.add(Dropout(dropout))
  if len(layers) == 1 :
    model.add(LSTM(layers[-1], activation='relu', input_shape=(num_features,1)))
  else:
    model.add(LSTM(layers[-1], activation='relu'))
  model.add(Dropout(dropout))
  model.add(Dense(1))
  model.compile(optimizer='adam', loss='mse')
  print("Fit model with {} sizes and Dropout = {}".format(layers,dropout))
  return model
```

In [0]:

```
def draw_layer_size(X_train,X_test,Y_test,num_features,model_name,layers,attempts=10):
    """Get the model which minimizes the test error given the data and the number of layers in all attempts"""
    min_rmse = math.inf
    dropouts = [0,0.1,0.2,0.3]
    best_model = None
    for i in range(attempts):
      layers_array = [randint(25, 100) for p in range(0, layers)]
      for dropout in dropouts:
          model = lstm_model_X_layers(layers_array,num_features,dropout)
          model.fit(X_train, Y_train, epochs=500, verbose=0, batch_size=256)
          Y_pred = model.predict(X_test, verbose=0)[:,0]
          new_rmse = np.sqrt(metrics.mean_squared_error(Y_test, Y_pred))
          print("Test RMSE = {}".format(new_rmse))
          if min_rmse > new_rmse:
              best_model = model
              min_rmse = new_rmse
    print("Mejor RMSE: {}".format(min_rmse))
    plot_model(best_model, to_file='{}.png'.format(model_name), show_shapes=True, show_layer_names=True)
    return best_model
```

In [0]:

```
def print_prediction(real, pred, dias):
    """ Prints two plots: 1:a bar for real data and a bar for predicted data 2:the difference between real and pr
edicted data
    """
    #Print real and predicted data
    y = np.zeros((real.shape[0],3))
    y[:,0]= np.arange(real.shape[0])
    y[:,1] = real
    y[:,2] = pred
    df = pd.DataFrame(y, columns=["Días", "Real", "Predicción"]).head(dias)
    ax = df.plot(x="Días", y=["Real", "Predicción"], kind="bar")
    plt.show()


    #Print the difference
    y = np.zeros((real.shape[0],2))
    y[:,0] = np.arange(real.shape[0])
    y[:,1] = real - pred
    df = pd.DataFrame(y, columns=["Días", "Diferencia"]).head(dias)
    ax = df.plot(x="Días", y="Diferencia", kind="bar")
    plt.show()
```

In [0]:

```
def get_number_of_layers(X_train,Y_train,num_features):
  train_cost = []
  val_cost = []
  dots = np.linspace(1,5,5)
  for elem in dots:
      print("LAYER N° {}".format(elem))
      model = Sequential()
      for i in range(0,int(elem)-1):
        model.add(LSTM(32, activation='relu',return_sequences=True, input_shape=(num_features,1)))

      if elem == 1 :
        model.add(LSTM(32, activation='relu', input_shape=(num_features,1)))
      else:
        model.add(LSTM(32, activation='relu'))
      model.add(Dense(1))
      model.compile(optimizer='adam', loss='mse')
      history = model.fit(X_train, Y_train, nb_epoch=250, batch_size=128, validation_split=0.2, shuffle=True,verb
ose=0)
      train_cost.append(np.sqrt(history.history['loss'][-1:]))
      val_cost.append(np.sqrt(history.history['val_loss'][-1:]))
  return train_cost,val_cost
```

Modelo nº 7: Nº de avisos en función del Mes y el día

Paso nº 1: Obtenemos variables

In [0]:

```
#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BA_CO", "AB_CO", "SUCESO"]].groupby(["AÑO",
"MES", "DIA", "DIA_CODIFICADO", "BA_CO", "AB_CO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO", "BA_CO", "AB_CO"]]



#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BA_CO", "AB_CO", "SUCESO"]].groupby(["AÑO", "
MES", "DIA", "DIA_CODIFICADO", "BA_CO", "AB_CO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO", "BA_CO", "AB_CO"]]
```

In [0]:

```
data_train.head(5)
```

Out[0]:

	AÑO	MES	DIA	DIA_CODIFICADO	BA_CO	AB_CO	SUCESO
0	2009	1	1	3	-1	0	4
1	2009	1	1	3	-1	2	1
2	2009	1	1	3	-1	20	15
3	2009	1	1	3	-1	28	1
4	2009	1	1	3	-1	30	7

Parte nº 2: Trasformamos los datos a array de numpy para poder utilziarlos en Keras

In [0]:

```
X_train = X_train.to_numpy()
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.to_numpy()
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

Parte nº 3: Buscamos a partir de que número de capas hay Overfitting

Es decir, buscamos el mínimo error en el conjunto de validación

In [0]:

```
train_cost,val_cost = get_number_of_layers(X_train,Y_train,5)
```

```
LAYER N° 1.0
LAYER N° 2.0
LAYER N° 3.0
LAYER N° 4.0
LAYER N° 5.0
```

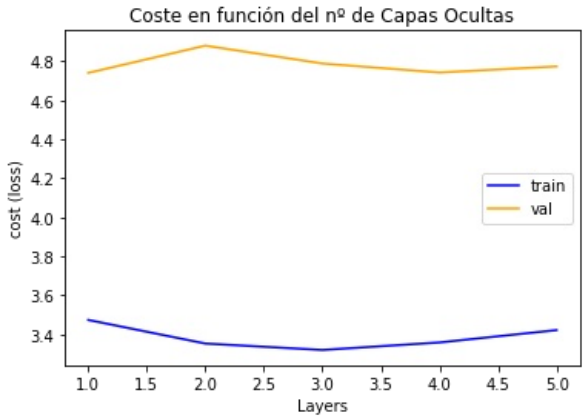
In [0]:

```
dots = np.linspace(1,5,5)
plt.plot(dots,train_cost, color="blue")
plt.plot(dots,val_cost, color="orange")
plt.legend(['train', 'val'])
plt.ylabel('cost (loss)')
plt.xlabel('Layers')
plt.title("Coste en función del n° de Capas Ocultas")
```

Out[0]:

```
Text(0.5, 1.0, 'Coste en función del n° de Capas Ocultas')
```

Out[0]:



Parte nº 4: Buscamos el mejor modelo con diferentes configuraciones de regularización

Debido a la gran similitud entre los resultados con 1 y 4 capas, se escoge 1 capa por menor coste de ejeución.

In [0]:

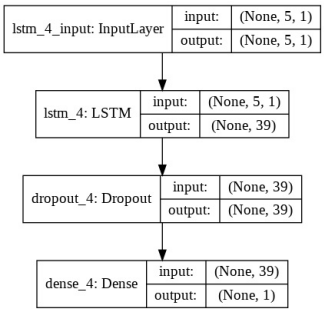
```
model = draw_layer_size(X_train,X_test,Y_test,5,"model_07",1,2)
```

```
Fit model with [27] sizes and Dropout = 0.1
Test RMSE = 4.453089760253579
Fit model with [27] sizes and Dropout = 0.2
Test RMSE = 4.40971828170391
Fit model with [27] sizes and Dropout = 0.3
Test RMSE = 4.488555126421712
Fit model with [39] sizes and Dropout = 0.1
Test RMSE = 4.351798020907941
Fit model with [39] sizes and Dropout = 0.2
Test RMSE = 4.4191655916292
Fit model with [39] sizes and Dropout = 0.3
Test RMSE = 4.403703604072228
Mejor RMSE: 4.351798020907941
```

In [0]:

```
Image(retina=True, filename='model_07.png')
```

Out[0]:



Parte nº5: Guardamos el modelo creado

In [0]:

```
pickle.dump(model, open('NN_cloud_07.sav', 'wb'))
files.download('NN_cloud_07.sav')
```

Modelo nº 8: Nº de avisos para un Mes, un día, un día de la semana¶

Paso nº 1: Obtenemos variables

In [0]:

```
#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DI", "AB_CO", "SUCESO"]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DI", "AB_CO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO", "DI", "AB_CO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DI", "AB_CO", "SUCESO"]].groupby(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "DI", "AB_CO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO", "DI", "AB_CO"]]
```

In [0]:

```
data_train.head(5)
```

Out[0]:

	AÑO	MES	DIA	DIA_CODIFICADO	DI	AB_CO	SUCESO
0	2009	1	1	3	-1	0	8
1	2009	1	1	3	-1	1	5
2	2009	1	1	3	-1	2	5
3	2009	1	1	3	-1	20	1
4	2009	1	1	3	0	0	1

Parte nº 2: Trasformamos los datos a array de numpy para poder utilziarlos en Keras

In [0]:

```
X_train = X_train.to_numpy()
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.to_numpy()
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

Parte nº 3: Buscamos a partir de que número de capas hay Overfitting

Es decir, buscamos el mínimo error en el conjunto de validación

In [0]:

```
train_cost,val_cost = get_number_of_layers(X_train,Y_train,5)
```

```
LAYER N° 1.0
LAYER N° 2.0
LAYER N° 3.0
LAYER N° 4.0
LAYER N° 5.0
```

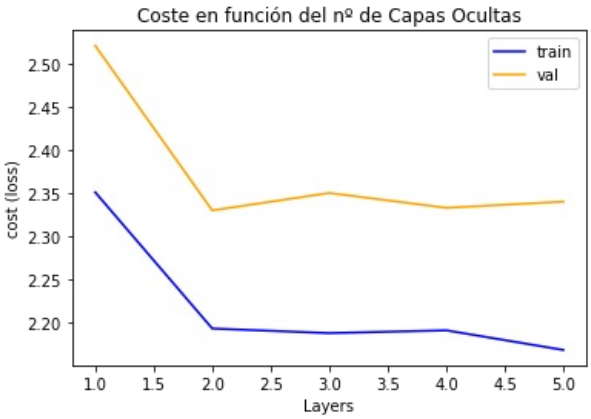
In [0]:

```
dots = np.linspace(1,5,5)
plt.plot(dots,train_cost, color="blue")
plt.plot(dots,val_cost, color="orange")
plt.legend(['train', 'val'])
plt.ylabel('cost (loss)')
plt.xlabel('Layers')
plt.title("Coste en función del n° de Capas Ocultas")
```

Out[0]:

```
Text(0.5, 1.0, 'Coste en función del n° de Capas Ocultas')
```

Out[0]:



Parte nº 4: Buscamos el mejor modelo con diferentes configuraciones de regularización

In [0]:

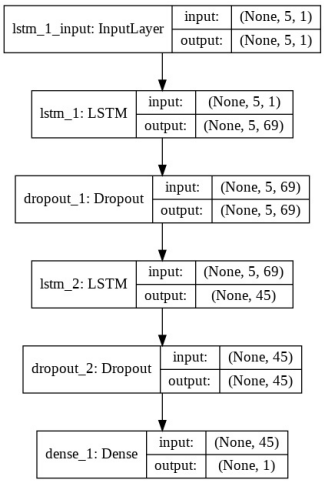
```
model = draw_layer_size(X_train,X_test,Y_test,5,"model_08",2,1)
```

```
Fit model with [69, 45] sizes and Dropout = 0.1
Test RMSE = 2.4126735380004023
Fit model with [69, 45] sizes and Dropout = 0.2
Test RMSE = 2.4542665640174715
Fit model with [69, 45] sizes and Dropout = 0.3
Test RMSE = 2.48917337783909
Mejor RMSE: 2.4126735380004023
```

In [0]:

```
Image(retina=True, filename='model_08.png')
```

Out[0]:



Parte nº5: Guardamos el modelo creado

In [0]:

```
pickle.dump(model, open('NN_cloud_08.sav', 'wb'))
files.download('NN_cloud_08.sav')
```

Modelo nº 9: nº de Avisos para un mes, día , día de la semana y dispositivo

Paso nº 1: Obtenemos variables

In [0]:

```
#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "AB_CO", "SUCESO"]].groupby(["AÑO", "MES", "
DIA", "DIA_CODIFICADO", "AB_CO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO", "AB_CO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "AB_CO", "SUCESO"]].groupby(["AÑO", "MES", "DI
A", "DIA_CODIFICADO", "AB_CO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO", "AB_CO"]]
```

In [0]:

```
data_train.head(5)
```

Out[0]:

	AÑO	MES	DIA	DIA_CODIFICADO	AB_CO	SUCESO
0	2009	1	1	3	1	126
1	2009	1	1	3	2	87
2	2009	1	1	3	18	7
3	2009	1	1	3	20	15
4	2009	1	1	3	28	1

Parte nº 2: Trasformamos los datos a array de numpy para poder utilziarlos en Keras

In [0]:

```
X_train = X_train.to_numpy()
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.to_numpy()
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

Parte nº 3: Buscamos a partir de que número de capas hay Overfitting

Es decir, buscamos el mínimo error en el conjunto de validación

In [0]:

```
train_cost,val_cost = get_number_of_layers(X_train,Y_train,4)
```

```
LAYER Nº 1.0
LAYER Nº 2.0
LAYER Nº 3.0
LAYER Nº 4.0
LAYER Nº 5.0
```

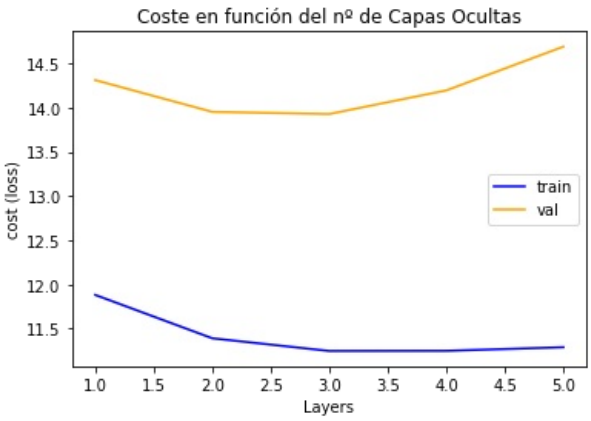
In [0]:

```
dots = np.linspace(1,5,5)
plt.plot(dots,train_cost, color="blue")
plt.plot(dots,val_cost, color="orange")
plt.legend(['train', 'val'])
plt.ylabel('cost (loss)')
plt.xlabel('Layers')
plt.title("Coste en función del n° de Capas Ocultas")
```

Out[0]:

```
Text(0.5, 1.0, 'Coste en función del n° de Capas Ocultas')
```

Out[0]:



Parte nº 4: Buscamos el mejor modelo con diferentes configuraciones de regularización

In [0]:

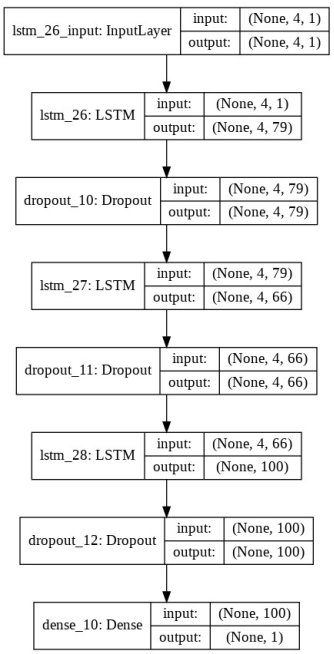
```
model = draw_layer_size(X_train,X_test,Y_test,4,"model_09",3,1)
```

```
Fit model with [79, 66, 100] sizes and Dropout = 0
Test RMSE = 14.868905366405214
Fit model with [79, 66, 100] sizes and Dropout = 0.1
Test RMSE = 14.476937530316837
Fit model with [79, 66, 100] sizes and Dropout = 0.2
Test RMSE = 14.089183754571682
Fit model with [79, 66, 100] sizes and Dropout = 0.3
Test RMSE = 14.300138003406701
Mejor RMSE: 14.089183754571682
```

In [0]:

```
Image(retina=True, filename='model_09.png')
```

Out[0]:



Parte nº5: Guardamos el modelo creado

In [0]:

```
pickle.dump(model, open('NN_cloud_09.sav', 'wb'))
files.download('NN_cloud_09.sav')
```

Modelo nº 10: nº de Avisos para un mes, día , día de la semana, base, turno y dispositivo

Paso nº 1: Obtenemos variables

In [0]:

```
#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BA_CO", "T_CO", "AB_CO", "SUCESO"]].groupby
(["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BA_CO", "T_CO", "AB_CO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO", "BA_CO", "T_CO", "AB_CO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "BA_CO", "T_CO", "AB_CO", "SUCESO"]].groupby([
"AÑO", "MES", "DIA", "DIA_CODIFICADO", "BA_CO", "T_CO", "AB_CO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO", "BA_CO", "T_CO", "AB_CO"]]
```

In [0]:

```
data_train.head(5)
```

Out[0]:

	AÑO	MES	DIA	DIA_CODIFICADO	BA_CO	T_CO	AB_CO	SUCESO
0	2009	1	1	3	-1	0	0	4
1	2009	1	1	3	-1	0	20	2
2	2009	1	1	3	-1	0	28	1
3	2009	1	1	3	-1	0	30	1
4	2009	1	1	3	-1	1	20	3

Parte nº 2: Trasformamos los datos a array de numpy para poder utilziarlos en Keras

In [0]:

```
X_train = X_train.to_numpy()
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.to_numpy()
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

Parte nº 3: Buscamos a partir de que número de capas hay Overfitting

Es decir, buscamos el mínimo error en el conjunto de validación

In [0]:

```
train_cost,val_cost = get_number_of_layers(X_train,Y_train,6)
```

```
LAYER Nº 1.0
LAYER Nº 2.0
LAYER Nº 3.0
LAYER Nº 4.0
LAYER Nº 5.0
```

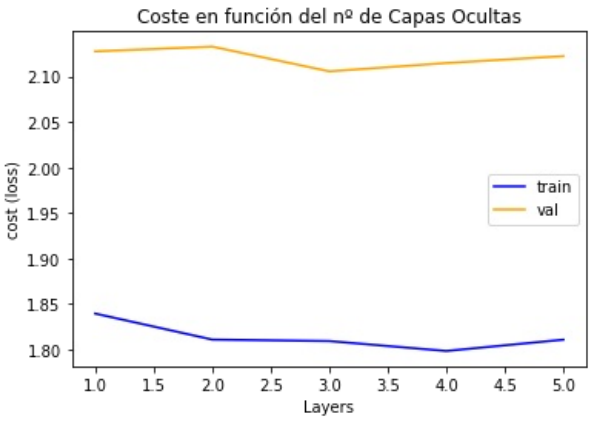
In [0]:

```
dots = np.linspace(1,5,5)
plt.plot(dots,train_cost, color="blue")
plt.plot(dots,val_cost, color="orange")
plt.legend(['train', 'val'])
plt.ylabel('cost (loss)')
plt.xlabel('Layers')
plt.title("Coste en función del n° de Capas Ocultas")
```

Out[0]:

```
Text(0.5, 1.0, 'Coste en función del n° de Capas Ocultas')
```

Out[0]:



Parte nº 4: Buscamos el mejor modelo con diferentes configuraciones de regularización

In [0]:

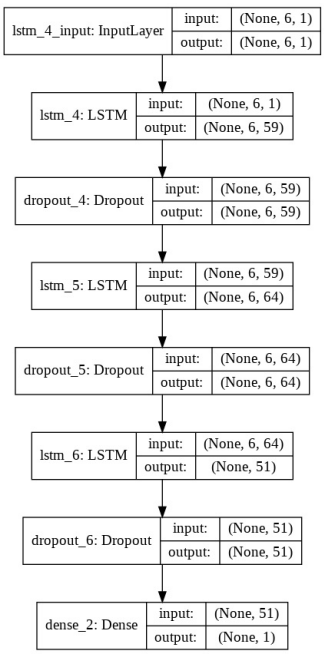
```
model = draw_layer_size(X_train,X_test,Y_test,6,"model_10",3,1)
```

```
Fit model with [59, 64, 51] sizes and Dropout = 0
Test RMSE = 2.079678392137927
Fit model with [59, 64, 51] sizes and Dropout = 0.1
Test RMSE = 2.058356630081132
Fit model with [59, 64, 51] sizes and Dropout = 0.2
Test RMSE = 2.059840146045658
Fit model with [59, 64, 51] sizes and Dropout = 0.3
Test RMSE = 2.111917913434203
Mejor RMSE: 2.058356630081132
```

In [0]:

```
Image(retina=True, filename='model_10.png')
```

Out[0]:



Parte nº5: Guardamos el modelo creado

In [0]:

```
pickle.dump(model, open('NN_cloud_10.sav', 'wb'))
files.download('NN_cloud_10.sav')
```

Modelo nº 12: nº de Avisos para un mes, día , día de la semana, dispositivo y turno

Paso nº 1: Obtenemos variables

In [0]:

```
#Define X and Y for training
data_train = train_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "T_CO", "AB_CO", "SUCESO"]].groupby(["AÑO",
"MES", "DIA", "DIA_CODIFICADO", "T_CO", "AB_CO"]).count()
data_train = data_train[(np.abs(stats.zscore(data_train)) < 3).all(axis=1)] #Eliminamos outliers
Y_train = data_train["SUCESO"]
data_train = data_train.reset_index()
X_train = data_train[["MES", "DIA", "DIA_CODIFICADO", "T_CO", "AB_CO"]]

#Define X and Y for test
data_test = test_dataframe[["AÑO", "MES", "DIA", "DIA_CODIFICADO", "T_CO", "AB_CO", "SUCESO"]].groupby(["AÑO", "M
ES", "DIA", "DIA_CODIFICADO", "T_CO", "AB_CO"]).count()
data_test = data_test[(np.abs(stats.zscore(data_test)) < 3).all(axis=1)] #Eliminamos outliers
Y_test = data_test["SUCESO"]
data_test = data_test.reset_index()
X_test = data_test[["MES", "DIA", "DIA_CODIFICADO", "T_CO", "AB_CO"]]
```

In [0]:

```
data_train.head(5)
```

Out[0]:

	AÑO	MES	DIA	DIA_CODIFICADO	T_CO	AB_CO	SUCESO
0	2009	1	1	3	0	1	52
1	2009	1	1	3	0	2	7
2	2009	1	1	3	0	18	3
3	2009	1	1	3	0	20	2
4	2009	1	1	3	0	28	1

Parte nº 2: Trasformamos los datos a array de numpy para poder utilziarlos en Keras

In [0]:

```
X_train = X_train.to_numpy()
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.to_numpy()
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

Parte nº 3: Buscamos a partir de que número de capas hay Overfitting

Es decir, buscamos el mínimo error en el conjunto de validación

In [0]:

```
train_cost,val_cost = get_number_of_layers(X_train,Y_train,5)
```

```
LAYER Nº 1.0
LAYER Nº 2.0
LAYER Nº 3.0
LAYER Nº 4.0
LAYER Nº 5.0
```

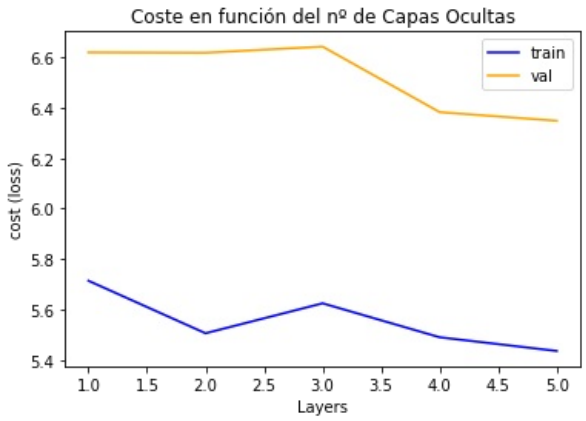
In [0]:

```
dots = np.linspace(1,5,5)
plt.plot(dots,train_cost, color="blue")
plt.plot(dots,val_cost, color="orange")
plt.legend(['train', 'val'])
plt.ylabel('cost (loss)')
plt.xlabel('Layers')
plt.title("Coste en función del n° de Capas Ocultas")
```

Out[0]:

```
Text(0.5, 1.0, 'Coste en función del n° de Capas Ocultas')
```

Out[0]:



Parte nº 4: Buscamos el mejor modelo con diferentes configuraciones de regularización

In [0]:

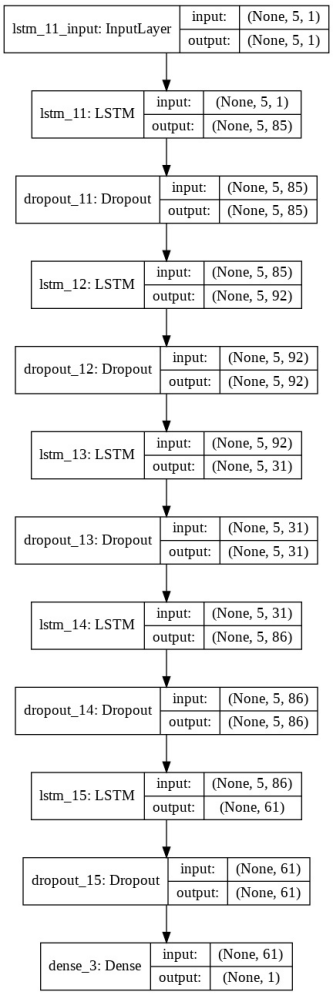
```
model = draw_layer_size(X_train,X_test,Y_test,5,"model_12",5,1)
```

```
Fit model with [85, 92, 31, 86, 61] sizes and Dropout = 0
Test RMSE = 6.589489205434713
Fit model with [85, 92, 31, 86, 61] sizes and Dropout = 0.1
Test RMSE = 6.526228716283307
Fit model with [85, 92, 31, 86, 61] sizes and Dropout = 0.2
Test RMSE = 6.456113157559142
Fit model with [85, 92, 31, 86, 61] sizes and Dropout = 0.3
Test RMSE = 6.602728435124551
Mejor RMSE: 6.456113157559142
```

In [0]:

```
Image(retina=True, filename='model_12.png')
```

Out[0]:



Parte nº5: Guardamos el modelo creado

In [0]:

```
pickle.dump(model, open('NN_cloud_12.sav', 'wb'))
files.download('NN_cloud_12.sav')
```

Propósito del documento

El propósito de este documento es facilitar la puesta en marcha por parte del tribunal del prototipo web creado para SAMUR - Protección Civil ya que el despliegue queda en sus manos a petición suya.

El prototipo se puede ejecutar en dos entornos y será accesible tanto por Google Chrome como por Mozilla Firefox::

- Entorno Docker (Recomendado)
- Entorno local

0.1. Entorno Docker

Para facilitar la ejecución del proyecto, sin necesidad de instalar todos los componentes en el equipo, se han creado dos contenedores (Dockers), uno de ellos contiene la app de node.js y el otro el servidor de bases de datos, de esta forma con unas simples instrucciones tendremos el proyecto puesto en marcha. Para ello:

0.1.1. Sistemas Linux

Instalar docker y docker-compose, si ya dispone de ambas, puede ignorar este paso, se ha seguido Docker Install Guide como guía:

```
$ sudo apt-get update

$ sudo apt-get install \
   apt-transport-https \
   ca-certificates \
   curl \
   gnupg-agent \
   software-properties-common

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

$ sudo apt-key fingerprint 0EBFCD88
```

```
$ sudo add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) \
  stable"

$ sudo apt-get update

$ sudo apt-get install docker-ce docker-ce-cli containerd.io

$ sudo curl -L
↪  "https://github.com/docker/compose/releases/download/1.25.5/docker-compose-$(uname
↪  -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

Si el comando anterior falla, probablemente sea por que falte un espacio después del final de cada línea (después de -L y despues del primer uname) y no se haya copiado correctamente, puede encontrarlo aqui.

```
$ sudo chmod +x /usr/local/bin/docker-compose
```

Iniciar sesión en Docker hub mediante el comando docker login, si ya dispone de cuenta de docker hub y lo tiene configurado en su equipo, no es necesario:

```
$ docker login
```

Username: tfgsamur2020 **Password**: samur_2020

Ejecutar proyecto (este apartado puede tardar varios minutos):

```
$ docker-compose build

$ docker-compose up
```

Tras ello si accedemos a la ruta localhost tendremos acceso al prototipo Web, introduciendo las siguientes credenciales:

User: 1 **Contraseña**: password

0.1.2. Sistemas Windows

Descargar Docker Desktop

Ejecuta el siguiente comando para comprobar la instalación de docker en un terminal de Windows:

```
$ docker -- version
```

Iniciar sesión en Docker hub mediante el comando docker login, si ya dispone de cuenta de docker hub y lo tiene configurado en su equipo, no es necesario:

```
$ docker login
```

Username: tfgsamur2020 **Password**: samur_2020

Ejecuta la imagen/ docker de test

```
$  docker run hello-world
```

Sitúate dentro de la carpeta app del proyecto y ejecuta los siguientes comandos (este apartado puede tardar varios minutos):

```
$  docker-compose build
```

```
$  docker-compose up
```

Tras ello la aplicación ya se estará ejecutando, solo deberá acceder a la dirección localhost para comprobar su funcionamiento.

User: 1 **Contraseña**: password

0.2. Entorno local

Para poder ejecutar el prototipo en local es necesario tener en el equipo instalado las siguientes tecnologías:

- Python 3.7: Para descargarlo debemos seguir el manual situado aqui
- Npm: Para descargarlo debemos seguir el manual situado en aqui
- XAMMP: Para descargarlo debemos seguir el manual situado en aqui

Una vez instalado ambos, para ejecutar el proyecto debemos realizar los siguientes pasos:

- Abrir Xammp
- Arrancar el servidor mysql y apache, e ir a la dirección: localhost/phpmyadmin/
- Crear una base de datos denominada “samur”, para ello en el entorno de desarrollo de phpmyadmin , escribiremos la siguiente consulta:

  ```
  CREATE DATABASE samur;
  ```

- Sobre la base de datos creada anteriormente, copiamos el contenido del archivo init_db.sql situado en el proyecto en la carpeta mysql y ejecutamos.

Ejecutar app: Situarnos en la carpeta app dentro del proyecto y ejecutar las instrucciónes

```
$  npm install
```

```
$  npm start app.js
```

Tras ello nos aparecerá un mensaje, indicando a qué ruta debemos acceder para acceder al prototipo web.

Introducir credenciales:

User: 1 **Contraseña**: password