Identificación de idiomas mediante técnicas procesamiento de lenguaje natural
Language identification using natural language processing techniques

Trabajo de Fin de Grado
Curso 2022–2023

Autores
Daniel Lucas Caturla
Rogger Huayllasco De la Cruz
Jaime Sánchez Rodríguez

Director
Mercedes García Merayo

Facultad de Informática
Universidad Complutense de Madrid

Identificación de idiomas mediante técnicas procesamiento de lenguaje natural
Language identification using natural language processing techniques

Trabajo de Fin de Grado

Departamento de Sistemas Informáticos y Computación

Autores

Daniel Lucas Caturla

Rogger Huayllasco De la Cruz

Jaime Sánchez Rodríguez

Director

Mercedes García Merayo

Facultad de Informática

Universidad Complutense de Madrid

29 de MAYO de 2023

Dedicatoria

A nuestros profesores y maestros, por conducirnos hacia la culminación de esta etapa y a la consecución de nuestros sueños

Agradecimientos

Dani

A mis padres, por su apoyo incondicional y por el sacrificio que han hecho para poder darme una educación universitaria, a pesar de todas las dificultades.

Rogger

A mis compañeros, cuya implicación y gran compañerismo han sido pilares fundamentales desde el inicio de este proyecto. Su apoyo constante y su espíritu colaborativo me han brindado la motivación necesaria para perseverar y alcanzar el éxito en este proyecto. Y a mis padres y hermano, por apoyarme y animarme a seguir adelante incluso en los momento mas difíciles.

Jaime

A mi padre, por sus innumerables y valiosos consejos durante la creación de esta memoria. A Momo, por su paciencia y ayuda durante las largas noches de redacción. Y a Alur, Crupi y Malks, por ayudarnos a salir de la trinchera.

Resumen

Identificación de idiomas mediante técnicas procesamiento de lenguaje natural

El presente proyecto, consiste en el análisis, diseño e implementación de un sistema que identifique el lenguaje en el que se ha escrito un texto. El objetivo es el de comparar diferentes implementaciones del algoritmo para evaluar la rapidez y eficiencia sobre cada idioma en el alcance.

Para ello se dispone de múltiples textos escritos en diferentes idiomas europeos (obtenidos de la base de datos del Parlamento Europeo) con los que trabajar a lo largo del proceso.

Así pues, el proyecto consta de dos partes. Por un lado, se elaborarán programas que adaptarán los textos originales a un formato entendible por los algoritmos de detección de idiomas escogidos, y por otro se realizarán pruebas de tiempo y eficiencia sobre los algoritmos de detección de idiomas para evaluar su potencia a la hora de detectar los diferentes idiomas del alcance.

Tanto los algoritmos de detección como el programa de adaptación de textos estarán escritos en el lenguaje de programación Python.

Palabras clave

Python, Inteligencia Artificial, Aprendizaje Automático, Random Forest, K-Nearest Neighbors, Support Vector, Comparativa, Idioma.

Abstract

Language identification using natural language processing techniques

The present project consists of the analysis, design and implementation of a system that identifies the language in which a text has been written. The objective is to compare different implementations of the algorithm to evaluate the speed and efficiency over each language in scope.

For this purpose, multiple texts written in different European languages (obtained from the database of the European Parliament) are available to work with throughout the process.

Thus, the project consists of two parts. On one hand, programs will be developed to adapt the original texts to a format understandable by the chosen language detection algorithms, and on the other hand, speed and efficiency tests will be performed on the language detection algorithms to evaluate their power in detecting the different languages within the scope.

Both the detection algorithms and the text adaptation program will be written in the Python programming language.

Keywords

Python, Artificial Intelligence, Machine Learning, Random Forest, K-Nearest Neighbors, Support Vector, Comparison, Language.

Índice

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

"En ese entonces se hablaba un solo idioma en toda la tierra."
— Génesis 11:1

El presente documento describe el proceso llevado a cabo para la elaboración del proyecto, explicando fase por fase los pasos realizados y aportando el resultado de los experimentos al final.

1.1. Motivación

En un mundo cada vez más globalizado e interconectado, existen muchas diferencias entre unos lugares y otros: distancia física, diferentes usos horarios, diferencias culturales, etc. Entre estas diferencias, quizá la que más afecta a la comunicación y los acuerdos entre personas y organizaciones es la lingüística.

Existen más de 7.000 idiomas en todo el mundo, incluyendo variantes y dialectos reconocidos. Si bien en Europa "solo" se hablan unos 286 idiomas, el caso de Asia es más impactante, con una suma de más de 2.300 idiomas y variantes.

Debido a esta diferenciación lingüística tan marcada, los dispositivos y programas de traducción simultánea han ido ganando con el tiempo más predominancia y son utilizados por cada vez más usuarios para uso personal (turismo, viajes de negocios, etc.). Existen muchos programas o dispositivos diferentes que permiten la traducción simultánea, pero todos ellos tienen algo en común: el primer paso consiste en identificar el idioma que se está escuchando, para poder traducirlo al idioma elegido por el usuario. Por ello, en una comunicación tan rápida y directa como es la verbal, la velocidad de traducción del programa debe ser competente y permitir el intercambio de frases con una velocidad tal que resulte cómodo para lo interlocutores. En realidad, una vez se conoce el idioma el mensaje fuente a traducir, los algoritmos han demostrado ser muy veloces en su labor de traducción. Sin embargo, detectar en qué idioma se está hablando o está escrito un texto suele ser lo que lleva más esfuerzo al programa. Hay muchos idiomas que comparten léxico, no así normas gramaticales ni tiempos verbales, por lo que la importancia de conocer qué algoritmos de detección de idiomas funcionan mejor en qué entornos es considerable.

1.2. Estado del arte

Existen múltiples trabajos con objetivos similares al presentado por este documento. En un estudio (Pavliy y Lewis, 2016) para el *Bulletin of Toyama University of International Studies* se compara la exactitud de los algoritmos de detección de idiomas de Google y Twitter para los idiomas ucraniano y ruso.

En el trabajo (Graham et al., 2014) se investiga como Twitter es capaz de identificar la nacionalidad de sus usuarios mediante el idioma de sus *tweets* y la geolocalización, pero no comparan la eficacia de los diferentes algoritmos utilizados por la red social.

Por último, cabe destacar un estudio (Grothe et al., 2008) acerca de la eficiencia y precisión de los métodos computacionales de detección de idiomas basados en los acercamientos de palabras cortas, palabras frecuentes y n-gramas, lo que acerca este estudio bastante al que se describe en esta memoria.

1.3. Objetivos

El objetivo último del presente proyecto es, como ya se ha comentado, el de comparar la eficacia de diferentes modelos de Aprendizaje Automático (en adelante AA) de detección de idiomas a la hora de tratar con textos escritos en diferentes lenguas. Este objetivo puede dividirse en dos fases.

1. Desarrollo del programa: Se elaborará un programa escrito en Python que permita tomar textos en diferentes idiomas y detectar en que idioma están escritos. Este programa constará de tres partes diferenciadas:

   a) Preprocesado de textos: Esta parte del programa es la encargada de tomar los textos originales y eliminar cualquier información no escrita en el idioma del texto o que no aporte conocimiento. Esto incluye caracteres especiales y palabras monolíteras (formadas por un carácter).

   b) Construcción del *dataset*: Esta parte construirá el conjunto de palabras y textos sobre el que se ejecutarán los modelos de detección de idiomas, tomando como entrada los textos preprocesados anteriormente.

   c) Implementación de la interfaz: De forma adicional, se construirá una interfaz ejecutable desde el entorno de programación capaz de detectar el idioma de un texto escrito (una vez entrenado el modelo cargado en el programa).

2. Experimentos: Se ejecutarán diferentes modelos de Aprendizaje Automático de detección sobre los idiomas en el alcance del proyecto y se compararán parámetros a definir como velocidad, recursos utilizados y porcentajes de aciertos. Este objetivo se puede desgranar en los siguientes pasos:

   a) Definición de los experimentos: Se definirán las métricas y los parámetros de los experimentos a llevar a cabo, así como la ejecución de los mismos.

b) Realización de los experimentos: Se seguirá la guía definida en el anterior paso y se anotarán todos los resultados obtenidos.

c) Análisis de resultados: Se procesarán los resultados y se presentarán de manera clara y entendible. Se elaborará una conclusión extraída de los resultados.

d) Selección del Modelo: Una vez analizados los resultados se seleccionará el modelo más eficiente y se cargará el programa ejecutable con ese modelo para su posterior uso.

1.4. Plan de trabajo

Al comienzo del proyecto, se estableció una estimación de tiempo para los objetivos y los pasos previos a su realización, de forma que se ajustara al tiempo total disponible para el proyecto.

1. **Investigación y revisión bibliográfica** (estimación: 2 semanas): En esta etapa inicial se realizará una investigación exhaustiva sobre los métodos y técnicas utilizadas en la detección de idiomas. Se buscarán artículos científicos, libros y otras fuentes relevantes para comprender cuales son las mejores prácticas en este dominio del Aprendizaje Automático y su funcionamiento.

2. **Recopilación y preparación de datos** (estimación: 2 semanas): Se recopilará un conjunto de datos que abarque la variedad de idiomas que permitirá hacer las pruebas de entrenamiento y evaluación en el futuro.

3. **Implementación del programa** (estimación: 8 semanas): Durante esta etapa se desarrollarán todos los algoritmos necesarios para la realización de los experimentos, así como algoritmos para la creación de los textos preprocesados, las tablas Tf-Idf y los *datasets*, que permitirán entrenar a los modelos.

4. **Entrenamiento y evaluación de los modelos** (estimación: 6 semanas): Los datos recopilados durante la segunda fase se usarán para entrenar y evaluar los modelos de detección de idiomas (Random Forest, K-Kneighbors y Support Vector). Se dividirán los datos en conjuntos de entrenamiento y prueba, se usarán técnicas de validación cruzada y se ajustarán los parámetros de los modelos para mejorar al máximo su precisión y eficiencia.

5. **Desarrollo de una interfaz de usuario** (estimación: 4 semanas): Se implementará una interfaz gráfica que sea intuitiva y que permita a los usuarios introducir texto plano o un archivo con extensión `txt` para predecir el idioma en el que está escrito. Para esto se empleará una serie de bibliotecas y herramientas adecuadas para hacer que la interfaz sea lo mas interactiva posible, atendiendo a su usabilidad y a la memorabilidad del usuario.

6. **Realización de los experimentos y análisis de los resultados** (estimación: 4 semanas): Se realizará una serie de experimentos exhaustivos utilizando un conjunto de datos adicionales, cuyos parámetros son elegidos de manera incremental para evaluar la robustez y el rendimiento de los modelos empleados. Se analizarán los resultados obtenidos y se elaborarán conclusiones sobre la efectividad de los algoritmos implementados.

Durante todas y cada de una de las fases se irán redactando las diferentes partes que componen la memoria.

1.5. Estructura del documento

El primer Capítulo (1. Introducción) consiste en un resumen ejecutivo con los objetivos del proyecto, el alcance del mismo y las fases en las que se divide el proyecto.

El segundo Capítulo (2. Preprocesado de Datos) abordará el proceso de creación del programa de preprocesado que adaptará los textos originales al formato aceptado por los algoritmos de detección de idiomas.

El tercer Capítulo (3. Creación del Dataset) describe el proceso de creación del programa que construirá los modelos computacionales sobre los textos tratados por el programa de preprocesado para poder llevar a cabo los experimentos del capítulo 5.

El cuarto Capítulo (4. Interfaz de Usuario) trata la creación de una interfaz que permite utilizar el programa en modo ventana.

El quinto Capítulo (5. Modelos) trata los diferentes modelos de detección de idioma, otorgando una definición sencilla de cada uno y comparando su funcionamiento.

El sexto Capítulo (6. Métodos de Evaluación) define las métricas y parámetros a medir y los experimentos a realizar para comparar la efectividad de cada algoritmo para los idiomas establecidos en el alcance.

El séptimo Capítulo (7. Resultados) mostrará los resultados obtenidos con los experimentos, aportando información visual y detallada para facilitar la comprensión de los resultados.

El octavo Capítulo (8. Conclusiones y Trabajo Futuro) contiene las conclusiones a las que se ha llegado gracias a los resultados de los experimentos, y aporta sugerencias de como continuar con el trabajo comenzado aquí.

Asimismo, se incluye la bibliografía y los apéndices al final del documento.

Capítulo 2

Preprocesado de datos

"Al emigrar al oriente, la gente encontró una llanura en la región de Sinar y allí se establecieron."
— Génesis 11:2

En el presente capítulo se aborda la creación del código de la fase de preprocesado, en la que se toman los textos originales y se obtienen los datos relevantes para emplear con los algoritmos de detección de idiomas. Se tratan todas las fases del desarrollo del software de la solución implementada, haciendo énfasis en la conversión de los requisitos funcionales y no funcionales en el diseño del aplicativo. Por último, se describen los entregables y archivos creados en el proceso descrito, que funcionan como base para la realización de la fase de los experimentos.

2.1. Tecnologías de desarrollo

Se utilizan diversas tecnologías de desarrollo para implementar el programa y evaluar los modelos. A continuación se detallan las principales herramientas utilizadas:

- Lenguaje de programación Python(v.3.11): Se ha decidido utilizar Python debido a su gran versatilidad y a las numerosas bibliotecas que facilitan el uso de modelos de Aprendizaje Automático (incluyendo el procesamiento de lenguaje natural). También se ha tenido en cuenta la eficiencia de las bibliotecas y sus algoritmos.
- Entorno virtual Anaconda(v22.9.0): Para facilitar la instalación de entornos de programación, paquetes y bibliotecas de Python se hará uso de Anaconda, que permite la gestión de entornos virtuales en el ámbito del desarrollo de software.
- Entorno Integrado de Desarrollo (IDE) Spyder(v5.4.3): Mediante Anaconda se instalará el IDE Spyder para Python, ya que proporciona un administrador de proyectos para organizar los diferentes archivos que se irán creando. También tiene un depurador integrado que permite la detención del programa en cualquier punto que elige el usuario, así como la exploración de las variables en dicho punto de detención, lo que facilita la implementación de código y la corrección de errores.

- Editor de textos Notepad++(v8.5): Se ha decidido utilizar Notepad++, que es un editor de textos con una serie de características muy útiles para la edición de código. Sin embargo, al no ser capaz de compilar y ejecutar código como el anterior entorno, los archivos de extensión `.py` generados se ejecutarán mediante consola.
- Aplicación de colaboración Microsoft Teams(1.6.0): Para las reuniones periódicas del equipo de desarrollo se utilizará esta aplicación, que permite mantener una comunicación fluida entre los miembros del equipo y facilita la planificación y coordinación de tareas.
- Repositorio online de datos Google Drive: Se usará como repositorio para almacenar y gestionar el código fuente del proyecto y los demás archivos relevantes (por ejemplo, los textos originales para preprocesar). Esta plataforma facilita la colaboración entre los miembros del equipo, permitiendo el acceso y la edición de código desde diferentes dispositivos.

Ya definido el software empleado para la ejecución del proyecto, se han escogido una serie de bibliotecas que facilitarán la implementación de los algoritmos de la parte de preprocesado de textos:

- `Re`: Es un modulo estándar que permite el uso de expresiones regulares. Se utiliza para la búsqueda de ciertos patrones y manipulación de cadenas de texto y caracteres especiales.
- `Os`: Es un modulo estándar que permite el uso de funciones propias del sistema operativo. Se utiliza para la eliminación de ficheros temporales.
- `Math`: Aporta gran número de funciones matemáticas. Se emplea para operaciones con números enteros y de coma flotante.

2.2. Análisis de requisitos

Los modelos de Aprendizaje Automático (Machine Learning en inglés) escogidos para el presente proyecto requieren que la entrada tenga un formato determinado. En concreto, es necesario que de los textos se utilice sólo lo que pueda aportar conocimiento a la máquina en la fase de aprendizaje y lo que posea conocimiento para la fase de experimentos (estas fases se presentan en el Capítulo 3). Todos los caracteres especiales, cifras, signos de puntuación, palabras monolíteras y cualquier elemento con un formato determinado que no aporte conocimiento se eliminarán del texto. Además, se extraen de los textos secuencias aleatorias de palabras para calcular su frecuencia relativa y se construyen las tablas TF-IDF (Term Frequency-Inverse Document Frequency[1]) que sirven de entrada a los modelos de Aprendizaje Automático.

[1]Las filas de la tabla representan los textos empleados y las columnas las secuencias de palabras que aparecen en ellos, de forma que los elementos de la tabla son las frecuencias relativas de cada secuencia en cada texto.

Por ello, se definen los siguientes requisitos funcionales (en adelante R.F.) que describen las funciones principales del código:

- **R.F.1** El programa eliminará caracteres especiales, signos de puntuación, palabras monolíteras y cifras del texto de entrada.
- **R.F.2** El programa tomará las secuencias de palabras *limpias* (es decir, que no contengan caracteres especiales ni ningún elemento que no aporte conocimiento al texto) de los textos y las escribirá en un único archivo de salida.

También se establece un requisito no funcional (en adelante R.N.F.) que aporta restricciones o condiciones a la ejecución:

- **R.N.F.1** En cada idioma pueden existir caracteres especiales como apóstrofes o tildes que aportan conocimiento[2]. Éstos caracteres deben conservarse para evitar la pérdida de conocimiento.

2.3. Diseño de la solución

Una vez determinados los requisitos de la solución, se definen las funcionalidades a alto nivel que debe tener el programa, también llamados casos de uso, y los componentes software que las englobarán.

Es importante tener en cuenta que este diseño está aún separado de los detalles de la implementación (que se abordarán en el siguiente apartado) y simplemente constituye una estructura funcional a alto nivel de la solución que podrá equivaler más o menos a la estructura de archivos que constituirán el código de la aplicación.

Como tal, el diagrama general de la solución es el que aparece en la Figura 2.1:

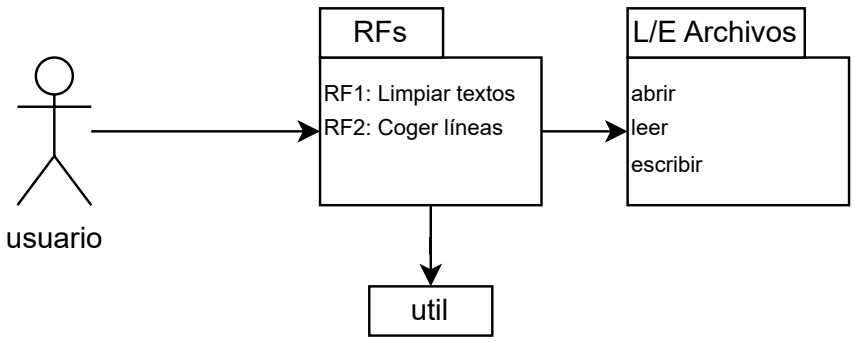

Figura 2.1: Diagrama de paquetes del programa de preprocesado

A continuación se detallan los componentes y funcionalidades que aparecen en el diagrama:

[2]e.g. La palabra *orégano* en español y *oregano* en inglés difieren solo en una tilde.

- FRs: En este componente se agrupan las funcionalidades principales de la solución, o en este caso, los requisitos funcionales definidos en el Análisis de Requisitos.
- L/E Archivos: Aquí se definen las operaciones básicas necesarias para la lectura y escritura en archivos de texto.
- util: Todas aquellas funcionalidades extra que permiten a los casos de uso del paquete FRs realizar su cometido se definirán en este paquete.

2.4. Implementación del código

Una vez definidos los componentes software principales que compondrán esta parte de la solución, queda trasladar las funcionalidades de alto nivel a funciones software de bajo nivel.

El lenguaje escogido para la implementación, como se detalló en la sección anterior, es Python (versión 3.11) debido a su facilidad para conectarse a interfaces de inteligencias artificiales. De hecho, al emplear Python no será necesario adaptar dos códigos en lenguajes diferentes, ya que los algoritmos escogidos para el procesamiento de frases están escritos en este lenguaje. Otra ventaja de este lenguaje consiste en la cantidad de bibliotecas ya creadas que aportan funcionalidades necesarias para el proyecto de forma optimizada, lo que evita tener que diseñar dichas funciones desde cero y ahorra costes en esfuerzo.

El diseño final del código para la parte de preprocesado queda, pues, como aparece en la Figura 2.2:

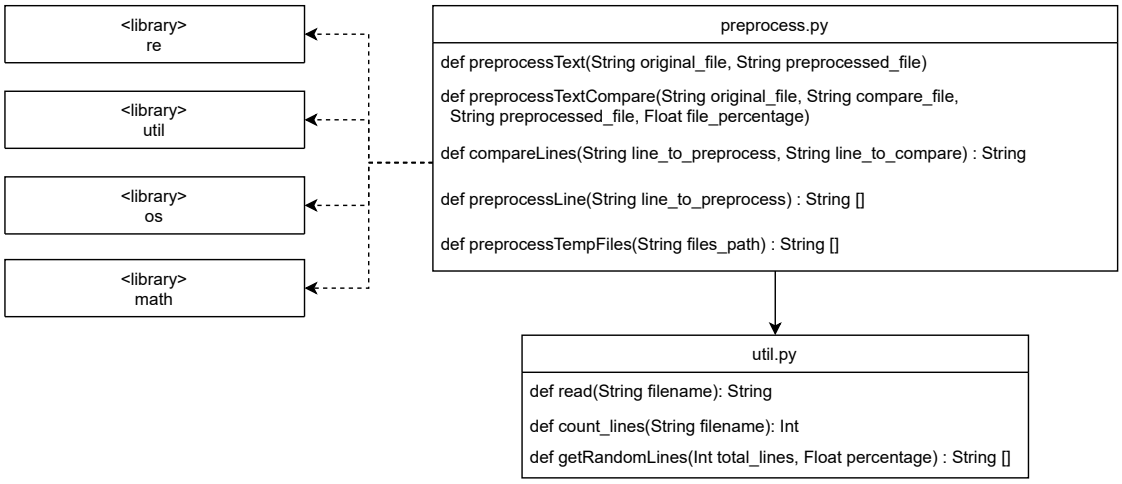

Figura 2.2: Diagrama de clases del programa de preprocesado

A continuación se describen los principales archivos del programa y las funciones que contienen:

- `preprocess.py`: Archivo principal de esta parte. Contiene las operaciones básicas que permiten realizar la limpieza del texto. Son las siguientes:

- `preprocessTextCompare`: Función principal del archivo. Preprocesa el porcentaje indicado por `file_percentage` del archivo `original_file` y lo escribe en el archivo `preprocessed_file`. Para la limpieza del texto compara `original_file` con `compare_file` para eliminar palabras en común entre ambos archivos.
- `preprocessText`: Función alternativa que preprocesa el archivo `original_file` y lo escribe en el archivo `preprocessed_file`, pero sin compararlo con ningún otro archivo.
- `compareLines`: Compara dos líneas preprocesadas por `preprocessLine()`, una del texto original y otra del texto a comparar, para eliminar las palabras en común en la línea del texto original.
- `preprocessLine`: Elimina los caracteres especiales de la línea pasada como parámetro.

▪ `util.py`: Archivo auxiliar con funciones necesarias para la realización de las operaciones de `preprocess.py`. Son las siguientes:

- `read`: Devuelve el contenido del fichero al que corresponde el nombre `filename`.
- `count_lines`: Devuelve el número de líneas que contiene el archivo al que corresponde el nombre `filename`.
- `getRandomLines`: Devuelve un conjunto de números aleatorios no repetidos, tantos como el producto entre `percentage` y `total_lines`.

Las bibliotecas y clases utilizadas en el código por las funciones anteriormente descritas son las definidas en la sección de Tecnologías de Desarrollo del presente capítulo.

2.5. Resultado del proceso

Una vez completadas la fase de diseño e implementación del preprocesado de textos, se dispone de los textos limpios que se emplearán en la construcción de los conjuntos de datos (en adelante *dataset*) para el entrenamiento y ejecución de los modelos. En el siguiente capítulo se detallará la construcción de estos *datasets* sobre los que se realizarán los experimentos, pero en este punto se dispone de los siguientes archivos como parte del código final del programa, que pueden verse en la Figura 2.3:

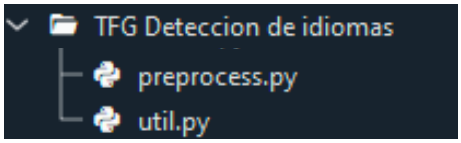

Figura 2.3: Directorio del programa de preprocesado de textos

Capítulo 3

Construcción del Dataset

"Luego dijeron: «Construyamos una ciudad con una torre que llegue hasta el cielo. De ese modo, nos haremos famosos y evitaremos ser dispersados por toda la tierra»."

— Génesis 11:4

En este capítulo se trata la creación del programa que construye el conjunto de datos a partir de los textos preprocesados anteriormente y se describe en que consiste el proceso de construcción y cada uno de los pasos en el diseño de la solución.

3.1. Contexto

El objetivo de esta fase consiste en construir un *dataset* que permita a los algoritmos de inteligencia artificial reconocer el idioma al que pertenece un texto.

El *dataset* consiste en una matriz donde las filas son secuencias de palabras extraídas de los diferentes textos preprocesados y las columnas las palabras que aparecen en las secuencias. De esta forma, se registra la importancia de cada palabra en cada colección de frases, construida cada una con textos de un único idioma.

Además de esta matriz, el *dataset* cuenta con información que relaciona cada secuencia de palabras con el idioma en el que está escrita y parámetros varios acerca de la construcción del *dataset*.

En las siguientes secciones se procede a definir la solución implementada para la construcción del *dataset*.

3.2. Tecnologías de desarrollo

Los programas y aplicaciones definidos en el anterior capítulo son también usados en esta parte del proyecto. Sin embargo, las librerías que otorgan la funcionalidad necesaria para llevar a cabo esta fase son las siguientes:

- `numpy`: Esta biblioteca es la encargada de la computación científica y el análisis de datos. Proporciona un conjunto de herramientas para trabajar con matrices y

arreglos multidimensionales y permite realizar operaciones matemáticas y estadísticas en grandes conjuntos de datos con alta eficiencia. En el programa se utiliza para hallar las columnas comunes entre el *dataset* y las tablas TF-IDF.

- `pickle`: Esta biblioteca se utiliza para guardar y cargar objetos en archivos de manera rápida. En el programa se usa para tratar los conjuntos de datos grandes de la manera mas eficiente posible.

- `itertools`: Esta biblioteca proporciona utilidades que nos facilitan el procesamiento y la iteración de las secuencias de datos contenidas en el *dataset*.

- `scikit-learn`: Contiene los modelos de Aprendizaje Automático utilizados por el programa para la detección de idiomas, así como funciones para trabajar con ellos. Los modelos utilizados en el programa vienen dados por las siguientes clases:

  - `RandomForestClassifier`
  - `KNeighborsClassifier`
  - `SVC`

  De esta libreria también se utilizan la clase `TfidfVectorizer` para la construcción de las tablas TF-IDF y clase `SelectKBest` para seleccionar las palabras más significativas de un dataset.

3.3. Análisis de requisitos

Se trata de definir la funcionalidad básica de la solución, que en breves palabras consiste en la construcción de las tablas TF-IDF y a partir de ellas completar el *dataset*.

Las funciones principales del código vendrán definidas por los siguientes R.F.:

- **R.F.3** El programa construirá una tabla de frecuencias (TF-IDF) a partir de secuencias de palabras extraídas aleatoriamente de un texto preprocesado.

- **R.F.4** El programa construirá un *dataset* y añadirá las tablas TF-IDF al mismo.

- **R.F.5** El programa podrá *entrenar* los diferentes modelos con la información contenida en el *dataset*.

- **R.F.6** Se podrá proporcionar al programa textos para que los modelos predigan en que idioma han sido escritos.

También se establece el siguiente R.N.F. que aporta restricciones o condiciones a la ejecución:

- **R.N.F.2** Las palabras no pueden estar repetidas en el *dataset*. Cada columna equivale a un término único.

- **R.N.F.3** Por defecto, todas las palabras tendrán valor TF-IDF igual a 0 para todas las secuencias de palabras de los idiomas en las que no haya habido ocurrencia de la misma.

3.4. Diseño de la solución

Como se deduce de los requisitos funcionales, se necesita funcionalidad que pueda manejar 3 elementos muy diferenciados: las tablas TF-IDF, los *datasets* y los modelos de Aprendizaje Automático. Por ello, se implementarán 3 módulos donde cada uno esté especializado en un elemento diferente.

Una vez realizada la primera aproximación entre los requisitos y el diseño, el resultado es el mostrado en la Figura 3.1.

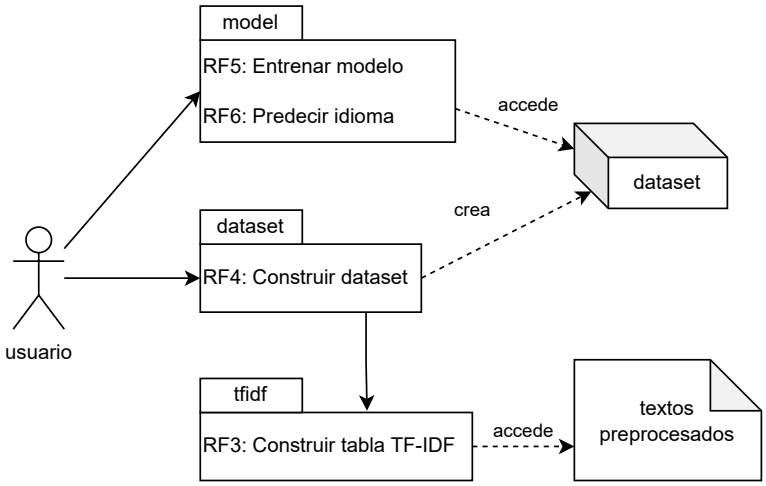

Figura 3.1: Diagrama de paquetes del programa de creación del *dataset*

Además de los requisitos funcionales que de nuevo aparecen en el diagrama, es interesante destacar que todo gira en torno a la gestión del *dataset*. Por ello, la relación entre los módulos `model` y `dataset` pasa por el acceso compartido al recurso principal del programa: el conjunto de secuencias de palabras, frecuencias y demás meta-información que permitirán a los modelos aprender y realizar su función.

3.5. Implementación del código

Con la arquitectura definida anteriormente, la implementación del código es prácticamente un paso directo, debido en parte a la cantidad de bibliotecas de las que dispone Python, que aportan una serie de funcionalidades muy útiles en la gestión de los datos. En la Figura 3.2 puede verse el diseño final de esta parte.

A continuación se describen los principales archivos del programa y las funciones que contienen:

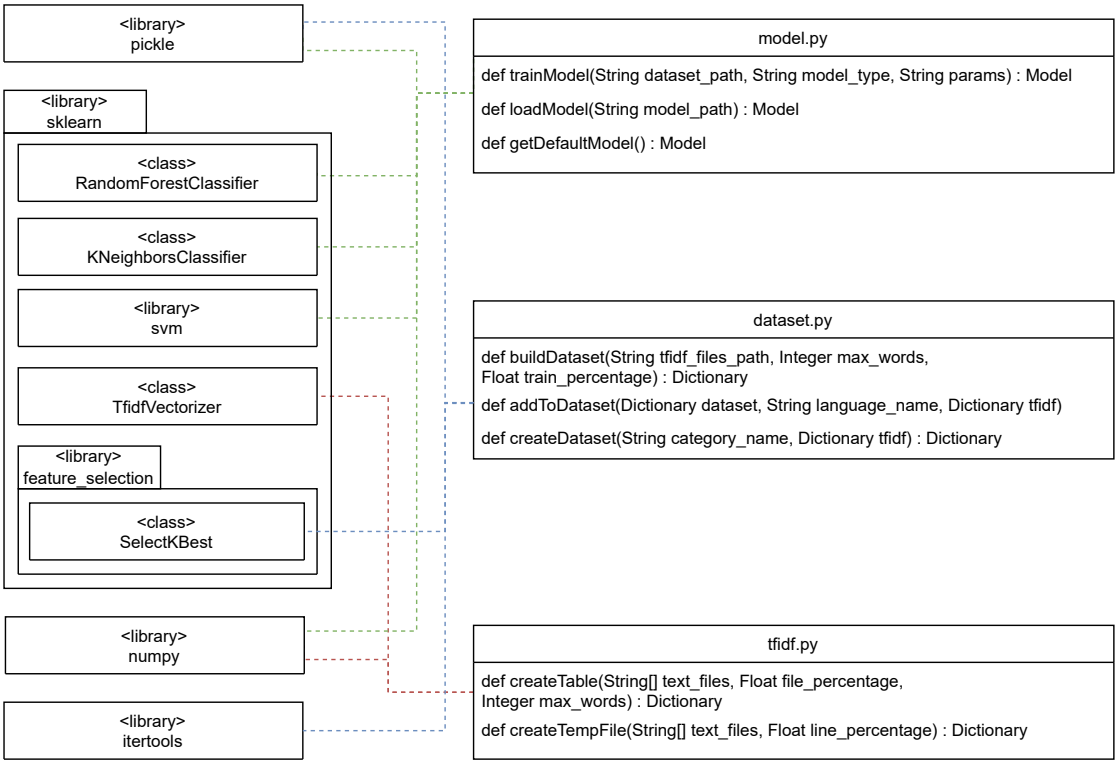

Figura 3.2: Diagrama de clases del programa de creación del *dataset*

- `tfidf.py`: Permite la construcción y edición de tablas de frecuencia TF-IDF.
  - `createTable(text_files, file_percentage, max_words)`: Construye la tabla de frecuencias TF-IDF de los textos incluidos en `text_files`. El parámetro `file_percentage` indica el porcentaje de lineas que se tomarán aleatoriamente de los textos y el parámetro `max_words` limita el número de palabras únicas que contiene el TF-IDF.
  - `createTempFile(text_files, line_percentage)`: Selecciona líneas aleatorias de cada texto dado por `text_files` y las escribe en un archivo temporal. El parámetro `line_percentage` indica el porcentaje de lineas a tomar de cada texto.
- `dataset.py`: Permite la construcción del *dataset*.
  - `buildDataset(tfidf_files_path, max_words, train_percentage)`: Crea el *dataset* con las tablas TF-IDF dadas por el parámetro `tfidf_files_path`. El parámetro `max_words` limita el número de palabras únicas que contiene el dataset y el parámetro `train_percentage` define el porcentaje de datos del dataset que se destinarán al entrenamiento de modelos.
  - `createDataset(category_name, tfidf)`: Crea e inicializa un *dataset* nuevo.
  - `addToDataset(dataset, language_name, tfidf)`: Añade la información contenida en `tfidf` a `dataset` junto con el idioma al que pertenece la tabla.

- `modelo.py`: Permite el entrenamiento y uso de los modelos de Aprendizaje Automático.
  - `trainModel(dataset_path, model_type, params)`: Entrena al modelo indicado por `model_type` con el *dataset* construido anteriormente y localizado en `dataset_path`. El parámetro opcional `params` permite el paso de valores adicionales para la configuración del entrenamiento.
  - `loadModel(model_path)`: Carga el modelo localizado en `model_path` para su uso.
  - `getDefaultModel()`: Devuelve un modelo entrenado y configurado, localizado en la carpeta default del proyecto..

Las bibliotecas y clases utilizadas en el código por las funciones anteriormente descritas son las definidas en la sección de Tecnologías de Desarrollo del presente capítulo.

3.6. Resultado del proceso

Una vez implementado el código que construye el *dataset* y permite entrenar a los modelos de Aprendizaje Automático, ya se dispone de toda la funcionalidad necesaria para realizar los experimentos, mostrada en la Figura 3.3.

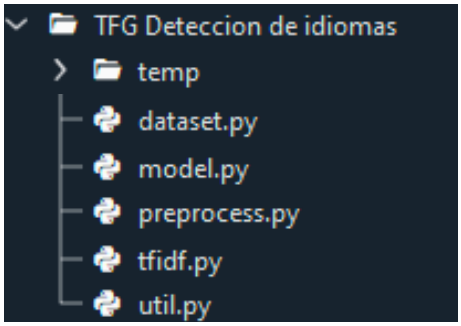

Figura 3.3: Directorio del programa con la funcionalidades de preprocesado y creación del *dataset*

Capítulo 4

Interfaz de Usuario

"Pero el Señor bajó para observar la ciudad y la torre que los hombres estaban construyendo."
— Génesis 11:5

Este capítulo abarca el proceso de implementación, ejecución y uso de una interfaz de usuario que permite utilizar los programas descritos en los capítulos anteriores de una forma más rápida y eficiente. Complementariamente, se presenta una segunda interfaz que permitirá probar las conclusiones de los experimentos a nivel usuario.

4.1. Funcionalidades

Las principales funciones de la interfaz son las siguientes:

1. Preprocesado del texto seleccionado.
2. Generación de la tabla TF-IDF correspondiente.
3. Construcción del *dataset* mediante las tablas.
4. Entrenamiento de los modelos de AA con el *dataset* construido.
5. Predicción del idioma de un texto mediante los modelos.

4.2. Implementación del código

La interfaz de usuario se ha implementado siguiendo el patrón MVC (Modelo-Vista-Controlador). Este patrón se basa en tres partes diferenciadas:

1. **Modelo**: Está formado por el conjunto de módulos de Python que se desarrollaron en los capítulos anteriores.
2. **Vista**: Para el diseño de la interfaz gráfica se han utilizado las bibliotecas estándar de Python `tKinter` y `tkk`, que facilitan la creación y colocación de elementos visuales.

3. **Controlador**: Se ha creado el archivo controller.py para conectar la interfaz gráfica con los módulos de la aplicación.

A continuación se describen las principales funciones que contiene `controller.py`:

- `runPreprocess(text_path, save_path, compare_path)`: Llama a la función de preprocesado `preprocessText`. El parámetro opcional `compare_path` permite llamar a la funcion `preprocessTextCompare`.
- `createTFIDF(texts, save_path, line_percentage, max_words)`: Llama a la función `createTFIDF` de `tfidf.py` y guarda el resultado en `save_path`.
- `buildDataset(tfidfs, save_path, max_words, train_percentage)`: Llama a la función `buildDataset` de `dataset.py` y guarda el resultado en save_path.
- `trainModel(dataset_path, save_path, model_type, model_params)`: Llama a la función `trainModel` de `model.py` y guarda el resultado en `save_path`.
- `runDetection(input_text,model_path)`: Función principal de la aplicación a nivel usuario. Predice el idioma del texto `input_text` con un modelo cargado por defecto. Incluye la opción de utilizar un modelo alternativo a través del parámetro opcional `model_path`.

4.3. Requisitos de ejecución

La aplicación se puede ejecutar en consola o en un entorno virtual de Python como `venv`, `virtualenv` o `conda`. En la carpeta raíz se encuentran los dos archivos ejecutables del proyecto.

- **main.py**: Carga la interfaz de usuario principal con todas las funcionalidades mencionadas anteriormente.
- **app.py**: Carga una interfaz gráfica que sirve como aplicación a nivel usuario. La única funcionalidad disponible en esta interfaz es la de predecir el idioma de un texto introducido por el usuario. El programa se cargará con un modelo entrenado previamente. El tipo de modelo, el *dataset* con el que será entrenado y la configuración de sus parámetros se seleccionará en base a las conclusiones obtenidas en el Capítulo 8. Conclusiones y Trabajo Futuro.

Los requisitos mínimos para la ejecución de ambos archivos son los siguientes:

- Tener instaladas las bibliotecas `scikit-learn`, `numpy`, `pandas` y `openpyxl` en el entorno utilizado. Se pueden instalar todas las bibliotecas necesarias ejecutando el siguiente comando en la consola: **pip install -r libraries.txt**, donde `libraries.txt` es un archivo incluido en la carpeta raíz del proyecto.

- Tener instalado Python (versión 3.11 o superior). Esto permite la ejecución de los dos archivos principales en consola por medio de los siguientes comandos: **python main.py** y **python app.py**

4.4. Funcionamiento

En las siguientes secciones se presentan las funcionalidades del programa principal `main.py` junto a a algunas de las vistas disponibles.

Menú Principal

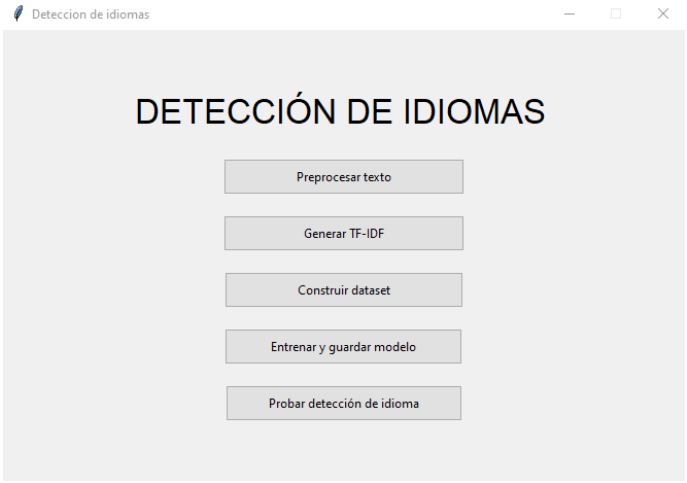

Figura 4.1: Vista del menú

Al ejecutar `main.py`, se abre un menú con todas las funcionalidades del programa. A continuación se muestran las vistas a las que se accede al pulsar los botones disponibles.

Preprocesar texto

Todas las funcionalidades del programa ofrecen una vista en las que se establecen los parámetros, como la mostrada en la figura 4.2. Los botones `Seleccionar` abren la ventana de exploración del sistema operativo y dan la opción de elegir un archivo, elegir varios archivos o determinar la ruta donde guardar un archivo, dependiendo del campo al que va ligado el botón.

En la vista de preprocesado se selecciona el texto a preprocesar, la ruta donde guardar el texto preprocesado e incluye la opción de comparar el texto a preprocesar con otro texto. Tras seleccionar los datos y pulsar el botón `Preprocesar`, se entra en una vista que refleja el estado de la ejecución y avisa al usuario cuando termina. Este proceso se sigue en todas las funcionalidades del programa, exceptuando `Probar detección de idioma`.

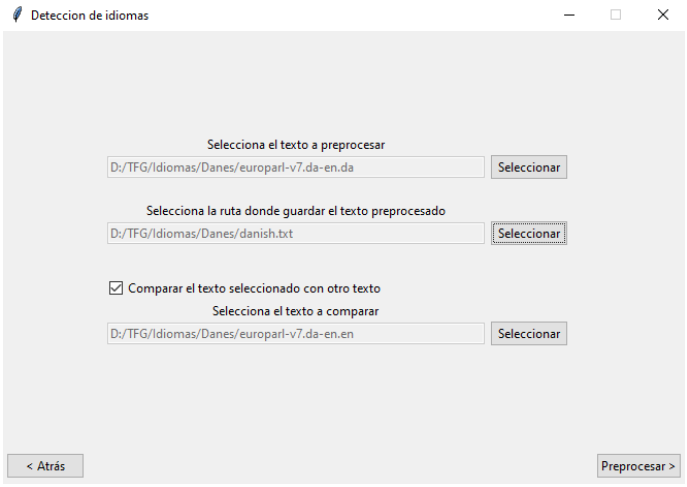

Figura 4.2: Vista del preprocesado de texto

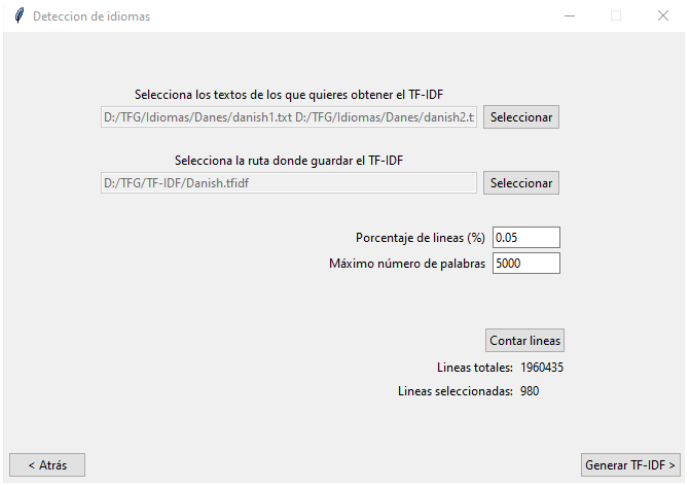

Figura 4.3: Vista del generador de tablas de TF-IDF

Generar TF-IDF

La vista para generar tabla TF-IDF, que se muestra en la Figura 4.3, cuenta con botones para seleccionar los archivos con los que generar las tablas y la ruta donde guardar el objeto generado. Adicionalmente, dispone de campos en los que se indican el porcentaje de líneas que se extraerá de cada texto seleccionado y el máximo número de palabras (columnas) que almacenará la tabla TF-IDF. El botón `Contar líneas` muestra el número exacto de líneas (filas) que tendrá la tabla TF-IDF.

Construir dataset

La vista de construcción de *datasets*, como se muestra en la Figura 4.4, es similar a la vista anterior. En este caso, se seleccionan los TF-IDF con los que construir el *dataset*. Nuevamente, se ajusta el número máximo de palabras (columnas) del *dataset* y además se indica el porcentaje de datos del *dataset* que se dedicará al entrenamiento de modelos.

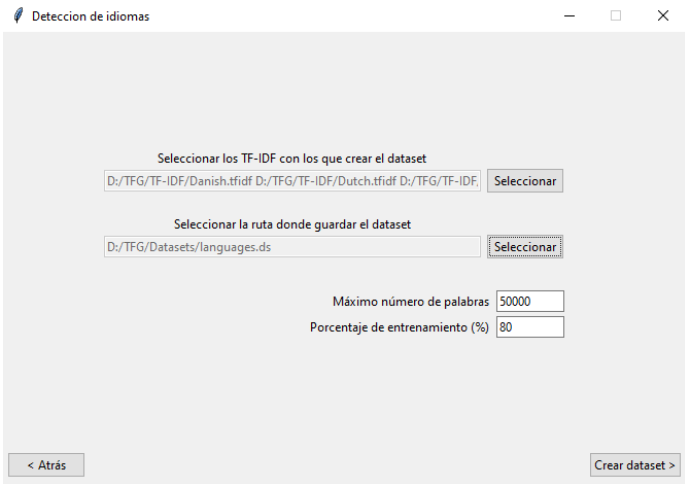

Figura 4.4: Vista de construcción del dataset

Entrenar y guardar modelo

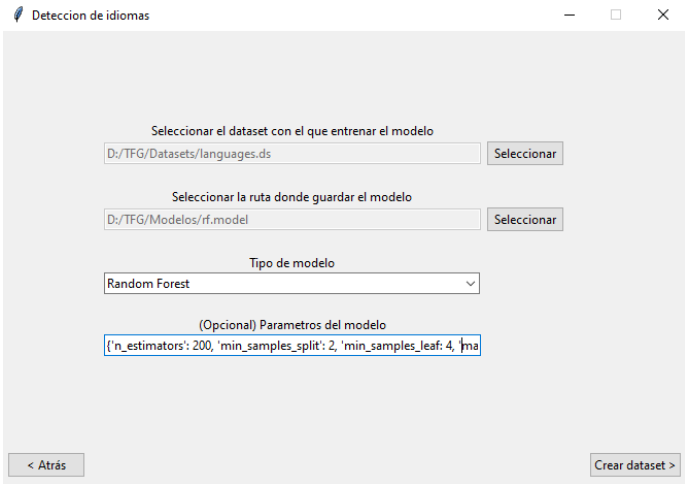

Figura 4.5: Vista del entrenamiento del modelo

En esta vista, que se muestra en la Figura 4.5, se selecciona el *dataset* con el que se entrenará el tipo de modelo escogido. De manera opcional, se pueden configurar los parámetros del algoritmo a través del campo de texto inferior mediante una cadena que representa un diccionario de Python. La cadena debe tener el siguiente formato:

```
{''nombre_parametro1'': valor_parametro1, ''nombre_parametro2'':
valor_parametro2, ...}
```

Probar detección de idioma

La funcionalidad de la vista mostrada en la Figura 4.6 permite probar la aplicación de usuario cargando distintos modelos. En la vista de selección hay dos opciones: usar el modelo por defecto cargado actualmente en la aplicación de usuario o seleccionar otro

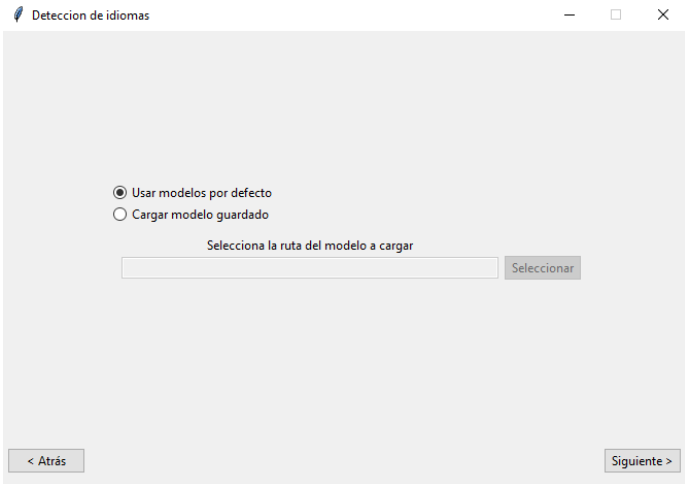

Figura 4.6: Vista de detección de idioma (selección)

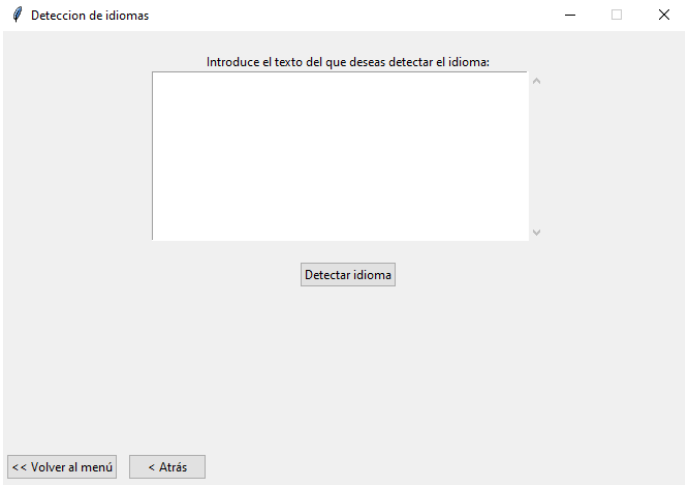

Figura 4.7: Vista de detección de idioma (ejecución)

modelo entrenado por el programa. Para cambiar el modelo por defecto, basta con sustituir el archivo `default.model` en la carpeta `default` por otro con el mismo nombre.

app.py

La aplicación de usuario dispone únicamente de la vista mostrada en la figura 4.7 cargada con el modelo por defecto. En la figura 4.8 puede verse un ejemplo de ejecución de la aplicación.

4.5. Resultado del proceso

Con la interfaz de usuario y la funcionalidad implementada en el capítulo anterior, ya se dispone de la versión completa del programa, que se utilizará en los siguientes capítulos para la realización de los experimentos. En la figura 4.9 puede verse el directorio completo

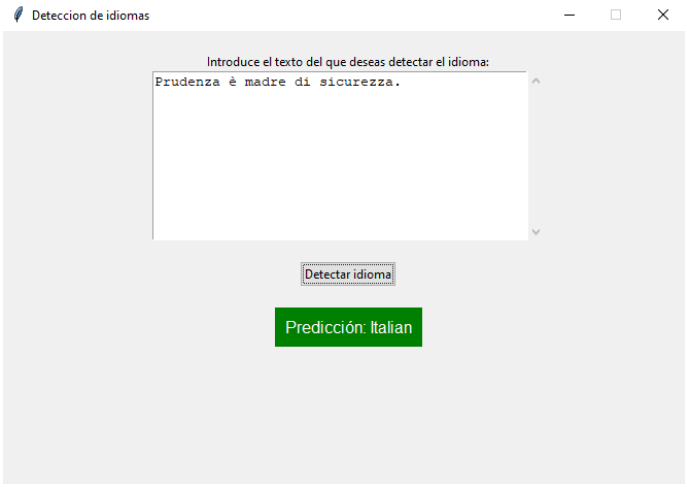

Figura 4.8: Vista de detección de idioma (usuario)

del programa.

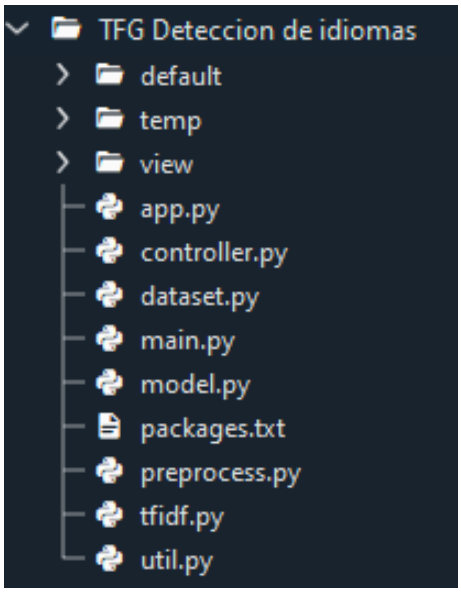

Figura 4.9: Directorio del programa completo

Capítulo 5

Modelos

"Entonces el Señor dijo: «Todos forman un solo pueblo y hablan un solo idioma; esto es solo el comienzo de sus obras y todo lo que se propongan lo podrán lograr»."
— Génesis 11:6

El en contexto de la inteligencia artificial, los modelos de Aprendizaje Automático son fundamentales en la resolución de problemas. En concreto, los modelos escogidos en este proyecto están enfocados al problema de la clasificación, es decir, determinar a que categoría pertenece un objeto. Los modelos empleados determinan la categoría (idioma) de los objetos en el universo escogido (palabras dentro de textos).

Los tres modelos de clasificación que se emplean en la fase de experimentos son el *Random Forest Classifier* (en adelante RFC), *K-Neighbors Classifier* (en adelante KNC) y *Support Vector Classifier* (en adelante SV). A continuación se tratará brevemente su funcionamiento y diferencias entre ellos.

Toda la información contenida en este capítulo ha sido extraída y recopilada de la documentación oficial de la librería de Python `scikit-learn` (Pedregosa et al., 2011).

5.1. Random Forest Classifier

Este modelo emplea técnicas de conjunto, basadas en el concepto de "bosques aleatorios"[1], lo que proporciona predicciones bastante precisas y estables.

Se emplea la técnica conocida como *bagging*, o embolsado, para reducir el sobreajuste (tendencia de los modelos a ajustarse a los datos de entrenamiento y responder peor ante casos reales) del modelo y mejorar su generalización. Esta técnica funciona especialmente bien al combinarse con la selección aleatoria de características en los árboles de decisión.

[1] Un bosque aleatorio es un conjunto de árboles de decisión en los que el orden de sus nodos, o preguntas, es aleatorio.

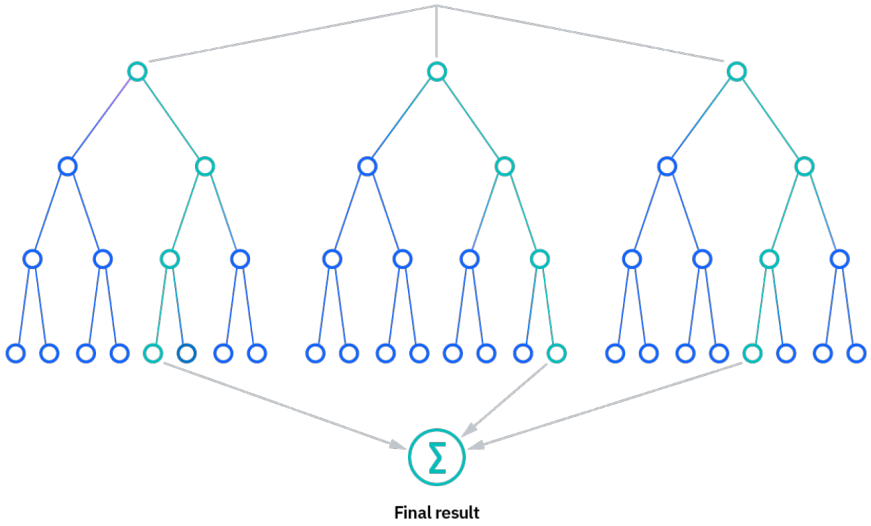

Figura 5.1: Bosque aleatorio[2]

Para este modelo, los árboles de decisión del bosque se entrenan de forma separada con grupos de datos de entrenamiento y características diferentes. Los árboles entonces toman decisiones para cada subconjunto basadas en las características más relevantes encontradas hasta ese momento. Finalmente, se combinan todas las predicciones para acabar eligiendo una de ellas.

Entre los principales puntos fuertes del RFC destaca la capacidad de gestionar datos heterogéneos de gran tamaño, mientras se evita llegar al problema del sobreajuste, que impide generalizar a los modelos de aprendizaje automático. Esto se consigue al entrenar los árboles de forma separada, ya que se reduce depender de las características concretas de cada caso y minimiza el sesgo del modelo entrenado. También se reduce la varianza, aunque en este caso, por la propia naturaleza aleatoria de los textos esto no supone un problema. Las palabras que más representan a un idioma suelen ser las preposiciones y los pronombres, ligados intrínsecamente a la gramática de ese lenguaje.

5.2. K-nearest Neighors Classifier

Este algoritmo se basa en un enfoque de aprendizaje denominado "aprendizaje basado en instancias" o "aprendizaje perezoso". El algoritmo clasifica una nueva instancia según la mayoría de las clases de sus vecinos más cercanos en el espacio de características. En lugar de construir un modelo explícito durante la fase de entrenamiento, almacena los datos de entrenamiento en una estructura llamada árbol de búsqueda k-d (*k-dimensional tree*) o en un índice de vecinos más cercanos (*k-d tree* o *k-nearest neighbors index*).

[2]Fuente: *https://www.ibm.com/topics/random-forest*

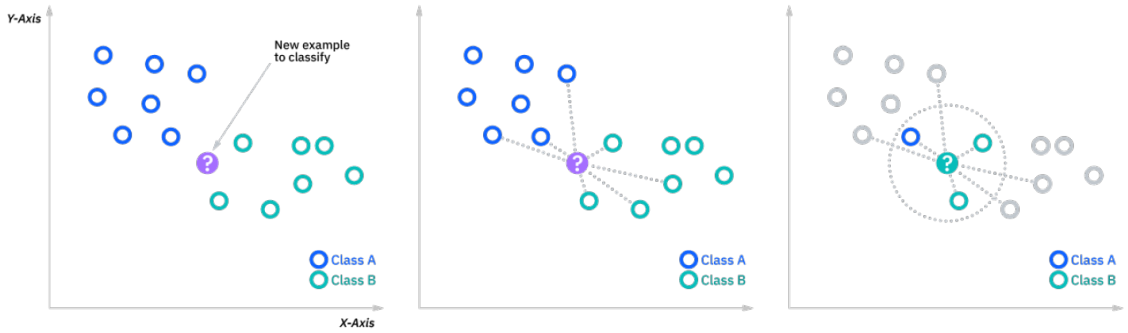

Figura 5.2: Espacio de clasificación[3]

Cuando se recibe un punto de datos desconocido para clasificar, el algoritmo encuentra los k puntos de datos más cercanos (según la distancia euclidiana) en el espacio de características, utilizando el árbol de búsqueda k-d o el índice de vecinos más cercanos. Luego, asigna al punto desconocido la clase más común entre sus vecinos cercanos (votación mayoritaria).

$$d(x,y) = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2}$$

Figura 5.3: Distancia euclidiana

Una de las ventajas clave del algoritmo es su capacidad para manejar conjuntos de datos con estructuras complejas y no linealmente separables. Además, no asume ninguna distribución particular de los datos, lo que lo hace útil en una amplia variedad de escenarios.

5.3. Support Vector Classifier

El algoritmo Support Vector Classifier (SVM), se basa en el concepto de máquinas de vectores de soporte y es ampliamente utilizado en problemas de clasificación binaria y multiclase. Con su enfoque en la maximización del margen de separación entre las clases, SVC destaca por su capacidad para lidiar con conjuntos de datos no linealmente separables mediante la aplicación de transformaciones no lineales a través de funciones de kernel.

La idea principal detrás del SVC es encontrar el hiperplano óptimo en un espacio de características de alta dimensión que maximice el margen de separación entre las clases. Un hiperplano es una superficie que divide el espacio de características en diferentes regiones correspondientes a las diferentes clases.

[3]Fuente: *https://www.ibm.com/topics/knn*

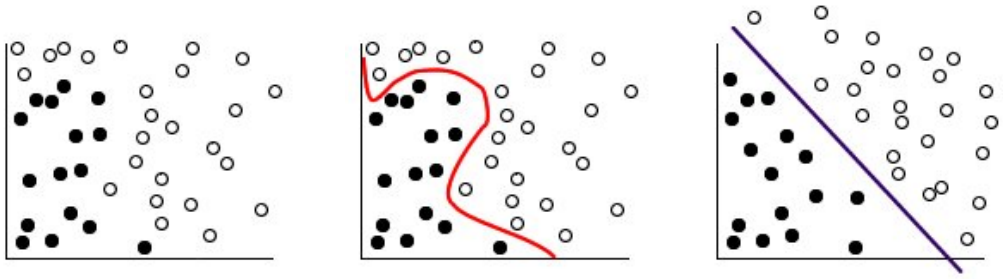

Figura 5.4: Conjuntos de características[4]

En el caso de conjuntos de datos linealmente separables, el SVC busca encontrar el hiperplano que logre la separación máxima entre las clases. Sin embargo, en muchos casos, los datos pueden no ser linealmente separables en el espacio de características. Para abordar esto, el SVC utiliza transformaciones no lineales a través de funciones de kernel para mapear los datos a un espacio de características de mayor dimensión donde puedan ser separables linealmente.

El SVC utiliza vectores de soporte, que son los puntos de datos más cercanos al hiperplano de separación, para determinar la ubicación y orientación del hiperplano. Estos vectores de soporte son fundamentales en el proceso de clasificación, ya que definen el margen de separación y juegan un papel crucial en la toma de decisiones.

[4]Fuente: *https://www.ibm.com/docs/en/spss-modeler/saas?topic=models-how-svm-works*

Capítulo 6

Métodos de Evaluación

"«Será mejor que bajemos a confundir su idioma para que ya no se entiendan entre ellos mismos»."
— Génesis 11:7

En el presente capítulo se explican los experimentos a realizar y se definen los parámetros y métricas de los mismos. Se comienza dando contexto sobre el proceso de experimentación y luego se explica cada parámetro y métrica aplicados.

6.1. Procedimiento

Inicialmente se decide cuales van a ser las métricas de evaluación y se construyen 4 *datasets* diferentes para realizar las pruebas con cada modelo. Estos se crean a partir de las tablas TF-IDF creadas con los textos preprocesados de cada idioma incluido en el *dataset*.

Para la realización de los experimentos, se va a utilizar el método de búsqueda aleatoria con validación cruzada (*Cross-Validation Randomized Search* en inglés). Este método permite probar diferentes combinaciones de parámetros, dando gran cantidad de resultados y facilitando el proceso de encontrar los valores más óptimos.

En este tipo de búsqueda se emplea el método de validación cruzada de K-iteraciones (*K-Fold* en inglés) que consiste en dividir el *dataset* en `K` subconjuntos, donde `K-1` son dedicados al entrenamiento del modelo y el restante se usa como subconjunto de prueba. Este proceso se ejecuta K veces, cambiando en cada iteración los subconjuntos de entrenamiento y el subconjunto de prueba.

La búsqueda aleatoria se encarga de generar combinaciones de parámetros, configurar modelos con esas combinaciones y hacer una validación cruzada con cada modelo. Las combinaciones se generan a partir de una cuadrícula (*grid* en inglés) que establece los parámetros a probar y los valores que puede tomar cada uno de ellos.

6.2. Configuración de la búsqueda aleatoria

La búsqueda aleatoria se ha implementado mediante la clase `RandomizedSearchCV` de la librería `scikit-learn` en el archivo `optimization.py` de la carpeta `test`. Este archivo contiene la siguiente configuración utilizada para los experimentos:

- `Grids`: se ha creado una cuadrícula de parámetros y valores para cada modelo.
- `K`: el número de subconjuntos se ha establecido en 5.
- `Combinaciones`: la búsqueda aleatoria genera 100 combinaciones de parámetros.

Con esta configuración, por cada *dataset* se hacen 1500 ejecuciones: 300 combinaciones (100 por cada modelo) que se iteran 5 veces. Por tanto, los experimentos tienen un total de 6000 ejecuciones.

6.3. Datasets

Para la realización de los experimentos se han creado 4 *datasets* de diferente tamaño y composición mediante el programa explicado en los Capítulos 2 y 3.

Los 4 *datasets* contienen una clasificación de 9 idiomas. Todos los idiomas son de la gran familia indoeuropea y pertenecen a 3 familias diferentes:

- `Gérmanicas`: sueco, neerlandés y danés.
- `Romances`: portugués, rumano e italiano.
- `Eslavas`: eslovaco, esloveno y polaco.

De los textos preprocesados de cada idioma se han extraído aleatoriamente un número diferente de líneas para crear tablas TF-IDF con ellas. Los *datasets* creados mediante estas tablas se identifican mediante letras griegas y son los siguientes:

- Alfa (α):
  - Número de líneas totales: 450
  - Número de líneas *train*: 360 (80 %)
  - Número de líneas *test*: 90 (20 %)
  - Número de palabras: 5260
- Beta (β):
  - Número de líneas totales: 1088
  - Número de líneas *train*: 870 (80 %)
  - Número de líneas *test*: 218 (20 %)

    - Número de palabras: 10398

- Gamma (γ):
    - Número de líneas totales: 1725
    - Número de líneas *train*: 1380 (80 %)
    - Número de líneas *test*: 345 (20 %)
    - Número de palabras: 14827

- Delta (δ):
    - Número de líneas totales: 2286
    - Número de líneas *train*: 1828 (80 %)
    - Número de líneas *test*: 458 (20 %)
    - Número de palabras: 18278

6.4. Parámetros

Los parámetros son aquellas variables de configuración que pueden cambiar entre diferentes entrenamientos de los modelos. Como dependen del funcionamiento de cada uno de ellos, a continuación se explican en el entorno de su modelo.

6.4.1. Random Forest Classifier

- `n_estimators`: Número de árboles de decisión que se construirán en el bosque aleatorio. Un valor más alto puede mejorar la precisión del modelo, pero también aumentar el tiempo de entrenamiento y la complejidad.

- `min_samples_split`: Número mínimo de muestras necesarias para dividir un nodo interno. Este valor puede ayudar a controlar la complejidad del modelo y prevenir el sobreajuste. Si es int, entonces considera min_samples_split como el número mínimo. Si es float, entonces min_samples_split es una fracción y ceil(min_samples_split * n_samples) son el número mínimo de muestras para cada división.

- `min_samples_leaf`: Número mínimo de muestras que se requieren para estar en una hoja (leaf) del árbol. Un valor más alto puede ayudar a prevenir el sobreajuste, pero también puede reducir la capacidad del modelo para encontrar patrones en los datos. Si es int, entonces considera min_samples_leaf como el número mínimo. Si es float, entonces min_samples_leaf es una fracción y ceil(min_samples_leaf * n_samples) son el número mínimo de muestras para cada nodo.

- `max_depth`: La profundidad máxima del árbol. Si es None, los nodos se expanden hasta que todas las hojas sean puras o hasta que todas las hojas contengan menos muestras que min_samples_split.

- `bootstrap`: Indicador booleano que indica si se utilizará la técnica de *bootstrap*[1] durante la construcción del bosque. Esta técnica puede mejorar la estabilidad del modelo y reducir la varianza.

6.4.2. K-Nearest Neighbors Classifier

- `weights`: Pesos que se utilizan para la clasificación de los vecinos cercanos. Puede ser uno de los siguientes valores:

  `uniform`: Todos los vecinos tienen el mismo peso.

  `distance`: La influencia de cada vecino se calcula como el inverso de su distancia. Los vecinos más cercanos tienen un peso mayor que los vecinos más lejanos.

  Se define una función personalizada de pesos, que toma la distancia como entrada y devuelve un peso. Para hacer esto, se debe proporcionar una función que toma una matriz de distancia y devuelve una matriz de pesos. En este proyecto no se ha definido ninguna función adicional.

- `p`: El exponente utilizado en la métrica de Minkowski. Cuando p = 1, esto equivale a utilizar la distancia de Manhattan. Cuando p = 2, esto equivale a utilizar la distancia euclídea. Para un valor arbitrario se utiliza por defecto la métrica de Minkowski.

- `n_neighbors`: Número de vecinos que se utilizarán para la clasificación de un punto del conjunto. Este es un valor entero positivo.

- `algorithm`: Algoritmo utilizado para calcular los vecinos más cercanos. Puede ser uno de los siguientes valores:

  `auto`: Intentará decidir el algoritmo más apropiado basándose en los valores pasados al método fit.

  `ball_tree`: Se utiliza una estructura de árbol de bolas para buscar los vecinos más cercanos.

  `kd_tree`: Se utiliza una estructura de árbol (k-dimensional) para buscar los vecinos más cercanos.

  `brute`: Se utiliza una búsqueda de fuerza bruta para encontrar los vecinos más cercanos.

- `leaf_size`: Tamaño de hoja pasado a `algorithm` cuando vale `ball_tree` o `kd_tree`. Esto puede afectar a la velocidad de construcción y consulta, así como a la memoria necesaria para almacenar el árbol. El valor óptimo depende de la naturaleza del problema.

[1] *Bootstrap consiste en tomar repetidas muestras del conjunto de entrenamiento y sustituirlas en el árbol actual según sea necesario.*

6.4.3. Support Vector Classifier

- `shrinking`: Parámetro que controla si se utiliza o no la reducción del tamaño del conjunto de vectores de soporte. Si está configurado a `true` la eliminación de vectores de soporte se utiliza para acelerar el entrenamiento y mejorar la eficiencia en la predicción. Si está configurado en `false` no se utiliza la eliminación de vectores de soporte.

- `kernel`: Función kernel utilizada para transformar los datos en un espacio de características de mayor dimensión. Las funciones kernel disponibles son `linear`, `poly`, `rbf` y `sigmoid`.

- `gamma`: Parámetro de `kernel` que controla la forma de la función de decisión. Los valores `scale` y `auto` se refieren a la estrategia de cálculo de `gamma`.

- `degree`: Grado del polinomio utilizado por la función kernel `poly`. Este parámetro solo se considera si `kernel` es `poly`.

- `coef0`: Parámetro de `kernel` que se usa en algunas funciones kernel para ajustar la influencia de los términos de orden superior en la función de decisión. Solo es significativo en "poly" o "sigmoid".

- `C`: Parámetro de regularización que controla el equilibrio entre ajustar el modelo a los datos de entrenamiento y evitar el sobreajuste. Este valor tiene que ser estrictamente positivo.

6.5. Métricas

Una vez establecidos los parámetros para cada modelo y realizadas las ejecuciones en todos los *datasets*, los valores a medir son la precisión y la rapidez de ejecutar una consulta de una palabra sobre un modelo.

La clase `RandomizedSearchCV` guarda cada combinación de parámetros probada junto a las siguientes métricas:

- `mean_test_score`: Puntuación media de validación cruzada para las muestras del conjunto *test*. Es decir, mide el índice de aciertos para esa configuración de ese modelo en el *dataset*.

- `mean_score_time`: Tiempo medio requerido para predecir las etiquetas para las muestras del conjunto *test*.

El archivo `optimization.py` recoge todos estos datos y los guarda en archivos con extensión `.xlsx`, en formato de tablas.

6.6. Resultado del proceso

Una vez realizados todos los experimentos, se dispone de 12 tablas en archivos separados (cada una resultado de ejecutar uno de los 3 modelos sobre uno de los 4 *datasets*) con extensión `.xlsx` con las mediciones para cada combinación de parámetros. En el siguiente capítulo se presentarán los experimentos de la forma más clara posible para permitir encontrar patrones y tendencias en los resultados de los experimentos.

Capítulo 7

Resultados

"De esta manera el Señor los dispersó desde allí por toda la tierra; por lo tanto, dejaron de construir la ciudad."
— Génesis 11:8

En este capítulo se exponen finalmente los resultados obtenidos de las pruebas realizadas con los modelos y *datasets* descritos anteriormente. Se comienza por una comparativa dentro de cada modelo de la variación de las métricas en función de los parámetros, y al final del capítulo se incluyen las gráficas que comparan los modelos entre sí.

7.1. Comparativa basada en parámetros

El resultado final de los experimentos consiste en una serie de archivos `.xlsx` con los diferentes valores de precisión y tiempo para cada modelo en cada *dataset* junto a la combinación de valores para los parámetros empleados que se utilizaron en cada ejecución o prueba[1]. Con esto es posible elaborar gráficas de puntos de dispersión en función de los parámetros. Se dispone, pues, de 12 tablas con 100 experimentos en cada una, haciendo un total de 1200 experimentos.

En cada gráfica mostrada a continuación se pueden ver en el eje x los 4 *datasets* (alfa, beta, gamma y delta), mientras que en el eje y figura o bien la precisión media del modelo en ese experimento o el tiempo (en segundos) medio que ha durado su ejecución. Los puntos dentro de la gráfica son todos los experimentos realizados con ese modelo en los *datasets*, y su color corresponde al valor concreto del parámetro analizado.

Por último, es importante añadir que algunos parámetros son a su vez subparámetros de otros (por ejemplo, `degree` de Support Vector es un subparámetro de `kernel` cuando este vale `poly`), por lo que las gráficas correspondientes solo muestran los experimentos en los que dicho parámetro tiene los valores que aceptan al subparámetro analizado (de nuevo, como ejemplo, la gráfica del parámetro `degree` contiene únicamente las ocurrencias en las que el valor de `kernel` es `poly`).

[1] Las tablas de resultados de cada experimento están incluidas íntegramente en el Apéndice A.

7.1.1. Comparativa de los parámetros de Random Forest

A continuación se muestran las gráficas que comparan la eficiencia del modelo de Random Forest para cada parámetro (estos parámetros se describen en el Capítulo 6. Métodos de Evaluación).

Número de estimadores (`n_estimators`)

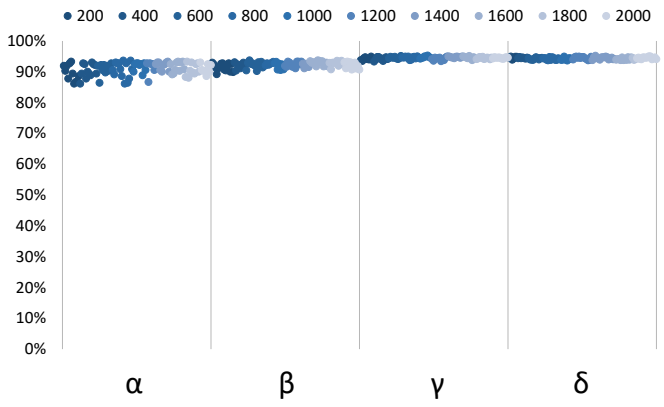

Figura 7.1: Precisión de Random Forest en función de `n_estimators`

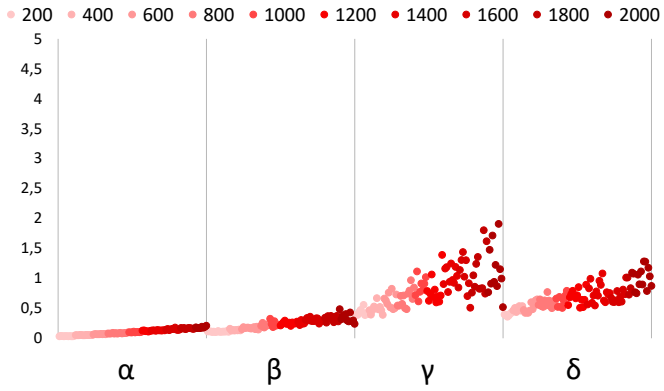

Figura 7.2: Tiempo de Random Forest en función de `n_estimators`

Se observa una tendencia clara en la precisión medida. A medida que aumenta el tamaño del *dataset* empleado, se obtiene una mayor precisión en el modelo. Esto implica que un *dataset* mas grande ofrece una mayor consistencia en los resultados y una mejora en la calidad del modelo generado. Sin embargo, no se distingue diferencia apreciable en función del número de estimadores, por lo que para estos conjuntos de palabras y textos este parámetro no tiene efecto.

Se puede ver que a medida que aumenta el tamaño de los datos aumenta el tiempo medio. El número de estimadores también empuja los tiempos a valores más altos de forma lineal. Sin embargo, el *dataset* gamma, aunque más pequeño que delta, tiene algunos

resultados de mayor tiempo. Esto puede implicar que esta métrica varia en función de la complejidad del conjunto de datos y no solo del peso o tamaño.

Tamaño máximo de corte (`min_samples_split`)

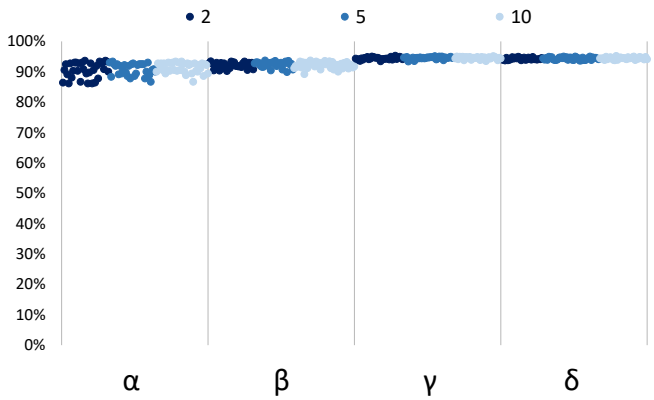

Figura 7.3: Precisión de Random Forest en función de `min_samples_split`

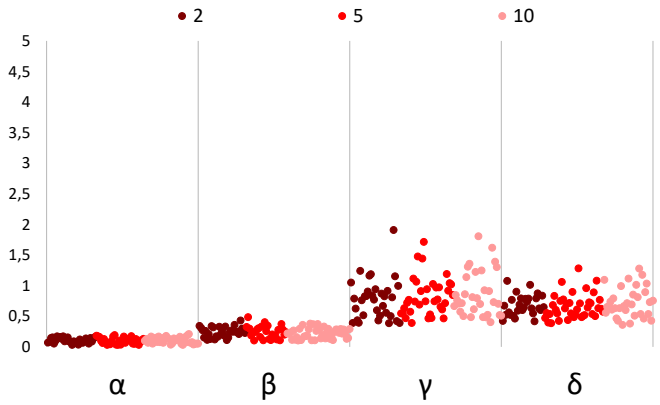

Figura 7.4: Tiempo de Random Forest en función de `min_samples_split`

Como en el parámetro anterior, no se aprecia gran diferencia en la métrica de precisión, salvo que un *dataset* mayor otorga mayor precisión al modelo.

Tampoco parece que el tamaño máximo de corte afecte a los tiempos del programa, aunque se observa que al aumentar el tamaño del *dataset* también aumenta la dispersión. Es posible que esto se deba a la típica diferencia entre casos mejores y peores a la hora de predecir los resultados, que con datos más grandes se hace más notable.

Tamaño mínimo de hoja (`min_samples_leaf`)

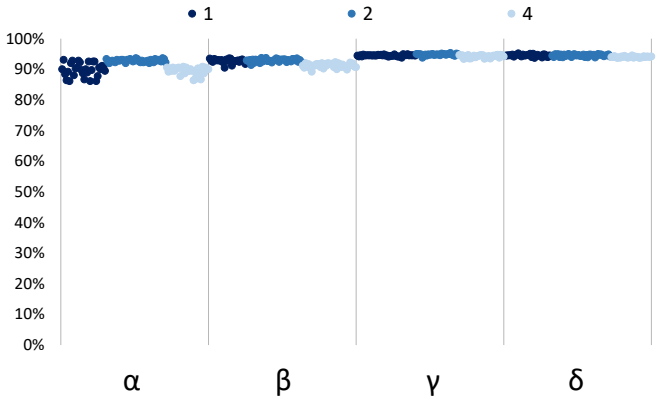

Figura 7.5: Precisión de Random Forest en función de `min_samples_leaf`

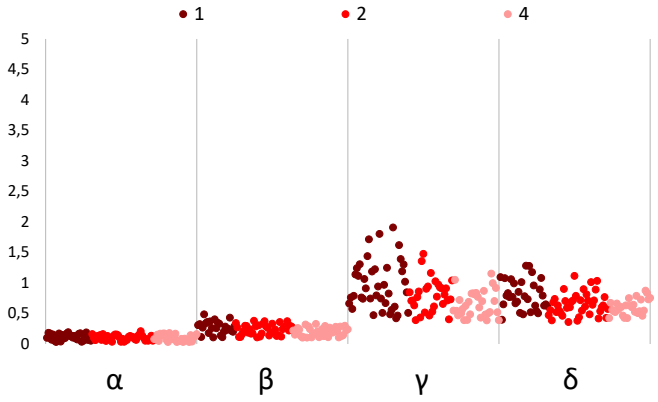

Figura 7.6: Tiempo de Random Forest en función de `min_samples_leaf`

Aquí comienza a observarse una diferencia ente los valores que toman los parámetros. En este caso, vemos que los valores de precisión más altos y más consistentes se alcanzan cuando el tamaño mínimo de hoja es 2. Sin embargo, esto solo parece afectar a la precisión cuanto el tamaño de los datos es menor, ya que para los *datasets* gamma y delta no se aprecia diferencia.

En los tiempos la situación cambia radicalmente. Si bien con *datasets* menores no se aprecia la diferencia, existe un patrón que liga el tamaño mínimo de hoja con el tiempo del algoritmo de forma inversamente proporcional. Es decir, según aumenta ese tamaño, el algoritmo tarda menos (de media, recordemos) es terminar su ejecución. No parece que los tiempos puedan bajar mucho más, pero es posible que usando hojas de tamaño mayor se alcancen unos tiempos menores y más consistentes (aunque solo para conjuntos de documentos de mayor tamaño y complejidad).

Profundidad máxima del árbol (`max_depth`)

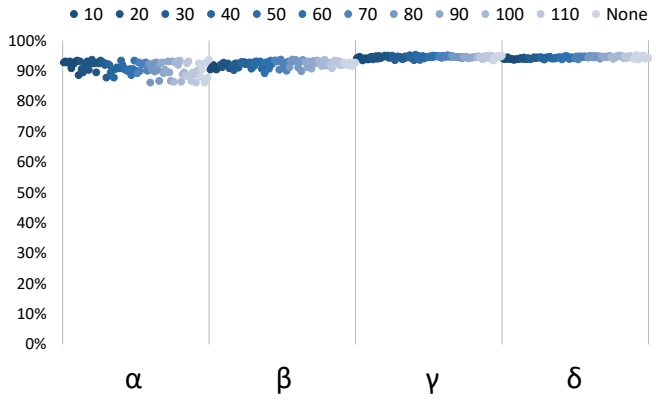

Figura 7.7: Precisión de Random Forest en función de `max_depth`

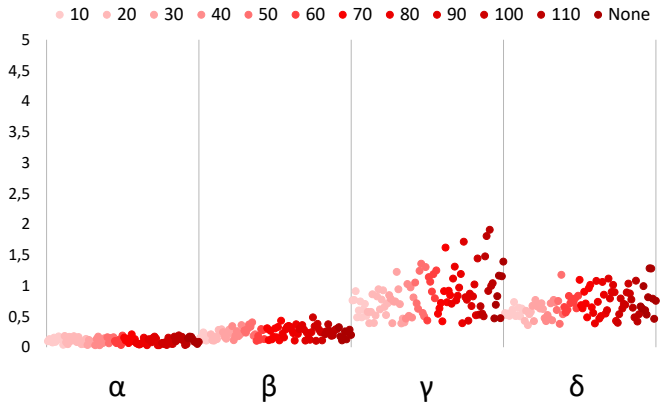

Figura 7.8: Tiempo de Random Forest en función de `max_depth`

Para la métrica de precisión la situación es casi idéntica a los anteriores parámetros: cuanto mayor es el tamaño del *dataset*, mayor precisión se alcanza. Sin embargo, no parece la precisión varíe en función de la profundidad máxima de los árboles.

Para el tiempo medio sí que parece existir un patrón similar al de la gráfica de tiempos anterior, si bien en este caso al aumentar la profundidad aumenta el tiempo y la dispersión (de nuevo, solo para los *datasets* de mayor tamaño).

Re-muestreo con remplazo (`bootstrap`)

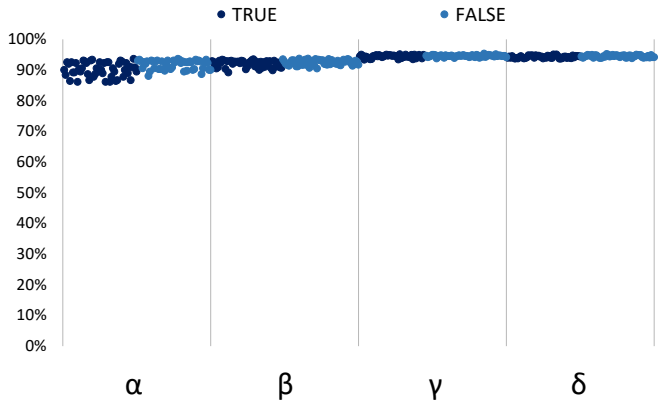

Figura 7.9: Precisión de Random Forest en función de `bootstrap`

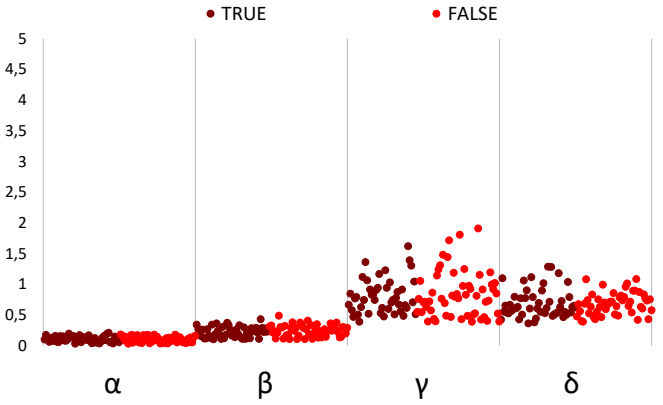

Figura 7.10: Tiempo de Random Forest en función de `bootstrap`

Para este parámetro si parece haber una diferencia en función de si se utiliza la técnica de re-muestreo con remplazo o no, aunque es pequeña y harían falta más experimentos para confirmar si estas diferencias son casuales o realmente dependen del parámetro `bootstrap`.

En los tiempos esta vez no parece haber diferencia notable, únicamente lo que ya pudo confirmarse: a mayor tamaño de *dataset*, más tiempo se tarda en terminar la ejecución del modelo (los puntos más altos en el *dataset* gamma pueden deberse más a la diferencia en complejidad con delta que a otros factores, como ya se vio en otras gráficas).

7.1.2. Comparativa de los parámetros de K-Nearest Neighbors

A continuación se muestran las gráficas que comparan la eficiencia del modelo de K-Nearest Neighbor para cada parámetro.

Función de peso (`weights`)

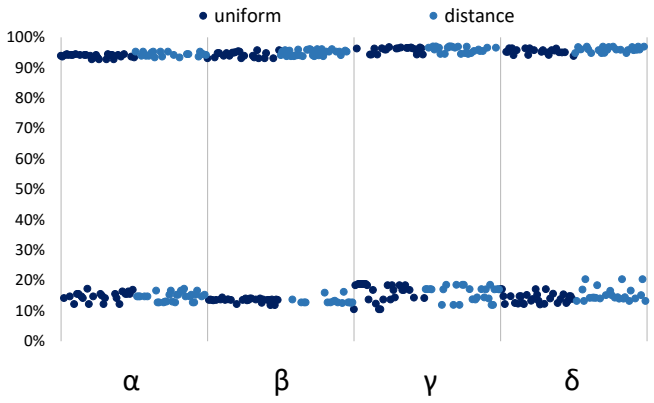

Figura 7.11: Precisión de K-Nearest Neighbors en función de `weights`

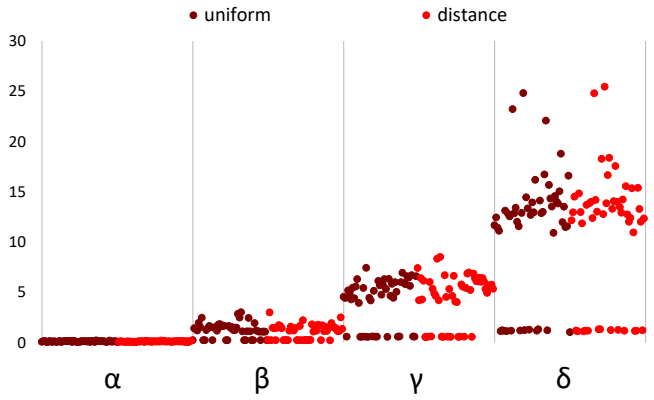

Figura 7.12: Tiempo de K-Nearest Neighbors en función de `weights`

En las gráficas del modelo K-Nearest Neighbors se detecta una distancia considerable entre los diferentes experimentos, tanto en precisión como en tiempo. Como se verá más adelante, esto se debe a otro parámetro, por lo que se analizará únicamente la parte alta de la gráfica.

No parece que la función de peso afecte a la precisión del modelo ni al tiempo. En ambas gráficas la distribución de los puntos es totalmente simétrica.

p de Minkowski (p)

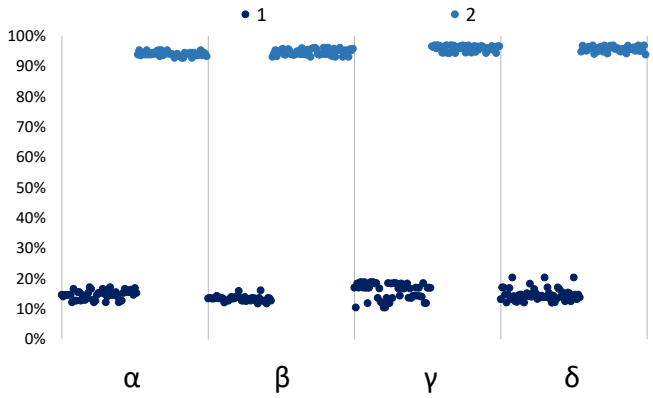

Figura 7.13: Precisión de K-Nearest Neighbors en función de p

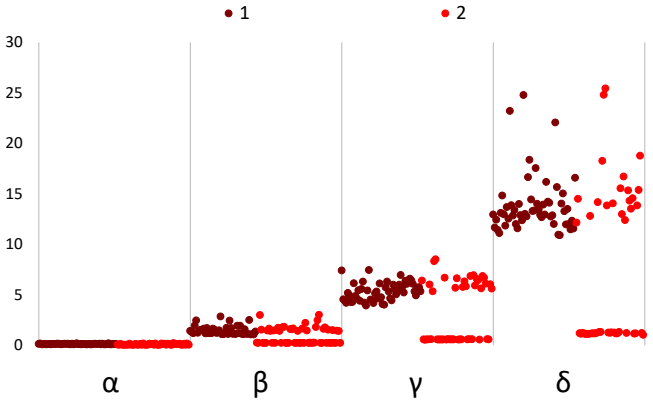

Figura 7.14: Tiempo de K-Nearest Neighbors en función de p

La p de Minkowski afecta de manera considerable a la precisión del algoritmo. Recordemos que este parámetro lo que indica es si la función de distancia empleada para detectar los vecinos más cercanos a un punto introducido en el universo del conjunto utiliza la distancia de Manhattan (p=*1*) o la distancia euclidiana (p=*2*). Se ve de forma directa que los valores de precisión aceptables del algoritmo aparecen cuando se emplea esta segunda fórmula de distancia.

En la gráfica de tiempo sucede una cosa curiosa. Si bien parece que la p de Minkowski no afecta al tiempo registrado, cuando se emplea la distancia de Manhattan la consistencia de los resultados aumenta, mientras que con la euclidiana en ocasiones tenemos tiempo muy altos (en torno a los 15 segundos) y a veces muy bajos (cercanos a 1 segundo). Esta diferencia corresponde a otro parámetro, que se explicará más adelante.

Número de vecinos (`n_neighbors`)

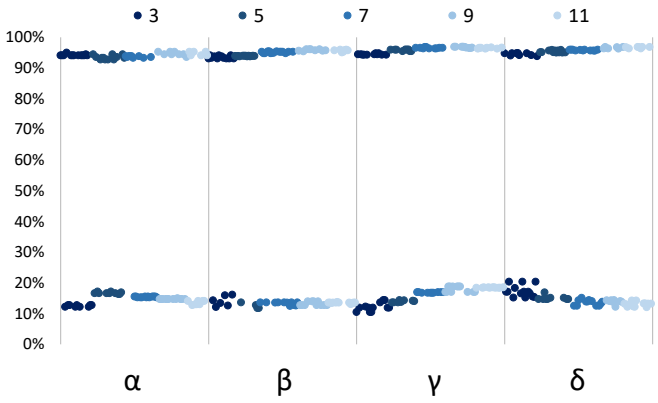

Figura 7.15: Precisión de K-Nearest Neighbors en función de `n_neighbors`

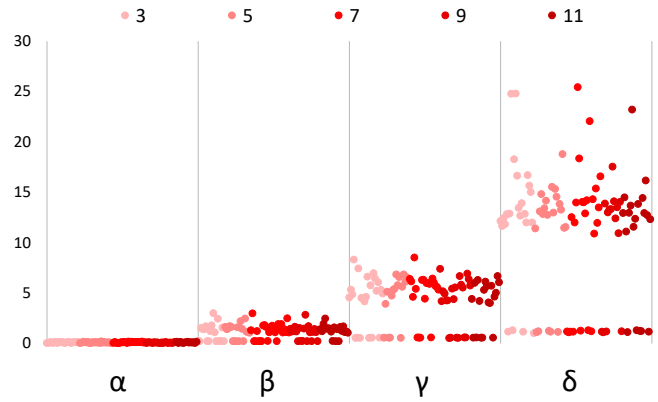

Figura 7.16: Tiempo de K-Nearest Neighbors en función de `n_neighbors`

El número de vecinos afecta positivamente a la precisión: según aumenta, también lo hace la certeza del modelo. Se observa que en el *dataset* alfa no se presenta en este patrón, es muy posible que se deba a anomalías de dispersión propias del algoritmo cuando maneja datos de poco tamaño.

La gráfica de tiempo es muy parecida a las anteriores, los tiempos no dependen del número de vecinos.

Algoritmo de cómputo (`algorithm`)

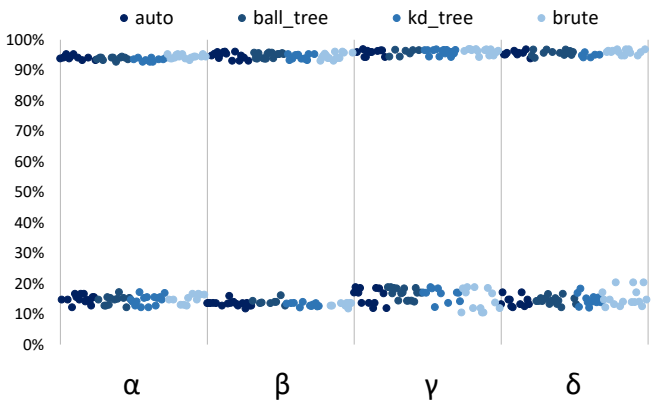

Figura 7.17: Precisión de K-Nearest Neighbors en función de `algorithm`

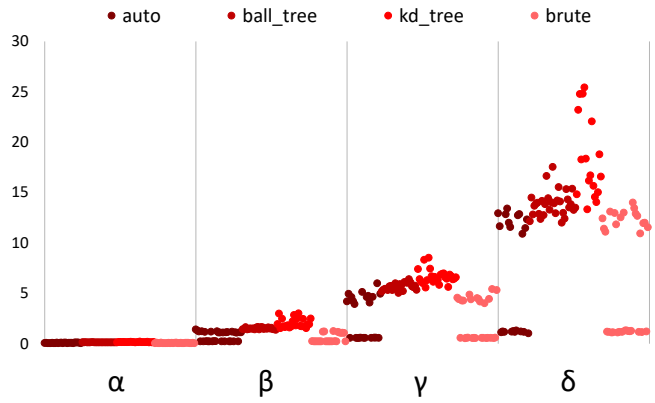

Figura 7.18: Tiempo de K-Nearest Neighbors en función de `algorithm`

El parámetro que le indica al modelo que algoritmo de cómputo utilizar para predecir los resultados es el que más afecta a los resultados en tiempo, pero no en la precisión. Recordemos que cuando el algoritmo se fija a automático (`algorithm=auto`), el modelo escoge entre los otros 3 el que considera da mejores resultados para ese experimento.

El algoritmo que más optimiza los tiempos es el de fuerza bruta, pero a costa de dar mayores tiempos en los casos peores, lo que no muestra una gran consistencia (los valores cuando el parámetro vale `auto` que corresponden a menores tiempos coincidirían con los resultados obtenidos con el algoritmo de fuerza bruta, pero no siempre). Sin embargo, incluso en caso peor, los tiempos del algoritmo de fuerza bruta son algo mejores que cuando se emplean los otros dos algoritmos, por lo que esta opción parece, a priori, la mejor elección.

Tamaño de hoja (`leaf_size`)

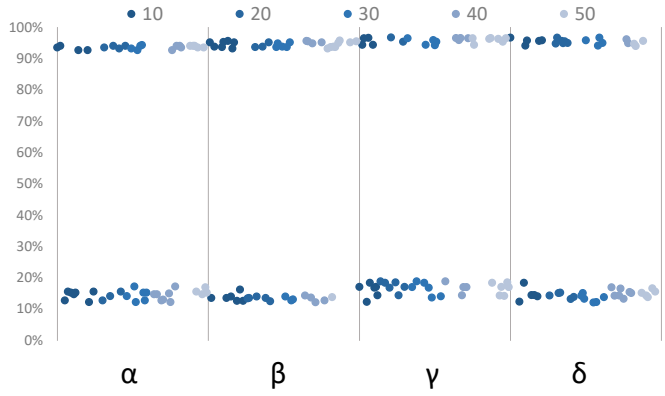

Figura 7.19: Precisión de K-Nearest Neighbors en función de `leaf_size`

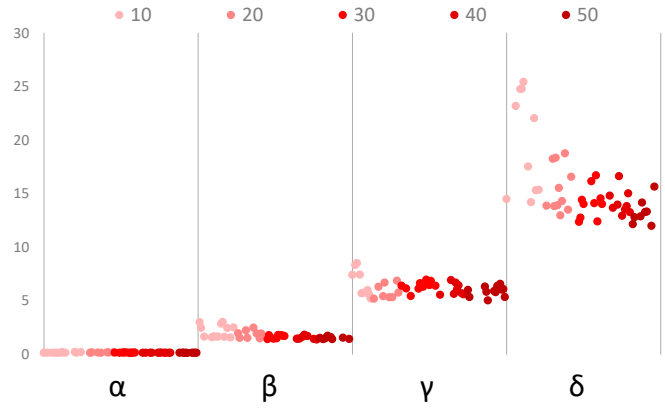

Figura 7.20: Tiempo de K-Nearest Neighbors en función de `leaf_size`

El tamaño de hoja solo se utiliza cuando el algoritmo de cómputo es o bien BallTree o bien KDTree, por lo que los puntos mostrados en la gráfica corresponden únicamente a los experimentos en los que el algoritmo de cómputo es uno de los dos.

No se observa en la gráfica de precisión una tendencia de los valores a cambiar en función del tamaño de hoja. Podría parecer, observando los huecos que quedan en algunos momentos, que para ciertos valores del parámetro `leaf_size` la p de Minkowski, que tanto afecta a las demás gráficas, no cambie los resultados, pero debido al menor número de puntos en la gráfica en comparación a las demás, es difícil sostener esta idea.

En la gráfica con los tiempos sí que parece haber un patrón. El tamaño del *dataset* aumenta el tiempo medio de los experimentos, pero si se aumenta el tamaño de hoja, se consigue disminuir sustancialmente la media de tiempo (y se consigue una menor dispersión).

7.1.3. Comparativa de los parámetros de Support Vector

A continuación se muestran las gráficas que comparan la eficiencia del modelo de Support Vector para cada parámetro.

Heurística de reducción (`shrinking`)

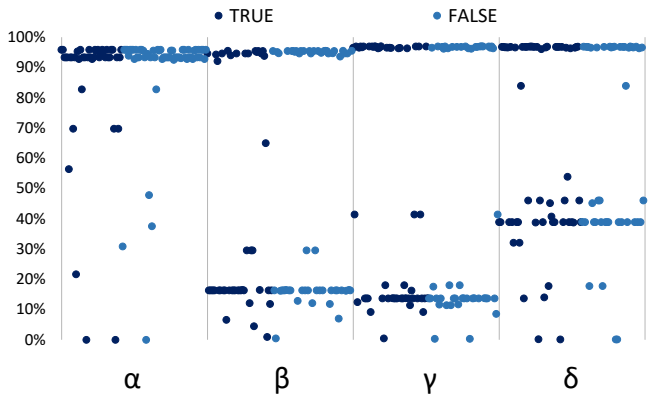

Figura 7.21: Precisión de Support Vector en función de `shrinking`

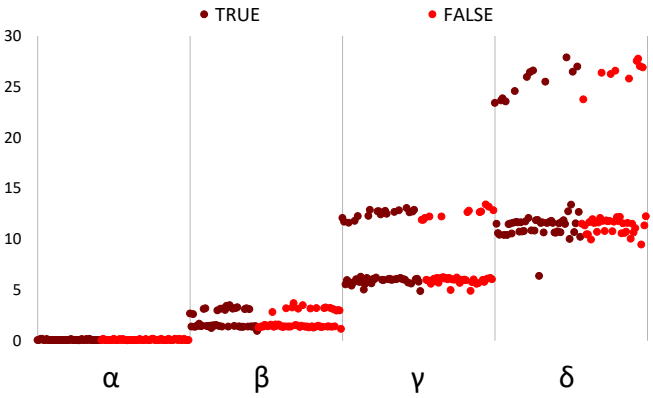

Figura 7.22: Tiempo de Support Vector en función de `shrinking`

La precisión no varía en función de la heurística, ni tampoco el tiempo. Como siempre, un tamaño mayor de los datos implica una mayor precisión pero también un mayor tiempo.

La alta diferencia en precisión de los puntos de la gráfica corresponde a otro parámetro, tratado más adelante. Lo mismo sucede con la gráfica de los tiempos, aunque como se verá a continuación, no se trata del mismo parámetro.

Núcleo del algoritmo (`kernel`)

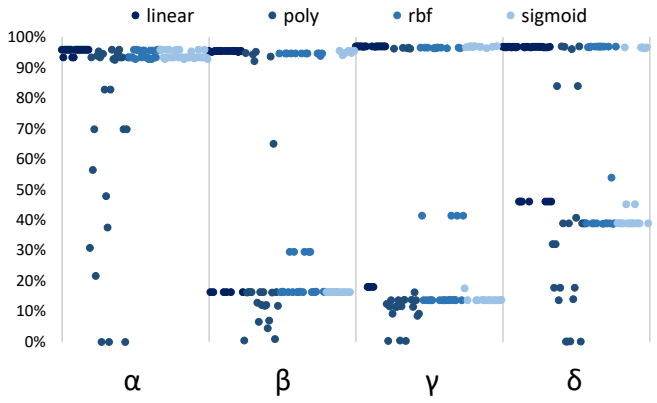

Figura 7.23: Precisión de Support Vector en función de `kernel`

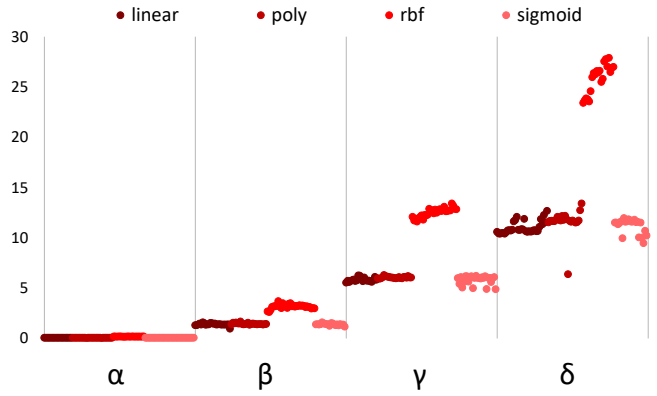

Figura 7.24: Tiempo de Support Vector en función de `kernel`

La gráfica de precisión es análoga a la del anterior parámetro, y de nuevo parece que la elección del núcleo del algoritmo tampoco tiene efecto en la precisión.

Sin embargo, en la gráfica de tiempos vemos que si el núcleo es `rbf` los tiempos son considerablemente mayores, y esta diferencia aumenta exponencialmente con el tamaño del *dataset*. Para valores grandes parece que los mejores núcleos a usar son `linear` y `sigmoid`.

Coeficiente del núcleo (`gamma`)

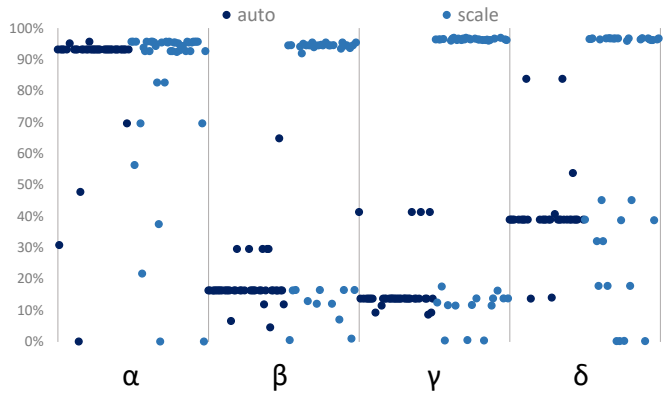

Figura 7.25: Precisión de Support Vector en función de `gamma`

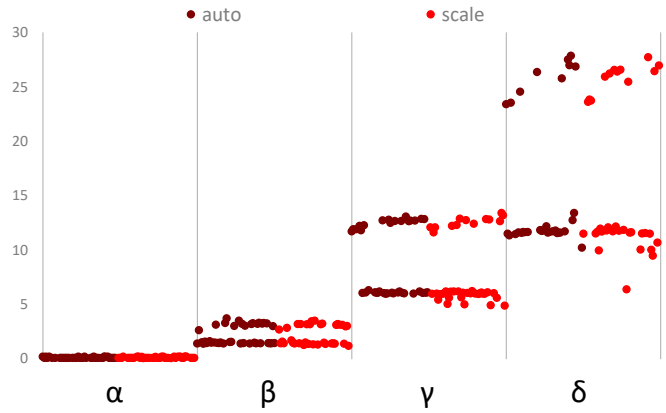

Figura 7.26: Tiempo de Support Vector en función de `gamma`

El coeficiente del núcleo solo se aplica al modelo cuando el núcleo es `poly`, `rbf` o `sigmoid`. Por ello se han omitido de las gráficas los experimentos en los que el núcleo es lineal. Como puede verse, la diferencia en la precisión se da debido a este parámetro. El valor para unos niveles de precisión aceptables es `scale`, aunque esto solo es evidente cuando el tamaño del *dataset* es grande.

En los tiempos no se observa diferencia notable entre los diferentes valores del parámetro, por lo que es fácil asumir que la eficiencia del algoritmo no depende de `gamma`.

Grado del polinomio (`degree`)

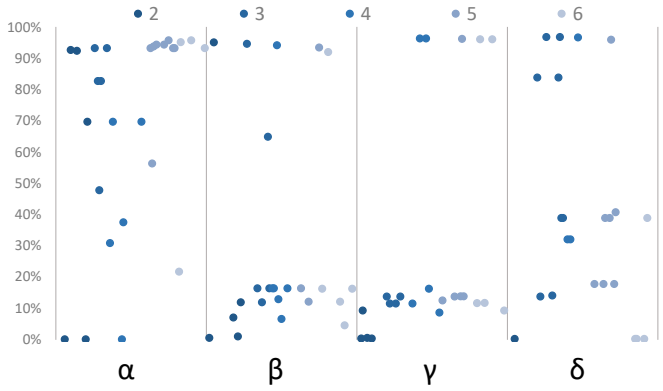

Figura 7.27: Precisión de Support Vector en función de `degree`

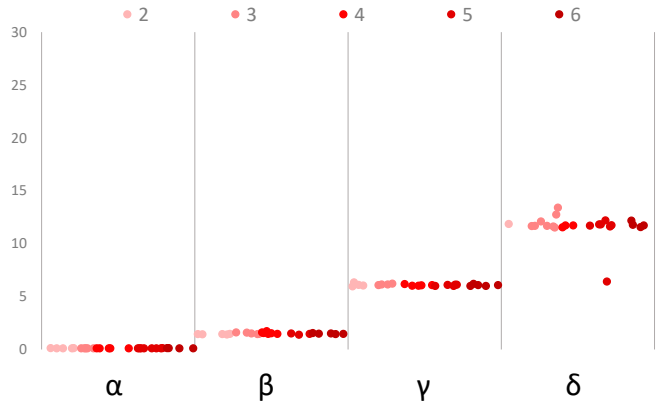

Figura 7.28: Tiempo de Support Vector en función de `degree`

En estas gráficas solo se muestran los experimentos en los que el núcleo del algoritmo es polinómico (`kernel=poly`), ya que `degree` es un sub-parámetro de `kernel`. Al tener pocos puntos es dificil encontrar patrones, pero los resultados obtenidos no muestran una relación entre el grado y la precisión o tiempo del algoritmo.

Término independiente de la función del núcleo (`coef0`)

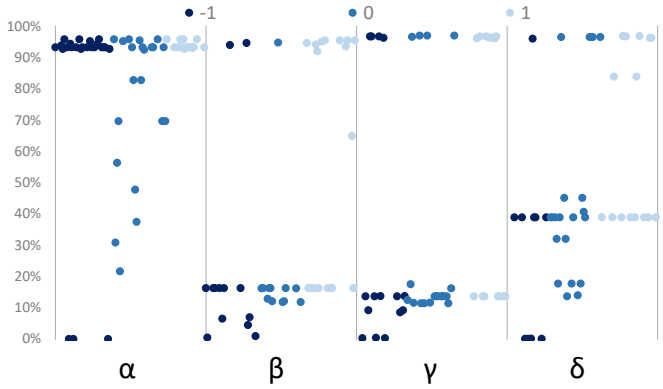

Figura 7.29: Precisión de Support Vector en función de `coef0`

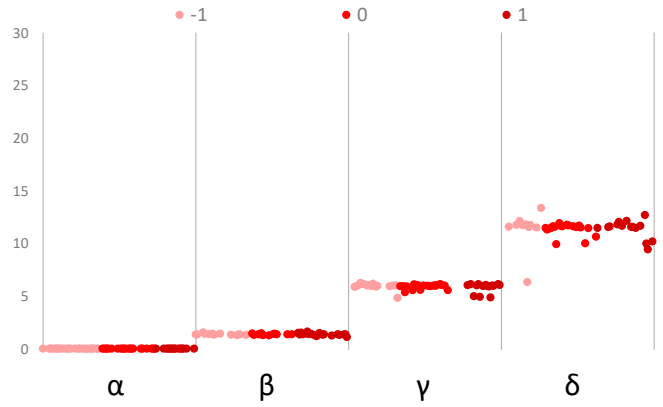

Figura 7.30: Tiempo de Support Vector en función de `coef0`

Este parámetro solo aplica cuando el núcleo del algoritmo es `poly` o `sigmoid`. De nuevo se han eliminado de la gráfica los puntos en los que esto no sucede. En ambas gráficas aparece un resultado parecido al de las gráficas de `degree` y `shrinking`: no existe relación entre el parámetros y los resultados de precisión y tiempo.

Parámetro de normalización (`C`)

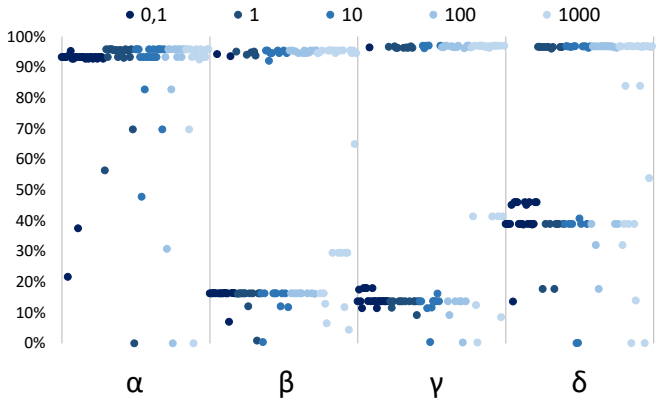

Figura 7.31: Precisión de Support Vector en función de `C`

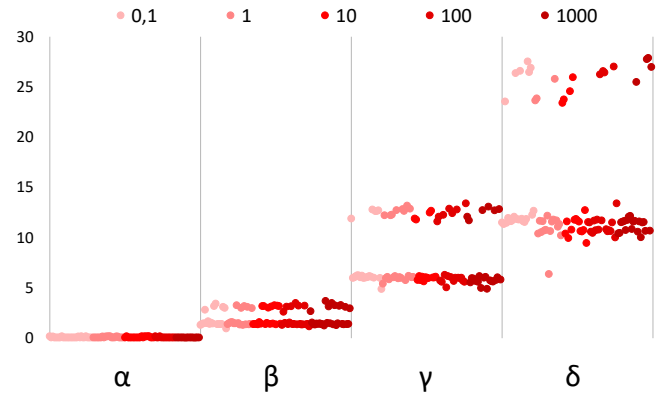

Figura 7.32: Tiempo de Support Vector en función de `C`

En estas gráficas tampoco hay variación en la precisión o el tiempo para los diferentes valores de `C`. Las variaciones entre los niveles de precisión y tiempo se deben, como ya hemos visto, a `gamma` y `kernel`.

7.2. Comparativa de los modelos

Las siguientes gráficas comparan la eficiencia de los 3 modelos en los 4 *datasets* empleados (recordemos que los 4 datasets son los mismos para los 3 modelos).

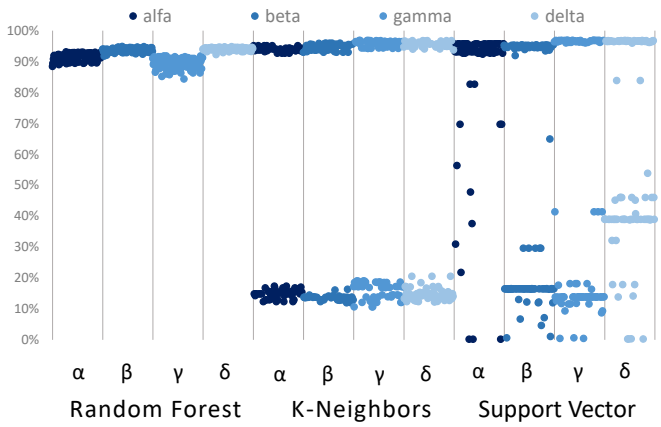

Figura 7.33: Comparativa de la precisión entre modelos

De entre los 3 modelos, el más consistente, o el que menos varía en función de sus parámetros, es Random Forest. Sin embargo, la eficiencia es sutilmente mayor en los otros dos si se parametrizan bien las ejecuciones. Es cierto que Random Forest no oscila tanto como los otros, pero como se ha visto, la oscilación de los otros modelos es debida a utilizar valores para sus parámetros no compatibles con el contenido de los *datasets*. Es posible que en otros dominios diferentes al de los idiomas esos modelos respondan mejor si se configuran con los parámetros adecuados.

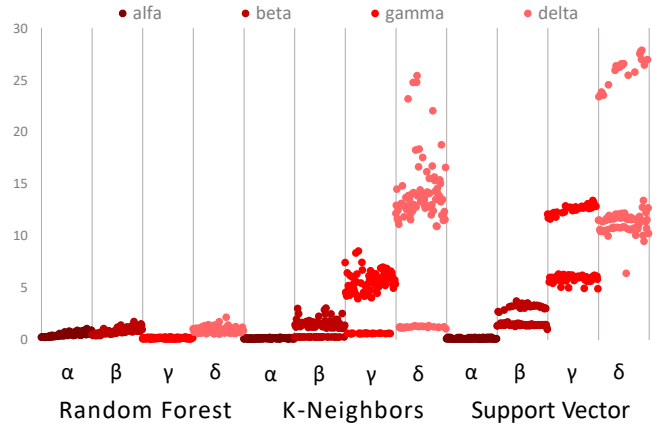

Figura 7.34: Comparativa del tiempo entre modelos

En la gráfica de tiempo si que se nota una diferencia bastante sustancial entre modelos. Todos ellos tienen tiempos similares en los *datasets* de menor tamaño, pero tanto K-Nearest Neighbors como Support Vector se disparan una vez aumentamos el tamaño de los datos. Hemos visto que KNN puede tener tiempos similares a RF cuando `algorithm=brute`, pero no siempre es así. Si el algoritmo de fuerza bruta se encuentra con casos poco favorables su tiempo también se dispara. Nótese que incluso en el caso mejor, los tiempos de KNN son un poco superiores a los de RF. Lo mismo ocurre con SV. Incluso seleccionando sus parámetros de forma que se minimice el tiempo, no es más eficiente que los peores casos

de RF en los *datasets* beta, gamma y delta.

Capítulo 8

Conclusiones y Trabajo Futuro

"Por eso a la ciudad se le llamó Babel, porque fue allí donde el Señor confundió el idioma de toda la gente de la tierra, y de donde los dispersó por todo el mundo."
— Génesis 11:9

Habiendo realizado un análisis de los resultados obtenidos con los experimentos, queda claro que el modelo más eficiente (tanto en precisión como en tiempo) es el Random Forest. Todos sus resultados de precisión se hallan por encima del 85 %, y sus máximos están cercanos al 96 %. En cuanto al tiempo medio del modelo, los resultados son muy satisfactorios, no superando los 2 segundos en ninguna de sus ejecuciones. De los parámetros que afectan la eficacia de este modelo, los únicos mínimamente significativos son el número de estimadores, el tamaño mínimo de hoja y la profundidad máxima del árbol. Se ha observado que cuanto menor es el número de estimadores, menor es el tiempo que tarda el modelo en realizar las predicciones. Asimismo, si el tamaño mínimo de hoja está fijado en 2, se obtienen mejores resultados de precisión. No sucede así con el tiempo, donde lo más beneficioso es utilizar tamaños mayores. La profundidad máxima de árbol no varía la precisión, pero según disminuye el valor de este parámetro, el tiempo que tarda el algoritmo disminuye de forma linea, por lo que lo más beneficioso el utilizar valores pequeños.

Esto no quita que KNN pueda optimizarse un poco más para acercarse al nivel de RF. Queda claro, viendo las gráficas, que la p de Minkowski tiene que ser 2 para obtener unos resultados de precisión aceptables. Entre todos los algoritmos de cómputo disponibles, el que da mejores resultados en tiempo es el de fuerza bruta (unos 2 segundos de media), pero esto está supeditado a que el algoritmo se encuentre ejecutando un caso mejor (cuando no es el caso, el tiempo medio de ejecución está en torno a los 15 segundos). El último parámetro que afecta al modelo de forma considerable es el tamaño de hoja. Si este valor crece, el tiempo de ejecución decrece, pero como puede verse en los resultados, utilizar los algoritmos BallTree y KDTree da peores resultados que el algoritmo de fuerza bruta en sus peores casos. De esta forma, los resultados para este parámetro no justifican emplear estos algoritmos lo suficiente.

Con diferencia, SV es el algoritmo menos recomendado para la detección de idiomas,

al menos en el alcance de este proyecto y para los tamaños de los *datasets* creados. De los posibles núcleos del algoritmo que pueden usarse, `rbf` aumenta considerablemente el tiempo de ejecución y no varía la precisión. En el caso de usarse `poly` o `sigmoid`, cuando el coeficiente del núcleo está fijado en `scale` tenemos una precisión similar a la de RF (en torno al 95 %), pero al compararlo con los dos modelos anteriores, vemos que su tiempo de ejecución no cae por debajo de los 5 segundos para el *dataset* gamma ni de los 12 para el delta (frente a los 2 segundos de RF y KNN en su configuración más óptima).

Como conclusión, se recomienda utilizar RF con un número de estimadores bajo (en torno a 200) y, si se quiere potenciar la precisión frente a la eficiencia, un tamaño de hoja de 2. En caso contrario, se recomienda minimizar el tamaño de hoja a costa de perder un poco de precisión. Para cualquiera de los dos escenarios un valor pequeño en la profundidad máxima del árbol acelera todo el proceso de predicción.

Una posible continuación del trabajo presentado aquí sería realizar experimentos con *datasets* de mayor tamaño en entornos con una alta capacidad de computación, además de incluir más parámetros de los que se escogieron en este proyecto para comprobar su relación con los resultados obtenidos en el dominio de la detección de idiomas. Existen más modelos que los evaluados aquí, por lo que se podría escalar todavía más el alcance del presente proyecto hasta poder encontrar patrones más claros en la variación de los parámetros.

Chapter 9

Introduction

"Now the whole earth had one language and the same words."
— Genesis 11:1

Introduction to the subject area. This chapter contains the translation of Chapter 1.

This document describes the process carried out for the elaboration of the project, explaining phase by phase the steps taken and providing the results of the experiments at the end.

9.1. Motivation

In an increasingly globalized and interconnected world, there are many differences between one place and another: physical distance, different time zones, cultural differences, etc. Among these differences, perhaps the one that most affects communication and agreements between people and organizations is linguistics.

There are more than 7,000 languages worldwide, including recognized variants and dialects. While in Europe "only" some 286 languages are spoken, the case of Asia is more striking, with a total of more than 2,300 languages and variants.

Due to this marked linguistic differentiation, simultaneous translation devices and programs have become more and more prevalent over time and are used by more and more users for personal use (tourism, business trips, etc.).

There are many different programs or devices that allow simultaneous translation, but they all have something in common: the first step is to identify the language that is being listened to in order to translate it into the language chosen by the user.

Therefore, in such a fast and direct communication as verbal communication, the translation speed of the program must be competent and allow the exchange of sentences at such a speed that it is comfortable for the interlocutors. In fact, once the language of the source message to be translated is known, the algorithms have proven to be very fast in their translation work.

However, detecting in which language a speech or text is being spoken or written is usually what takes the most effort for the program. There are many languages that share

a lexicon, but not grammatical rules or verb tenses, so the importance of knowing which language detection algorithms work best in which environments is considerable.

9.2. State of the art

There are multiple works similar to the one presented by this paper. In a study (Pavliy y Lewis, 2016) conducted by B. Pavliy and J. Lewis for the *Bulletin of Toyama University of International Studies* in 2016 compares the accuracy of Google and Twitter language detection algorithms for Ukrainian and Russian languages.

Graham, Hale and Gaffney investigated in their 2014 paper (Graham et al., 2014) on how Twitter is able to identify the nationality of its users through the language of their tweets and geolocation, but they do not compare the effectiveness of the different algorithms used by the social network.

Finally, it is worth noting the study (Grothe et al., 2008) by Grothe, De Luca and Nürnberger in 2008 on the efficiency and accuracy of computational language detection methods based on short word, frequent word and n-gram approaches, which brings this study quite close to the one described in this report.

9.3. Objectives

The ultimate goal of the present project is, as already discussed, to compare the effectiveness of different language detection models when dealing with texts written in different languages. This objective can be separated into two phases.

1. Program creation: A program written in Python will be developed to take texts in different languages and detect in which language they are written. This program will consist of three different parts:

   a) Text preprocessing: This part of the program is in charge of taking the original texts and eliminating any information not written in the language of the text or that does not provide knowledge. This includes special characters and monolithic words (formed by one character).

   b) Construction of the dataset: This part will build the set of words and texts on which the language detection models will be executed, taking as input the previously preprocessed texts.

   c) Interface implementation: In addition, an executable interface will be built from the programming environment capable of detecting the language of a written text (once the model loaded in the program has been trained).

2. Experiments: Different Machine Learning detection models will be run on the languages in the scope of the project and parameters to be defined such as speed and

hit rates will be compared. This objective can be broken down into the following steps:

a) Definition of the experiments: The metrics and parameters of the experiments to be carried out will be defined, as well as the execution of the experiments.

b) Execution of the experiments: The guide defined in the previous step will be followed and all the results obtained will be written down.

c) Analysis of results: The results will be processed and presented in a clear and understandable way. A conclusion will be drawn from the results.

d) Model Selection: Once the results have been analyzed, the most efficient model will be selected and the executable program with that model will be loaded for its later use.

9.4. Work plan

At the beginning of the project, a time estimate for the objectives and the steps leading up to their completion was established to fit the total time available for the project.

1. **Research and literature review** (estimate: 2 weeks): In this initial stage, an exhaustive research on the methods and techniques used in language detection will be carried out. Scientific articles, books and other relevant sources will be searched to understand what are the best practices in this domain of Machine Learning and how it works.

2. **Data collection and preparation** (estimate: 2 weeks): A data set covering the variety of languages that will allow future training and evaluation testing will be collected.

3. **Implementation of the program** (estimate: 8 weeks): During this stage all the algorithms necessary to carry out the experiments will be developed, as well as algorithms for the creation of the preprocessed texts, the Tf-Idf tables and the datasets, which will allow the models to be trained.

4. **Model training and evaluation** (estimate: 6 weeks): The data collected during the second phase will be used to train and evaluate the language detection models (Random Forest, K-Neighbors and Support Vector). The data will be divided into training and test sets, cross-validation techniques will be used, and model parameters will be adjusted to maximize accuracy and efficiency.

5. **Development of a user interface** (estimate: 4 weeks): A graphical user interface will be implemented that is intuitive and allows users to enter plain text or a file with extension `txt` to predict the language in which it is written. For this, a series of libraries and tools will be used to make the interface as interactive as possible, taking into account usability and user memorability.

6. **Conducting the experiments and analyzing the results** (estimate: 4 weeks): A series of comprehensive experiments will be performed using 4 datasets, whose parameters are chosen incrementally to evaluate the robustness and performance of the models used. The results obtained will be analyzed and conclusions on the effectiveness of the implemented algorithms will be drawn.

During each and every phase, the different parts of the report will be written.

9.5. Document structure

The first chapter (1. Introduction) consists of an executive summary with the objectives of the project, the scope of the project and the phases into which the project is divided.

The second chapter (2. Data preprocessing) will deal with the process of creating the preprocessing program that will adapt the original texts to the format accepted by the language detection algorithms.

The third chapter (3. Dataset creation) describes the process of creating the program that will build the computational models on the texts processed by the preprocessing program in order to carry out the experiments of chapter 5.

The fourth chapter (4. User Interface) deals with the creation of an interface that allows the program to be used in windowed mode.

The fifth chapter (5. Models) discusses the different language detection models, giving a simple definition of each one and comparing their operation.

The sixth chapter (6. Evaluation Methods) defines the metrics and parameters to be measured and the experiments to be performed to compare the effectiveness of each algorithm for the languages established in the scope.

The seventh chapter (7. Results) will show the results obtained with the experiments, providing visual and detailed information to facilitate the understanding of the results.

The eighth chapter (8. Conclusions and Future Work) contains the conclusions reached from the results of the experiments, and provides suggestions on how to continue the work started here.

The bibliography and appendices are also included at the end of the document.

Chapter 10

Conclusions and Future Work

"Therefore its name was called Babel, because there the Lord confused the language of all the earth. And from there the Lord dispersed them over the face of all the earth."

— Genesis 11:9

Having performed an analysis of the results obtained with the experiments, it is clear that the most efficient model (both in accuracy and time) is the Random Forest. All its accuracy results are above 85%, and its maxima are close to 96%. As for the average time of the model, the results are very satisfactory, not exceeding 2 seconds in any of its executions. Among the parameters affecting the efficiency of this model, the only minimally significant ones are the number of estimators, the minimum leaf size and the maximum tree depth. It has been observed that the smaller the number of estimators, the shorter the time it takes for the model to make predictions. Likewise, if the minimum leaf size is set to 2, better accuracy results are obtained. This is not the case for time, where it is more beneficial to use larger sizes. The maximum tree depth does not vary the accuracy, but as the value of this parameter decreases, the time taken by the algorithm decreases linearly, so it is more beneficial to use smaller values.

This does not detract from the fact that KNN can be further optimized to approach the RF level. It is clear from the plots that Minkowski's p has to be 2 for acceptable accuracy results. Among all the available computational algorithms, the one that gives the best results in time is the brute force algorithm (about 2 seconds on average), but this is subject to the algorithm being running a better case (when this is not the case, the average execution time is around 15 seconds). The last parameter that affects the model considerably is the leaf size. If this value increases, the execution time decreases, but as can be seen from the results, using the BallTree and KDTree algorithms gives worse results than the brute force algorithm in its worst cases. Thus, the results for this parameter do not sufficiently justify employing these algorithms.

SV is by far the least recommended algorithm for language detection, at least in the scope of this project and for the sizes of the *datasets* created. Of the possible kernels of the algorithm that can be used, `rbf` increases the execution time considerably and does not vary the accuracy. In the case of using `poly` or `sigmoid`, when the kernel coefficient is

set to `scale` we have a similar accuracy to RF (around 95%), but when compared to the two previous models, we see that their execution time does not fall below 5 seconds for *dataset* gamma or 12 seconds for delta (compared to 2 seconds for RF and KNN in their most optimal configuration).

In conclusion, it is recommended to use RF with a low number of estimators (around 200) and, if accuracy versus efficiency is to be enhanced, a leaf size of 2. Otherwise, it is recommended to minimise the leaf size at the cost of some loss of accuracy. For either scenario a small value for the maximum tree depth speeds up the whole prediction process.

A possible continuation of the work presented here would be to perform experiments with larger *datasets* in environments with high computing power, as well as including more parameters than those chosen in this project to test their relationship with the results obtained in the domain of language detection. There are more models than those evaluated here, so the scope of this project could be scaled even further to find clearer patterns in the variation of the parameters.

Contribuciones Personales

Daniel Lucas Caturla

Mi función a nivel organizativo fue la de ir juntando la parte de código de Rogger con la mía y preparar versiones del código que se fueron subiendo al repositorio.

Durante la fase inicial del proyecto, me hice cargo de probar las herramientas que escogimos de entre las presentadas por Jaime. Las pruebas consistieron en estudiar el proceso de instalación, verificar que la herramienta funciona e indagar en su manejo básico.

Durante la creación del programa, especifiqué en primer lugar junto a mis compañeros los requisitos del proyecto y ayudé a diseñar la arquitectura. Con la arquitectura establecida, hice el diseño de los módulos TF-IDF, Dataset y Modelos.

En primer lugar, investigué el funcionamiento de las tablas TF-IDF en el paquete *scikit-learn* y busqué información sobre la implementación de los modelos escogidos para la realización del proyecto en ese mismo paquete. A partir de esta información y juntándola con los requisitos del proyecto, elaboré la arquitectura del *dataset*. A continuación, hice la implementación en código con la ayuda de Rogger, teniendo en cuenta también su trabajo previo con el preprocesado del texto. Tras completar esta parte elaboré, con la ayuda de Rogger, un código para el tratamiento de los modelos adaptado al de preprocesado, tablas TF-IDF y *dataset*. Durante todo este proceso fui redactando el Capítulo 3 junto a Rogger.

Con el programa ya completado, me puse a diseñar e implementar una interfaz para facilitar y agilizar su uso. Aprovechando esto, también metí una funcionalidad al programa que sirviera para probar la investigación a nivel usuario. La funcionalidad es la de detectar el idioma de cualquier texto introducido por un usuario y utiliza todos los módulos implementados anteriormente. Tras acabar con el diseño y la implementación, redacté el Capítulo 4.

Durante la fase de experimentos, ejecuté el modelo K-Neighbors con el código implementado por Rogger y le pasé los resultados de las pruebas a Jaime para que creara las gráficas. Las gráficas las sacó a partir de unas plantillas que me encargué de crear en Excel. Las plantillas incluían unas fórmulas para recoger los datos de las tablas según el valor de un determinado parámetro, teniendo un valor por columna. Además, diseñé unas tablas formateadas que seleccionaban los datos recogidos por las fórmulas.

Por último, contribuí a la redacción del Capitulo 6 y ayudé a sacar las conclusiones finales del proyecto.

Rogger Huayllasco De la Cruz

A nivel organizativo me centré en poder adecuar los horarios de cada uno de los miembros del equipo para poder tener reuniones periódicas de manera que el avance del proyecto sea continuo.

Inicialmente empecé con la comprensión, junto con el equipo, de todos los requisitos necesarios del proyecto. Participé en el análisis de las necesidades y objetivos del proyecto, ayudando también a establecer unos criterios para su finalización con éxito.

Basándome en los requisitos que se establecieron al inicio, contribuí al diseño de la arquitectura software identificando los componentes clave y las relaciones entre ellos.

A nivel de implementación de código y ya con los requisitos y la arquitectura *software* definidos, comencé a implementar el código para la limpieza de textos, para la cual usé una serie de archivos de texto en varios idiomas. Tras varias pruebas de ejecución conseguí transformarlos al formato deseado. También ayudé a implementar la lógica de la creación de los *datasets* y tablas TF-IDF, con esto ya teníamos los conjuntos de datos necesarios para los siguientes pasos. Tras esto ayudé también a implementar la lógica para el uso de los modelos establecidos en el proyecto (Random Forest, K-Nearest Neighbors y Support Vector).

Con el conjunto de datos definidos y la implementación de los modelos me pude centrar en los experimentos. Para esto realicé una búsqueda exhaustiva en los parámetros de cada uno de los modelos, buscando en paginas oficiales y documentación de otros experimentos realizados anteriormente relacionados con el tema del proyecto. Paralelamente a esto, establecí cuáles serían las métricas principales a medir en los experimentos. Con todo lo anterior ya definido, podía comenzar con la implementación a nivel de código, el cual consistía en establecer distintos valores para los parámetros de cada modelo, intentando que abarquen el mayor rango posible, y aplicarlos aleatoriamente al modelo. Tras varias pruebas con distintos datos pude ejecutar el modelo Random-Forest y obtener una tabla con los resultados de las ejecuciones, sus parámetros elegidos así como el porcentaje de acierto y el tiempo medio de ejecución por cada entrenamiento.

Durante la implementación de código me centre también en la optimización y rendimiento de los algoritmos, tanto a nivel de tiempo de ejecución como de espacio en memoria, para lo cual tuve que buscar entre las distintas librerías que aporta Python para usar el algoritmo más óptimo.

Por último, mi aporte en la memoria fue ayudar a la redacción del Capitulo 2 durante la implementación de este. También añadí las definiciones formales de los parámetros de cada uno de los modelos. Y aporte en la redacción del Capitulo 3 junto con Daniel. En el Capitulo 7 pude añadir algunas de las gráficas que reflejaban el resultado de los experimentos y pude añadir comentarios explicativos de cada una de estas. Por ultimo realicé junto con mis compañeros las conclusiones finales tras analizar en conjunto todos los resultados.

Jaime Sánchez Rodríguez

A nivel organizativo, me encargué de configurar el entorno para el almacenamiento de los datos del proyecto y las herramientas de comunicación y gestión de tareas. Mantuve actualizados el repositorio y el gestor de tareas según avanzaba el proyecto, reflejando en todo momento el estado del proyecto tanto en sus archivos y entregables como las tareas que se fueron completando. La planificación se llevó a cabo en forma de *sprints*, en los que mantuvimos reuniones quincenales para actualizar el cuadro de tareas y resolver dudas.

Para la fase previa del proyecto, realicé la búsqueda de recursos teóricos para el planteamiento del proyecto, tales como documentación oficial de los modelos utilizados, trabajos de investigación previos (estado del arte) e investigué las múltiples herramientas de desarrollo del software a nuestra disposición. Compartí con mis compañeros los puntos fuertes y débiles de cada una y escogimos cada programa basándonos en las necesidades del proyecto.

Durante la fase de creación del programa definí junto a mis compañeros los requisitos funcionales de la solución y diseñé la arquitectura de la solución a partir de ellos. También elaboré los diagramas que pueden verse en los capítulos dedicados a la creación del programa y colaboré con mis compañeros para que la implementación reflejara los requisitos del diseño. Durante las reuniones quincenales apoyé a mis compañeros aportando documentación técnica de Python y puntualmente ayudé a programar alguna función que requería adaptar el diseño establecido.

Durante la fase de experimentación ejecuté las pruebas para el modelo Support Vector y recopilé los resultados de mis compañeros para la elaboración de las gráficas. Como las pruebas fueron muchas y el tiempo que tardaron en terminar largo, se decidió que cada uno de nosotros realizara un modelo diferente desde nuestros equipos. Esto también sirvió para probar el programa en diferentes sistemas operativos (MacOS, Windows y Linux) y verificar que funcionara correctamente. De las gráficas con los resultados de los experimentos analicé cada una y redacté las conclusiones del equipo.

Por último, fui el principal redactor de la parte teórica y de análisis de resultados de la memoria (capítulos 1, 5, 6, 7 y 8), incluí todos los recursos utilizados (imágenes, gráficas, referencias bibliográficas y tablas de resultados) y revisé el documento una vez estuvo completo. En la parte acerca de la creación del programa (capítulos 2, 3 y 4) aporté comentarios y adapté el trabajo realizado por mis compañeros para que fuera consistente con los demás capítulos. Colaboré con mis compañeros para que el papel de cada función y de cada librería queden correctamente reflejados en la memoria. De esta forma, considero que el programa puede entenderse y modificarse con la información contenida en este documento y los comentarios incluidos en el propio código fuente.

Bibliografía

Graham, M., Hale, S. A. y Gaffney, D. Where in the world are you? geolocation and language identification in twitter. *The Professional Geographer*, vol. 66(4), páginas 568–578, 2014.

Grothe, L., De Luca, E. W. y Nürnberger, A. A comparative study on language identification methods. En *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*. European Language Resources Association (ELRA), Marrakech, Morocco, 2008.

Pavliy, B. y Lewis, J. *The performance of Twitter's language detection algorithm and Google's Compact Language Detector on language detection in Ukrainian and Russian tweets*. Bulletin of Toyama University of International Studies, 2016.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. y Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, vol. 12, páginas 2825–2830, 2011.

Apéndice A

Tablas de Resultados

En este apéndice figuran las tablas completas con los resultados de ejecución de los modelos sobre los *datasets* creados.

A.1. Resultados de ejecución de Random Forest

Tabla A.1: Tabla de resultados de Random Forest en Alfa

Exp.	**est.**	**split**	**leaf**	**depth**	**boot.**	**mean_score**	**mean_time**
1	1000	10	1	60	TRUE	0,9	0,0990219593
2	2000	5	1	20	FALSE	0,9305555556	0,1816407681
3	1800	5	1	110	TRUE	0,8833333333	0,1726393223
4	1800	5	2	50	FALSE	0,9333333333	0,1618367672
5	1000	5	2	110	TRUE	0,925	0,09802241325
6	600	5	2	50	TRUE	0,9194444444	0,06181416512
7	600	2	1	100	TRUE	0,8638888889	0,0708190918
8	1400	5	2	90	TRUE	0,925	0,1284288406
9	1200	10	2	90	FALSE	0,9277777778	0,125906086
10	800	2	4	20	FALSE	0,9055555556	0,07921724319
11	1600	2	2	None	FALSE	0,925	0,1440320969
12	800	5	4	90	TRUE	0,8916666667	0,07801685333
13	1600	2	1	70	TRUE	0,8916666667	0,1562357426
14	600	10	2	80	TRUE	0,9222222222	0,06561436653
15	400	2	1	80	TRUE	0,8611111111	0,05361242294
16	200	5	1	40	FALSE	0,9194444444	0,03180670738
17	1000	2	1	10	FALSE	0,9277777778	0,0962223053
18	1800	2	1	None	FALSE	0,8805555556	0,1718388557
19	1800	2	4	90	TRUE	0,9027777778	0,1519438267

Continúa en la página siguiente

Tabla A.1 – Continuación de la tabla de resultados en Alfa

Exp.	est.	split	leaf	depth	boot.	mean_score	mean_time
20	400	5	4	20	FALSE	0,8944444444	0,04818220139
21	1800	2	2	110	FALSE	0,9305555556	0,162236166
22	1000	10	2	70	FALSE	0,9277777778	0,111625433
23	2000	10	4	20	FALSE	0,9055555556	0,1692377567
24	400	10	2	100	FALSE	0,9277777778	0,05081157684
25	1600	10	4	40	TRUE	0,8944444444	0,1352313519
26	1400	2	2	10	FALSE	0,9305555556	0,1298291206
27	200	5	1	80	FALSE	0,9027777778	0,04060883522
28	1600	10	1	50	TRUE	0,9055555556	0,1480557442
29	800	10	2	100	TRUE	0,9305555556	0,0798201561
30	2000	2	4	40	FALSE	0,9027777778	0,1708382607
31	800	2	1	10	FALSE	0,9277777778	0,09042067528
32	2000	10	4	None	FALSE	0,9027777778	0,1682386875
33	800	10	2	10	TRUE	0,9277777778	0,07741727829
34	1000	5	1	20	TRUE	0,9222222222	0,09842152596
35	1800	5	1	90	FALSE	0,8972222222	0,1728386879
36	2000	5	1	110	TRUE	0,8861111111	0,1932067394
37	200	5	2	100	FALSE	0,9194444444	0,03600811958
38	1400	10	2	20	FALSE	0,9305555556	0,1309300423
39	1200	2	1	110	TRUE	0,8666666667	0,1300284863
40	1600	2	2	10	TRUE	0,9305555556	0,1540529251
41	600	10	4	60	FALSE	0,9055555556	0,06721568108
42	1600	10	2	100	TRUE	0,9333333333	0,1536349773
43	200	5	4	40	TRUE	0,8777777778	0,03140721321
44	1600	10	4	20	FALSE	0,9027777778	0,1770081997
45	800	2	4	10	FALSE	0,9083333333	0,08661899567
46	1600	10	2	90	FALSE	0,9305555556	0,1422316074
47	1000	5	2	20	FALSE	0,925	0,1024224758
48	200	5	2	30	FALSE	0,9277777778	0,0340069294
49	800	5	4	60	TRUE	0,8861111111	0,07681717873
50	1400	10	4	60	TRUE	0,9055555556	0,1256286621
51	800	2	2	20	FALSE	0,9361111111	0,08321805
52	1800	5	4	110	TRUE	0,8944444444	0,1682383537
53	200	10	2	60	FALSE	0,9333333333	0,03680839539
54	400	10	2	80	TRUE	0,925	0,05561232567
55	1400	10	1	70	TRUE	0,9	0,1379398346
56	600	5	2	70	FALSE	0,9222222222	0,06621413231

Continúa en la página siguiente

Tabla A.1 – Continuación de la tabla de resultados en Alfa

Exp.	est.	split	leaf	depth	boot.	mean_score	mean_time
57	1000	10	1	110	TRUE	0,8888888889	0,102222538
58	1200	10	2	30	FALSE	0,9277777778	0,1176261425
59	200	2	4	90	FALSE	0,8944444444	0,03240714073
60	1400	10	2	None	TRUE	0,9222222222	0,1300291061
61	1400	5	1	10	FALSE	0,925	0,1304297447
62	800	2	1	None	TRUE	0,8611111111	0,0884203434
63	600	2	1	80	FALSE	0,8972222222	0,0714158535
64	1200	5	2	70	FALSE	0,925	0,1188268661
65	1800	10	1	20	TRUE	0,925	0,165237093
66	1200	5	2	20	TRUE	0,925	0,1112254143
67	800	2	2	None	FALSE	0,9277777778	0,08141770363
68	2000	5	1	10	FALSE	0,925	0,1752397537
69	200	2	1	110	TRUE	0,8611111111	0,03600811958
70	400	5	1	None	TRUE	0,8777777778	0,0514125824
71	400	5	4	110	FALSE	0,9	0,04981126785
72	1200	2	1	40	TRUE	0,9055555556	0,121628046
73	400	10	4	20	TRUE	0,8944444444	0,04801120758
74	800	2	4	100	TRUE	0,8638888889	0,08161816597
75	200	10	4	90	TRUE	0,8666666667	0,03280701637
76	600	2	2	30	TRUE	0,9194444444	0,06462068558
77	1400	5	2	10	TRUE	0,9305555556	0,1260280609
78	1800	10	2	80	TRUE	0,9277777778	0,1556349754
79	800	2	4	30	TRUE	0,8777777778	0,08741955757
80	600	10	4	80	FALSE	0,9027777778	0,07141680717
81	2000	10	1	60	TRUE	0,9	0,1900436401
82	2000	10	2	70	TRUE	0,9222222222	0,2082469463
83	400	5	4	100	TRUE	0,8888888889	0,05541210175
84	600	10	1	40	TRUE	0,9111111111	0,06521468163
85	1600	2	2	10	FALSE	0,9333333333	0,1360608578
86	1000	2	2	70	FALSE	0,9277777778	0,09906167984
87	400	2	2	90	FALSE	0,9305555556	0,05103569031
88	600	10	2	20	FALSE	0,9277777778	0,07521700859
89	200	5	4	80	TRUE	0,8666666667	0,03280739784
90	1600	2	4	40	TRUE	0,9083333333	0,1458347321
91	200	10	4	10	FALSE	0,8861111111	0,03940877914
92	1000	5	4	50	FALSE	0,9055555556	0,1046234131
93	400	10	2	20	FALSE	0,925	0,0534113884

Continúa en la página siguiente

Tabla A.1 – Continuación de la tabla de resultados en Alfa

Exp.	est.	split	leaf	depth	boot.	mean_score	mean_time
94	800	2	2	None	TRUE	0,9361111111	0,08621935844
95	800	5	1	40	TRUE	0,9083333333	0,09562215805
96	1400	2	2	20	FALSE	0,9333333333	0,1340307713
97	400	10	2	30	FALSE	0,9222222222	0,05161151886
98	1400	5	4	70	FALSE	0,9	0,1310290813
99	1800	2	4	50	FALSE	0,9	0,1600363731
100	400	10	1	100	TRUE	0,8944444444	0,05241193771

Tabla A.2: Tabla de resultados de Random Forest en Beta

Exp.	**est.**	**split**	**leaf**	**depth**	**boot.**	**mean_score**	**mean_time**
1	1800	2	2	40	TRUE	0,9287356322	0,3408881187
2	1600	2	1	70	FALSE	0,9344827586	0,3092262268
3	1400	2	2	50	TRUE	0,9264367816	0,2702210903
4	1000	5	1	None	FALSE	0,9275862069	0,3258732319
5	1400	10	4	80	TRUE	0,9114942529	0,2180490971
6	600	2	4	10	TRUE	0,9045977011	0,1520835876
7	1200	10	4	110	FALSE	0,9195402299	0,212246418
8	1800	2	4	110	TRUE	0,908045977	0,2587131023
9	200	2	2	70	TRUE	0,9298850575	0,1150270939
10	1200	10	4	50	FALSE	0,916091954	0,2360507488
11	200	10	1	40	TRUE	0,924137931	0,1139160633
12	1200	10	1	70	TRUE	0,9333333333	0,2662042141
13	200	10	2	10	FALSE	0,9137931034	0,1154621124
14	400	10	2	80	TRUE	0,9310344828	0,158634901
15	800	2	2	20	FALSE	0,9206896552	0,2074475765
16	400	2	4	70	TRUE	0,9034482759	0,1260620594
17	2000	5	1	100	FALSE	0,9310344828	0,4861783504
18	1600	2	1	100	TRUE	0,9287356322	0,3396763802
19	200	10	4	60	TRUE	0,891954023	0,103223896
20	1600	5	2	50	FALSE	0,9310344828	0,2990662098
21	1200	10	2	90	FALSE	0,9298850575	0,2938122749
22	200	10	4	20	FALSE	0,9114942529	0,1105256557
23	2000	2	4	90	FALSE	0,9114942529	0,31827178
24	600	5	2	40	TRUE	0,9287356322	0,1844418049
25	1600	10	1	90	TRUE	0,9298850575	0,36058321
26	1200	5	2	10	TRUE	0,9195402299	0,2406146526
27	600	10	2	70	FALSE	0,9367816092	0,1678367615
28	600	10	2	None	FALSE	0,9252873563	0,174438715
29	200	5	4	100	FALSE	0,916091954	0,1038215637
30	600	10	1	60	TRUE	0,9275862069	0,180265379
31	1600	10	4	90	FALSE	0,9172413793	0,2871086121
32	2000	10	2	100	FALSE	0,9333333333	0,3780844212
33	1200	5	4	30	FALSE	0,916091954	0,2430561543
34	1600	5	2	80	FALSE	0,9367816092	0,28726511
35	200	10	2	20	TRUE	0,9264367816	0,1082233429
36	1400	5	2	70	TRUE	0,9264367816	0,303468132

Continúa en la página siguiente

Tabla A.2 – Continuación de la tabla de resultados en Beta

Exp.	est.	split	leaf	depth	boot.	mean_score	mean_time
37	200	2	2	60	TRUE	0,9275862069	0,119026804
38	600	2	2	20	TRUE	0,9218390805	0,1728288651
39	1800	10	1	50	TRUE	0,9298850575	0,3700742245
40	200	10	4	90	FALSE	0,9068965517	0,1058236599
41	600	5	4	70	FALSE	0,9195402299	0,1628333569
42	1800	5	2	50	TRUE	0,9287356322	0,3320738316
43	1200	10	1	40	TRUE	0,9218390805	0,2592586994
44	1800	10	2	110	FALSE	0,9298850575	0,3740837097
45	600	2	4	30	TRUE	0,9022988506	0,164038372
46	2000	2	4	70	FALSE	0,916091954	0,3288748264
47	1600	10	2	100	FALSE	0,9356321839	0,2884648323
48	1200	2	4	10	TRUE	0,9114942529	0,2064482689
49	200	5	1	10	FALSE	0,9045977011	0,1062234402
50	1200	2	2	110	FALSE	0,9333333333	0,2644591331
51	400	10	2	30	TRUE	0,924137931	0,1368003368
52	2000	5	1	50	FALSE	0,9310344828	0,4044905186
53	600	2	2	110	FALSE	0,9310344828	0,1864189148
54	1600	2	2	80	FALSE	0,9310344828	0,333108139
55	200	2	4	None	FALSE	0,9195402299	0,1098258018
56	600	2	4	None	FALSE	0,9183908046	0,1508342266
57	800	5	1	None	TRUE	0,9275862069	0,2584578514
58	600	10	2	110	TRUE	0,932183908	0,1930434704
59	1400	2	4	100	FALSE	0,9172413793	0,2420534134
60	1800	5	1	80	FALSE	0,9356321839	0,3456766605
61	400	5	4	None	FALSE	0,9183908046	0,1274281502
62	1600	10	1	80	TRUE	0,9344827586	0,3306742668
63	400	10	1	10	TRUE	0,9114942529	0,1366299152
64	1000	10	2	90	FALSE	0,932183908	0,2356521606
65	200	5	1	100	TRUE	0,9275862069	0,1096241951
66	2000	10	4	60	TRUE	0,9068965517	0,2944638252
67	400	10	2	20	FALSE	0,9229885057	0,1310987473
68	800	10	4	40	FALSE	0,9229885057	0,1782394886
69	200	10	4	70	TRUE	0,9	0,1098251343
70	800	5	1	40	TRUE	0,9264367816	0,2036461353
71	1800	2	4	110	FALSE	0,9195402299	0,2664391041
72	400	5	2	90	FALSE	0,9298850575	0,1442324638
73	1600	10	4	100	FALSE	0,9172413793	0,2440554619

Continúa en la página siguiente

Tabla A.2 – Continuación de la tabla de resultados en Beta

Exp.	est.	split	leaf	depth	boot.	mean_score	mean_time
74	1000	5	4	50	TRUE	0,9068965517	0,1846413136
75	1400	2	4	None	FALSE	0,9149425287	0,2464598656
76	1800	2	2	40	FALSE	0,9275862069	0,3602814674
77	200	5	2	90	TRUE	0,9252873563	0,1190267563
78	1200	2	2	90	FALSE	0,9333333333	0,2422554016
79	1000	10	1	None	FALSE	0,9264367816	0,2850655556
80	2000	10	4	80	TRUE	0,9103448276	0,2808869362
81	800	10	1	80	TRUE	0,9287356322	0,2304520607
82	1200	2	1	30	TRUE	0,924137931	0,2456546307
83	1200	5	2	60	FALSE	0,9333333333	0,2758625031
84	1600	10	2	110	TRUE	0,9287356322	0,2902650833
85	1000	2	4	50	TRUE	0,9057471264	0,2104485035
86	200	5	4	80	TRUE	0,8988505747	0,1079962254
87	1800	5	2	80	FALSE	0,9356321839	0,3734835625
88	2000	2	1	70	TRUE	0,9298850575	0,4294513226
89	800	10	1	80	FALSE	0,932183908	0,2157428741
90	2000	10	4	30	TRUE	0,9103448276	0,284264183
91	1000	10	2	30	TRUE	0,924137931	0,2204507351
92	400	10	4	60	FALSE	0,9218390805	0,1312289238
93	600	10	4	10	FALSE	0,9137931034	0,1430316925
94	2000	2	4	110	FALSE	0,9183908046	0,2984551907
95	1600	2	2	60	FALSE	0,9298850575	0,2956672192
96	1800	5	2	20	TRUE	0,924137931	0,2801296234
97	1000	2	4	20	TRUE	0,908045977	0,2258499146
98	600	5	1	None	FALSE	0,9264367816	0,1974373817
99	1600	10	1	20	FALSE	0,9172413793	0,2904651642
100	2000	5	4	40	TRUE	0,9068965517	0,241369772

Tabla A.3: Tabla de resultados de Random Forest en Gamma

Exp.	est.	split	leaf	depth	boot.	mean_score	mean_time
1	600	10	1	110	TRUE	0,9442028986	0,6639233112
2	800	5	4	110	FALSE	0,9463768116	0,5480383873
3	1600	10	2	30	TRUE	0,95	0,8457057953
4	1800	10	1	10	FALSE	0,9413043478	0,7599171638
5	1800	2	4	50	FALSE	0,9434782609	1,051542759
6	800	5	4	70	FALSE	0,9456521739	0,5435287952
7	800	10	2	80	FALSE	0,9492753623	0,7053641319
8	800	10	1	20	TRUE	0,9456521739	0,5749437332
9	800	10	2	50	FALSE	0,9492753623	0,7130724907
10	600	5	2	70	FALSE	0,9485507246	0,6314387321
11	400	5	4	None	TRUE	0,9333333333	0,4674246311
12	200	2	2	20	FALSE	0,9376811594	0,3907916546
13	1000	5	4	70	FALSE	0,9427536232	0,7198210239
14	2000	5	2	10	TRUE	0,9434782609	0,7667020321
15	1200	2	1	30	TRUE	0,9413043478	0,7895364761
16	800	10	4	110	TRUE	0,9376811594	0,5029153347
17	600	5	4	20	FALSE	0,9463768116	0,5732303619
18	200	5	4	30	TRUE	0,9347826087	0,3884928226
19	1200	2	4	50	TRUE	0,9355072464	0,6247915268
20	1800	10	2	20	FALSE	0,947826087	0,8599788189
21	200	2	2	60	FALSE	0,9442028986	0,4333990097
22	400	2	4	20	FALSE	0,9398550725	0,3945903301
23	1600	10	1	60	FALSE	0,947826087	1,139464855
24	1800	2	1	50	FALSE	0,9449275362	1,24118309
25	1000	5	1	80	TRUE	0,9449275362	1,115774393
26	1600	10	1	80	FALSE	0,9456521739	1,307862473
27	600	2	1	80	FALSE	0,9449275362	0,7613098145
28	1200	5	1	30	TRUE	0,9456521739	0,7405808926
29	1800	10	2	50	TRUE	0,9485507246	1,358808184
30	1200	5	1	60	TRUE	0,9442028986	1,062750196
31	2000	5	2	110	FALSE	0,9471014493	1,477542734
32	600	2	2	20	TRUE	0,9485507246	0,5172332287
33	1200	5	4	None	FALSE	0,9442028986	0,6900214195
34	2000	5	1	10	TRUE	0,9456521739	0,9118843079
35	800	10	4	10	FALSE	0,9391304348	0,4864190102
36	1800	2	4	20	TRUE	0,9413043478	0,8262129784

Continúa en la página siguiente

Tabla A.3 – Continuación de la tabla de resultados en Gamma

Exp.	est.	split	leaf	depth	boot.	mean_score	mean_time
37	1600	5	1	100	FALSE	0,9442028986	1,440743017
38	2000	5	1	90	FALSE	0,9449275362	1,714548111
39	1800	2	4	None	TRUE	0,9434782609	0,8291090488
40	800	5	1	30	TRUE	0,9471014493	0,7431591034
41	1400	2	2	60	TRUE	0,95	0,9041880131
42	2000	5	2	20	TRUE	0,9463768116	0,9406842232
43	400	5	2	40	FALSE	0,947826087	0,4659571171
44	1400	2	2	None	TRUE	0,9485507246	1,164669704
45	400	5	2	100	TRUE	0,947826087	0,5286428928
46	1200	10	4	90	FALSE	0,9456521739	0,8134881496
47	1400	2	1	60	FALSE	0,9463768116	1,185765362
48	1000	10	2	20	TRUE	0,9456521739	0,6137961388
49	200	5	1	None	TRUE	0,9471014493	0,4715116024
50	2000	10	1	30	TRUE	0,9449275362	1,224455547
51	800	10	1	60	FALSE	0,9485507246	0,7946305752
52	1800	2	4	80	FALSE	0,9456521739	0,8361886978
53	1400	2	4	90	FALSE	0,9456521739	0,7707789898
54	1400	2	1	30	TRUE	0,9442028986	0,941113615
55	1800	10	1	110	FALSE	0,9449275362	1,805385113
56	1200	2	4	10	TRUE	0,934057971	0,5965269089
57	200	2	4	90	FALSE	0,9413043478	0,4291946888
58	1600	5	2	40	TRUE	0,9514492754	1,026673555
59	1200	10	4	30	TRUE	0,9384057971	0,701392746
60	1800	5	1	10	TRUE	0,9405797101	0,7435678005
61	400	2	1	40	TRUE	0,9420289855	0,5077563286
62	800	5	1	80	FALSE	0,9434782609	0,9704562187
63	1000	5	2	20	TRUE	0,95	0,7754713535
64	2000	2	4	90	TRUE	0,9427536232	0,8700842857
65	400	2	1	90	TRUE	0,9434782609	0,6684235573
66	1400	10	1	60	FALSE	0,9485507246	1,247926998
67	600	2	1	90	FALSE	0,9434782609	0,8269936085
68	600	10	1	10	FALSE	0,9413043478	0,4836103916
69	1400	5	2	40	FALSE	0,9492753623	0,9724712372
70	200	2	1	50	TRUE	0,9405797101	0,5564719677
71	1400	10	2	20	FALSE	0,9471014493	0,9234260082
72	1000	2	2	110	TRUE	0,9492753623	0,9138216019
73	400	10	4	40	TRUE	0,9347826087	0,481896162

Continúa en la página siguiente

Tabla A.3 – Continuación de la tabla de resultados en Gamma

Exp.	est.	split	leaf	depth	boot.	mean_score	mean_time
74	200	2	4	30	FALSE	0,9398550725	0,3866485596
75	800	5	2	20	TRUE	0,95	0,6563506126
76	600	2	4	50	FALSE	0,9442028986	0,5031446457
77	400	10	4	20	FALSE	0,9427536232	0,523567915
78	1000	2	2	40	FALSE	0,9528985507	0,8093825817
79	2000	2	1	110	FALSE	0,9420289855	1,908453321
80	2000	2	4	None	FALSE	0,9456521739	1,15228467
81	1200	5	1	10	TRUE	0,9376811594	0,6119464874
82	200	2	1	70	FALSE	0,9492753623	0,422005415
83	1000	10	2	70	FALSE	0,947826087	0,9117567062
84	400	5	1	30	FALSE	0,9471014493	0,4643042088
85	1200	5	2	20	FALSE	0,9456521739	0,7129608154
86	200	10	2	30	FALSE	0,9434782609	0,4031969547
87	1800	10	1	70	TRUE	0,9456521739	1,618101883
88	600	10	2	70	FALSE	0,9514492754	0,7312916756
89	1200	10	1	None	TRUE	0,9492753623	1,390295887
90	1400	5	1	80	FALSE	0,9449275362	1,191471243
91	1600	10	1	50	TRUE	0,9456521739	1,302005672
92	2000	2	4	110	FALSE	0,9471014493	0,99522686
93	1600	10	4	10	TRUE	0,9376811594	0,7040100098
94	1600	5	4	70	FALSE	0,9471014493	0,9190119743
95	1400	5	2	110	TRUE	0,9492753623	1,040880251
96	1000	5	1	90	FALSE	0,9449275362	1,01963253
97	800	5	1	70	FALSE	0,9456521739	0,8461387634
98	2000	10	4	90	FALSE	0,9456521739	0,5206749916
99	400	2	4	80	FALSE	0,9413043478	0,390965271
100	1600	10	1	10	TRUE	0,9442028986	0,509925127

Tabla A.4: Tabla de resultados de Random Forest en Delta

Exp.	**est.**	**split**	**leaf**	**depth**	**boot.**	**mean_score**	**mean_time**
1	600	5	2	40	FALSE	0,9441934277	0,4731088161
2	600	2	4	40	TRUE	0,9398188487	0,4225660324
3	2000	10	1	70	TRUE	0,9436499738	1,095665026
4	1600	2	4	30	FALSE	0,9392724006	0,673231554
5	800	10	4	80	FALSE	0,9431005315	0,5579957962
6	1200	5	2	10	TRUE	0,9398158545	0,5523999691
7	200	5	1	100	FALSE	0,9452908152	0,3967404366
8	800	10	1	60	TRUE	0,9447413729	0,6503734112
9	800	5	4	100	TRUE	0,9392709035	0,5353532791
10	200	5	2	80	FALSE	0,9491234374	0,3872976303
11	1200	5	2	70	FALSE	0,9474810989	0,681807375
12	800	2	4	40	TRUE	0,9370821169	0,5228691101
13	1000	10	4	None	TRUE	0,9387274497	0,6634518623
14	2000	2	1	80	FALSE	0,9458402575	1,077972603
15	1400	2	4	60	TRUE	0,9376315592	0,5649682522
16	1200	10	2	80	FALSE	0,9491204431	0,7383725166
17	1600	10	4	60	TRUE	0,9414626843	0,6093505859
18	1200	10	1	100	TRUE	0,9452893181	0,792686367
19	1000	2	2	10	FALSE	0,9392724006	0,5238663673
20	800	10	2	None	FALSE	0,9480290441	0,6315485001
21	1400	5	1	60	FALSE	0,9480245527	0,8319684982
22	1600	5	4	50	TRUE	0,9403667939	0,6051978588
23	400	10	4	80	FALSE	0,9436469796	0,4635856152
24	600	10	4	20	FALSE	0,9392709035	0,4286185265
25	1600	10	4	70	TRUE	0,9376315592	0,630440855
26	800	5	2	10	FALSE	0,9387274497	0,5472201347
27	600	5	4	30	TRUE	0,9387229583	0,4248067856
28	2000	2	1	30	TRUE	0,9409117449	0,7683157444
29	400	2	2	80	TRUE	0,9480245527	0,5042519093
30	2000	5	4	50	FALSE	0,9403652968	0,7511187553
31	2000	5	1	80	TRUE	0,9463867056	1,060035896
32	800	5	1	110	TRUE	0,9474810989	0,7729864597
33	800	5	4	100	FALSE	0,9447443671	0,6278532982
34	600	2	2	40	FALSE	0,9431020286	0,4621226788
35	2000	2	2	70	TRUE	0,9469331537	0,8997267246
36	1400	10	1	None	FALSE	0,9518586721	0,9914401054

Continúa en la página siguiente

Tabla A.4 – Continuación de la tabla de resultados en Delta

Exp.	est.	split	leaf	depth	boot.	mean_score	mean_time
37	1200	5	2	100	FALSE	0,9496683884	0,7028399467
38	1400	2	2	20	FALSE	0,9436469796	0,5906990528
39	200	10	2	20	TRUE	0,9392724006	0,3606499672
40	1200	10	1	30	FALSE	0,9447398757	0,664597559
41	600	10	2	90	TRUE	0,9474781046	0,6015025139
42	1000	2	2	100	FALSE	0,9491219403	0,6890263557
43	1400	10	2	40	FALSE	0,94474287	0,714234972
44	2000	10	2	90	TRUE	0,9452893181	1,114042807
45	1600	2	2	90	TRUE	0,9463852085	0,7900689125
46	200	10	2	50	TRUE	0,9431035257	0,3788417339
47	400	5	2	40	TRUE	0,9436484767	0,4373562813
48	1600	2	4	None	TRUE	0,9376300621	0,6018974304
49	1200	5	2	60	FALSE	0,9480260499	0,7152789593
50	1600	10	1	30	TRUE	0,9414596901	0,6928808689
51	1200	10	1	80	FALSE	0,9480260499	0,8532962799
52	1200	10	1	10	FALSE	0,9441934277	0,5076017857
53	1400	5	2	10	TRUE	0,9392738977	0,5470498085
54	2000	5	2	90	TRUE	0,9463837114	0,8967583656
55	1800	10	1	90	TRUE	0,9452923123	1,014734459
56	1200	2	4	110	FALSE	0,9414596901	0,6411997795
57	400	5	1	110	FALSE	0,9463896998	0,4858191013
58	1000	5	4	None	TRUE	0,9392724006	0,52770648
59	1400	2	1	60	FALSE	0,9469316566	0,7832270145
60	1600	5	2	10	TRUE	0,9387229583	0,6103971481
61	1600	10	2	None	FALSE	0,9474796018	0,8137227058
62	2000	5	1	None	TRUE	0,9502148364	1,283768559
63	1600	2	1	10	FALSE	0,9392724006	0,7354542732
64	1400	5	2	70	TRUE	0,9474781046	0,7104545593
65	2000	10	1	None	TRUE	0,9502208249	1,279654455
66	600	5	2	70	TRUE	0,9447443671	0,4838784218
67	2000	2	4	None	FALSE	0,9403667939	0,7891034603
68	1000	5	4	10	TRUE	0,9365386631	0,5063120365
69	1200	5	1	10	TRUE	0,9370851112	0,5188370705
70	800	10	4	20	TRUE	0,9381780073	0,5375162125
71	2000	10	1	50	TRUE	0,9431005315	1,177326155
72	1400	5	1	70	FALSE	0,9469331537	0,958357954
73	1600	2	2	70	FALSE	0,9469331537	1,012362051

Continúa en la página siguiente

Tabla A.4 – Continuación de la tabla de resultados en Delta

Exp.	est.	split	leaf	depth	boot.	mean_score	mean_time
74	1200	5	2	10	FALSE	0,9381765102	0,6423963547
75	1000	2	1	30	TRUE	0,9436454824	0,6276207447
76	1800	5	4	60	TRUE	0,9387274497	0,7255675793
77	1000	5	2	30	FALSE	0,9436469796	0,7143747807
78	400	5	1	50	FALSE	0,944197919	0,5366578579
79	400	10	4	None	TRUE	0,942549592	0,4627348423
80	600	2	4	80	TRUE	0,9398203458	0,5464465618
81	1400	5	2	100	FALSE	0,9496668912	0,8911111832
82	800	5	1	10	TRUE	0,9381750131	0,5010455132
83	1200	10	1	90	FALSE	0,9485754922	0,8950238228
84	1400	5	1	110	FALSE	0,9474810989	1,08250041
85	2000	10	2	100	TRUE	0,9469331537	1,035138607
86	200	2	2	110	FALSE	0,9485739951	0,4172615528
87	1000	2	1	60	TRUE	0,9409177334	0,7906825542
88	2000	2	4	100	FALSE	0,941458193	0,8721684933
89	1400	2	1	10	FALSE	0,9398158545	0,6296907425
90	400	10	2	110	TRUE	0,9458402575	0,5362170219
91	1400	2	2	10	FALSE	0,9420046411	0,601949501
92	600	5	2	60	TRUE	0,94474287	0,6306616783
93	1800	2	4	60	FALSE	0,940913242	0,8350315094
94	1400	5	2	40	FALSE	0,9458357662	0,7672872066
95	600	2	1	70	TRUE	0,9452893181	0,6481681824
96	1400	10	4	40	FALSE	0,940368291	0,7307927608
97	800	5	1	30	TRUE	0,9436484767	0,6202421665
98	200	10	2	90	FALSE	0,9485724979	0,4275726795
99	1400	10	4	None	FALSE	0,9414596901	0,7528395653
100	800	5	2	20	FALSE	0,941458193	0,5737242222

A.2. Resultados de ejecución K-Nearest Neighbors

Tabla A.5: Tabla de resultados de K-Nearest Neighbors en Alfa

Exp.	weights	p	neigh.	l_size	algorithm	mean_score	mean_time
1	uniform	2	7	40	auto	0,9388888889	0,07181653976
2	distance	2	9	40	brute	0,9527777778	0,05121130943
3	distance	1	9	40	kd_tree	0,1472222222	0,1422319412
4	uniform	2	7	10	ball_tree	0,9361111111	0,1452325344
5	uniform	1	11	30	kd_tree	0,1416666667	0,1668376923
6	distance	1	9	50	brute	0,1472222222	0,1046237946
7	distance	1	9	50	auto	0,1472222222	0,0982219696
8	uniform	2	3	40	auto	0,9416666667	0,04801063538
9	uniform	2	9	10	auto	0,9444444444	0,05101151466
10	distance	2	7	30	brute	0,9388888889	0,04886512756
11	distance	2	11	20	auto	0,9527777778	0,04921112061
12	distance	1	9	40	ball_tree	0,1472222222	0,1373827457
13	distance	2	7	20	brute	0,9388888889	0,04807081223
14	uniform	2	11	10	ball_tree	0,9416666667	0,1472334862
15	distance	1	9	20	auto	0,1472222222	0,1024227142
16	distance	2	7	40	auto	0,9388888889	0,04921097755
17	distance	2	5	10	auto	0,9444444444	0,0482108593
18	uniform	1	9	50	brute	0,1472222222	0,115226078
19	uniform	2	3	10	brute	0,9416666667	0,04921092987
20	uniform	2	3	50	ball_tree	0,9416666667	0,1362305641
21	uniform	1	3	40	auto	0,1222222222	0,1058233261
22	distance	2	3	30	brute	0,95	0,0468105793
23	distance	2	9	50	auto	0,9527777778	0,04741034508
24	distance	2	7	30	ball_tree	0,9333333333	0,1346303463
25	distance	1	5	20	auto	0,1666666667	0,09702124596
26	distance	1	11	20	ball_tree	0,1277777778	0,1404321671
27	distance	1	3	10	ball_tree	0,1277777778	0,1458329678
28	distance	2	11	10	brute	0,9527777778	0,04741034508
29	distance	2	5	20	kd_tree	0,9361111111	0,1734396935
30	uniform	2	9	20	brute	0,9444444444	0,04921116829
31	uniform	1	7	20	auto	0,1555555556	0,1082245827
32	uniform	1	7	10	ball_tree	0,1555555556	0,1492337227
33	distance	1	3	40	kd_tree	0,1277777778	0,1486334801
34	uniform	2	3	20	auto	0,9416666667	0,05141119957

Continúa en la página siguiente

Tabla A.5 – Continuación de la tabla de resultados en Alfa

Exp.	weights	p	neigh.	l_size	algorithm	mean_score	mean_time
35	uniform	1	9	30	auto	0,1472222222	0,1094239235
36	distance	1	11	10	brute	0,1305555556	0,1036225319
37	distance	1	11	40	ball_tree	0,1305555556	0,1346297264
38	uniform	1	11	20	kd_tree	0,1416666667	0,1664375782
39	distance	2	9	20	brute	0,9527777778	0,04781122208
40	uniform	2	3	50	kd_tree	0,9416666667	0,1528348446
41	distance	1	7	10	ball_tree	0,1527777778	0,1418316364
42	uniform	2	3	20	ball_tree	0,9416666667	0,1354308128
43	uniform	1	5	30	kd_tree	0,1722222222	0,1570353508
44	distance	1	5	10	auto	0,1666666667	0,1088253498
45	distance	1	11	40	brute	0,1305555556	0,09922218323
46	distance	2	11	50	ball_tree	0,9416666667	0,1296295166
47	uniform	1	3	30	kd_tree	0,1222222222	0,1572355747
48	uniform	2	7	20	brute	0,9388888889	0,04681034088
49	uniform	2	5	30	kd_tree	0,9277777778	0,173638773
50	distance	1	3	30	brute	0,1277777778	0,103423357
51	uniform	1	9	10	ball_tree	0,1472222222	0,1406319618
52	distance	1	7	10	kd_tree	0,1527777778	0,1938437939
53	distance	1	7	20	brute	0,1527777778	0,1078246117
54	uniform	2	5	10	brute	0,9333333333	0,04981341362
55	uniform	2	3	40	brute	0,9416666667	0,04801068306
56	uniform	2	5	20	auto	0,9333333333	0,05081105232
57	distance	2	7	20	kd_tree	0,9333333333	0,1728392601
58	uniform	2	5	10	ball_tree	0,9277777778	0,1434324265
59	uniform	1	7	50	kd_tree	0,1555555556	0,1459441185
60	distance	1	5	40	auto	0,1666666667	0,1026034355
61	uniform	1	9	40	ball_tree	0,15	0,1367014885
62	uniform	1	3	40	kd_tree	0,1222222222	0,1440321922
63	uniform	1	11	10	auto	0,1416666667	0,1004220486
64	uniform	2	5	40	kd_tree	0,9277777778	0,1514291286
65	distance	1	5	10	brute	0,1666666667	0,1008224487
66	uniform	2	9	40	brute	0,9444444444	0,05021123886
67	distance	1	5	40	ball_tree	0,1722222222	0,1338305473
68	uniform	2	3	40	kd_tree	0,9416666667	0,1498338699
69	distance	1	9	10	auto	0,1472222222	0,1007989407
70	uniform	2	3	30	ball_tree	0,9416666667	0,1320302963
71	uniform	2	3	10	auto	0,9416666667	0,04921050072

Continúa en la página siguiente

Tabla A.5 – Continuación de la tabla de resultados en Alfa

Exp.	weights	p	neigh.	l_size	algorithm	mean_score	mean_time
72	distance	2	5	10	brute	0,9444444444	0,04861092567
73	uniform	2	5	10	kd_tree	0,9277777778	0,2168486595
74	distance	2	3	30	ball_tree	0,9444444444	0,1330304146
75	distance	1	9	40	brute	0,1472222222	0,1004227161
76	distance	1	7	30	ball_tree	0,1527777778	0,1386322498
77	uniform	1	7	20	kd_tree	0,1555555556	0,159236145
78	uniform	1	11	20	auto	0,1416666667	0,1326289654
79	uniform	2	3	40	ball_tree	0,9416666667	0,1420320988
80	uniform	1	3	10	ball_tree	0,1222222222	0,141231966
81	distance	1	3	30	kd_tree	0,1277777778	0,1620366573
82	distance	1	3	20	auto	0,1277777778	0,1042253017
83	distance	1	7	30	kd_tree	0,1527777778	0,162637949
84	distance	1	5	30	brute	0,1666666667	0,1020236969
85	uniform	2	7	50	ball_tree	0,9361111111	0,1296294212
86	distance	2	5	40	kd_tree	0,9361111111	0,1492342949
87	uniform	1	5	10	brute	0,1638888889	0,09942202568
88	uniform	2	9	50	brute	0,9444444444	0,04841060638
89	uniform	1	7	10	auto	0,1555555556	0,1246282101
90	distance	2	11	40	brute	0,9527777778	0,04921059608
91	uniform	1	7	10	kd_tree	0,1555555556	0,1908432007
92	uniform	1	5	40	brute	0,1638888889	0,09882216454
93	uniform	1	7	50	auto	0,1555555556	0,1038228035
94	distance	1	9	50	kd_tree	0,1472222222	0,1410321712
95	uniform	2	9	50	kd_tree	0,9361111111	0,1464334488
96	uniform	1	5	50	kd_tree	0,1694444444	0,1428326607
97	distance	2	5	30	brute	0,9444444444	0,0442091465
98	distance	1	7	50	ball_tree	0,1527777778	0,1350306034
99	distance	2	11	20	kd_tree	0,9416666667	0,1722391605
100	uniform	2	5	10	auto	0,9333333333	0,04420995712

Tabla A.6: Tabla de resultados de K-Nearest Neighbors en Beta

Exp.	**weights**	**p**	**neigh.**	**l_size**	**algorithm**	**mean_score**	**mean_time**
1	uniform	2	3	10	brute	0,9310344828	0,2492563725
2	uniform	1	11	30	auto	0,1356321839	1,420521307
3	uniform	1	7	20	auto	0,1367816092	1,317897892
4	uniform	1	11	40	auto	0,1356321839	1,207773733
5	uniform	1	11	20	kd_tree	0,1344827586	1,961087036
6	distance	2	3	50	brute	0,9413793103	0,2464488029
7	distance	2	7	10	kd_tree	0,9528735632	2,994675922
8	uniform	2	3	50	kd_tree	0,9333333333	1,536146355
9	uniform	1	11	10	kd_tree	0,1356321839	2,47555871
10	uniform	2	7	30	brute	0,9482758621	0,2354535103
11	uniform	2	7	40	auto	0,9482758621	0,2396532536
12	uniform	1	7	10	auto	0,1367816092	1,237680054
13	uniform	1	3	40	ball_tree	0,1436781609	1,452721357
14	distance	2	7	50	brute	0,9540229885	0,2421981335
15	distance	2	11	40	ball_tree	0,9574712644	1,461858845
16	uniform	2	5	10	ball_tree	0,9390804598	1,639417601
17	distance	2	5	50	ball_tree	0,9379310345	1,415555716
18	distance	2	7	10	auto	0,9540229885	0,2408537865
19	distance	2	9	40	ball_tree	0,9563218391	1,467531347
20	uniform	1	11	20	ball_tree	0,1356321839	1,521939611
21	uniform	2	7	40	brute	0,9482758621	0,2434538841
22	uniform	1	11	50	ball_tree	0,1379310345	1,478132772
23	uniform	2	5	50	ball_tree	0,9390804598	1,524742842
24	distance	2	5	50	kd_tree	0,9379310345	1,729405403
25	distance	1	5	40	kd_tree	0,1367816092	1,641560459
26	distance	2	3	30	ball_tree	0,9379310345	1,423446083
27	distance	2	5	10	brute	0,9402298851	0,2367170334
28	uniform	1	11	50	auto	0,1356321839	1,170691776
29	uniform	2	7	30	kd_tree	0,9494252874	1,78700242
30	uniform	2	7	40	kd_tree	0,9494252874	1,827012634
31	distance	2	5	40	brute	0,9402298851	0,244957304
32	distance	2	11	10	auto	0,9597701149	0,2472253323
33	distance	2	5	10	auto	0,9402298851	0,2461756706
34	uniform	2	11	50	kd_tree	0,9505747126	1,600310898
35	uniform	1	3	40	kd_tree	0,1218390805	1,700583506
36	distance	1	9	30	brute	0,1275862069	1,189068842

Continúa en la página siguiente

Tabla A.6 – Continuación de la tabla de resultados en Beta

Exp.	weights	p	neigh.	l_size	algorithm	mean_score	mean_time
37	distance	2	5	10	ball_tree	0,9379310345	1,596959162
38	distance	1	9	20	brute	0,1275862069	1,217474651
39	uniform	2	9	10	ball_tree	0,9540229885	1,673577356
40	uniform	2	11	10	auto	0,9528735632	0,239253521
41	uniform	2	3	20	brute	0,9310344828	0,2446550846
42	distance	1	9	40	auto	0,1275862069	1,201071835
43	distance	2	11	50	ball_tree	0,9586206897	1,422178364
44	distance	2	11	30	auto	0,9597701149	0,2372459412
45	uniform	1	7	10	ball_tree	0,1356321839	1,653312826
46	distance	2	9	20	brute	0,9609195402	0,2325585365
47	distance	2	11	10	ball_tree	0,9574712644	1,645571518
48	distance	2	5	50	brute	0,9402298851	0,2352529526
49	distance	2	5	20	kd_tree	0,9379310345	2,228502703
50	uniform	2	5	30	ball_tree	0,9390804598	1,460729122
51	uniform	1	9	20	ball_tree	0,1402298851	1,539947462
52	distance	2	3	20	brute	0,9413793103	0,2414541721
53	distance	2	9	50	auto	0,9609195402	0,2538570881
54	distance	2	5	20	brute	0,9402298851	0,2460557461
55	distance	2	7	30	auto	0,9540229885	0,2490093708
56	uniform	1	3	50	auto	0,1344827586	1,233688498
57	uniform	1	11	20	brute	0,1356321839	1,249481487
58	distance	2	9	10	brute	0,9609195402	0,2536550045
59	uniform	1	9	30	ball_tree	0,1402298851	1,492836666
60	uniform	1	9	10	kd_tree	0,1402298851	2,86084547
61	distance	2	3	30	kd_tree	0,9379310345	1,809607649
62	uniform	2	5	20	kd_tree	0,9390804598	2,485360956
63	uniform	2	3	10	kd_tree	0,9333333333	3,023562193
64	distance	1	3	50	auto	0,1597701149	1,106588793
65	uniform	1	7	50	auto	0,1367816092	1,122652578
66	uniform	2	9	40	brute	0,9574712644	0,2426540375
67	uniform	2	3	50	auto	0,9310344828	0,2404541969
68	distance	2	7	10	ball_tree	0,9528735632	1,635768461
69	distance	1	5	30	brute	0,1275862069	1,178666878
70	distance	2	7	30	kd_tree	0,9528735632	1,779001379
71	distance	1	9	30	kd_tree	0,1275862069	1,720989323
72	uniform	1	9	10	auto	0,1402298851	1,122107077
73	uniform	1	3	10	kd_tree	0,1264367816	2,455796146

Continúa en la página siguiente

Tabla A.6 – Continuación de la tabla de resultados en Beta

Exp.	weights	p	neigh.	l_size	algorithm	mean_score	mean_time
74	distance	2	9	40	auto	0,9609195402	0,235052824
75	distance	1	11	30	kd_tree	0,1310344828	1,706185341
76	distance	1	11	30	auto	0,1333333333	1,140656757
77	distance	2	7	50	kd_tree	0,9528735632	1,541748524
78	distance	1	9	30	auto	0,1275862069	1,165063143
79	uniform	1	7	50	brute	0,1367816092	1,119452381
80	uniform	2	3	20	auto	0,9310344828	0,2418544292
81	uniform	2	7	50	auto	0,9482758621	0,2464552402
82	uniform	1	7	20	kd_tree	0,1356321839	1,938815355
83	uniform	1	11	20	auto	0,1356321839	1,207013416
84	distance	2	7	20	ball_tree	0,9528735632	1,497463179
85	distance	2	3	10	auto	0,9413793103	0,2278504848
86	distance	1	7	20	kd_tree	0,1252873563	1,933035803
87	uniform	1	5	50	auto	0,1183908046	1,138258743
88	uniform	1	7	30	auto	0,1367816092	1,086844635
89	distance	1	3	10	ball_tree	0,1620689655	1,5791574
90	uniform	2	3	10	auto	0,9310344828	0,2334604263
91	distance	2	9	50	ball_tree	0,9563218391	1,435579681
92	distance	2	7	40	ball_tree	0,9528735632	1,426922035
93	distance	1	9	50	auto	0,1275862069	1,130154943
94	distance	1	7	20	auto	0,1275862069	1,13725543
95	distance	1	7	10	kd_tree	0,1264367816	2,504965019
96	uniform	1	5	40	brute	0,1183908046	1,077843189
97	uniform	1	7	30	brute	0,1367816092	1,101048326
98	uniform	1	11	30	brute	0,1356321839	1,084442759
99	uniform	2	9	10	brute	0,9574712644	0,2406536579
100	distance	1	9	40	ball_tree	0,1275862069	1,372108412

Tabla A.7: Tabla de resultados de K-Nearest Neighbors en Gamma

Exp.	**weights**	**p**	**neigh.**	**l_size**	**algorithm**	**mean_score**	**mean_time**
1	distance	1	9	10	kd_tree	0,1710144928	7,421825361
2	uniform	1	3	30	brute	0,1050724638	4,560227871
3	uniform	1	11	50	brute	0,184057971	4,437105322
4	distance	1	9	20	auto	0,1710144928	4,203011465
5	distance	2	7	30	kd_tree	0,965942029	6,412647343
6	uniform	2	11	30	brute	0,9630434783	0,5938459873
7	uniform	1	9	20	ball_tree	0,1884057971	5,192311859
8	uniform	1	9	30	auto	0,1884057971	4,963519812
9	distance	1	9	50	brute	0,1710144928	4,276364374
10	uniform	1	9	40	brute	0,1884057971	4,288978481
11	distance	1	7	30	kd_tree	0,1702898551	6,154755592
12	distance	2	9	30	brute	0,9695652174	0,544122839
13	distance	2	5	30	auto	0,9594202899	0,5763300419
14	distance	2	9	20	brute	0,9695652174	0,5805307865
15	distance	2	11	50	kd_tree	0,965942029	6,022229767
16	distance	2	11	20	brute	0,965942029	0,5977444649
17	distance	2	3	50	ball_tree	0,9449275362	5,350806618
18	distance	1	3	10	brute	0,1195652174	4,879843521
19	distance	1	7	50	auto	0,1702898551	4,632785606
20	uniform	1	9	30	ball_tree	0,1884057971	5,451429558
21	uniform	1	9	30	brute	0,1884057971	4,41426754
22	distance	2	3	10	kd_tree	0,9449275362	8,336987019
23	uniform	1	9	40	kd_tree	0,1884057971	5,573457479
24	uniform	1	11	20	kd_tree	0,184057971	6,305325413
25	distance	1	11	20	auto	0,1855072464	4,214104176
26	uniform	1	5	40	auto	0,1369565217	3,937061882
27	distance	2	7	10	kd_tree	0,965942029	8,525723553
28	distance	2	9	50	auto	0,9695652174	0,5836082935
29	uniform	2	3	20	auto	0,9434782609	0,5705288887
30	uniform	2	3	10	auto	0,9434782609	0,5829314709
31	distance	2	9	40	brute	0,9695652174	0,5643275261
32	uniform	1	7	20	ball_tree	0,168115942	5,432327843
33	uniform	2	11	40	brute	0,9630434783	0,5803309441
34	uniform	1	3	10	kd_tree	0,1231884058	7,44367919
35	distance	2	9	20	kd_tree	0,9688405797	6,705962801
36	uniform	2	3	40	brute	0,9434782609	0,5655273438

Continúa en la página siguiente

Tabla A.7 – Continuación de la tabla de resultados en Gamma

Exp.	weights	p	neigh.	l_size	algorithm	mean_score	mean_time
37	distance	1	3	30	brute	0,1195652174	4,539024496
38	uniform	1	3	20	brute	0,1050724638	4,44960146
39	uniform	1	3	50	brute	0,1050724638	4,181423235
40	uniform	2	7	30	auto	0,9637681159	0,6115393639
41	uniform	1	5	20	auto	0,1369565217	5,137759542
42	uniform	2	5	40	auto	0,9594202899	0,5837321281
43	distance	2	11	30	brute	0,965942029	0,5850557804
44	uniform	2	5	50	auto	0,9594202899	0,5920723438
45	distance	1	11	20	ball_tree	0,1855072464	5,320416498
46	distance	2	7	50	brute	0,9688405797	0,5761293411
47	distance	1	3	50	auto	0,1195652174	4,628273535
48	uniform	1	11	30	kd_tree	0,184057971	6,134598541
49	uniform	2	11	10	ball_tree	0,9673913043	5,700485373
50	uniform	1	5	50	auto	0,1369565217	4,758103609
51	distance	2	3	30	kd_tree	0,9449275362	6,633631039
52	distance	1	11	50	auto	0,1855072464	4,08772192
53	distance	1	11	50	brute	0,1855072464	4,021098995
54	distance	2	3	30	brute	0,947826087	0,5672321796
55	distance	2	7	20	brute	0,9688405797	0,5903010368
56	uniform	1	11	10	ball_tree	0,184057971	5,732892561
57	uniform	1	7	30	kd_tree	0,168115942	6,304725647
58	uniform	1	5	20	ball_tree	0,1442028986	5,331339025
59	uniform	2	11	10	auto	0,9630434783	0,5755297661
60	distance	2	3	10	ball_tree	0,9449275362	5,777510977
61	uniform	2	7	50	kd_tree	0,9637681159	6,324707508
62	uniform	1	11	20	auto	0,184057971	4,660489941
63	uniform	2	9	50	kd_tree	0,9666666667	5,836766529
64	uniform	1	7	20	brute	0,168115942	4,452966356
65	uniform	1	7	10	ball_tree	0,168115942	5,999612665
66	distance	1	9	10	ball_tree	0,1710144928	5,558653736
67	uniform	1	11	50	ball_tree	0,184057971	5,042676973
68	distance	2	3	50	brute	0,947826087	0,5913334846
69	distance	1	5	40	brute	0,1376811594	5,426688623
70	uniform	2	9	50	auto	0,9652173913	0,5963766098
71	distance	2	5	20	kd_tree	0,9550724638	6,869418669
72	uniform	2	9	30	auto	0,9652173913	0,6054472923
73	uniform	1	7	50	auto	0,168115942	6,009435749

Continúa en la página siguiente

Tabla A.7 – Continuación de la tabla de resultados en Gamma

Exp.	weights	p	neigh.	l_size	algorithm	mean_score	mean_time
74	uniform	2	9	40	kd_tree	0,9666666667	6,941999531
75	uniform	2	7	50	ball_tree	0,9637681159	5,912734795
76	distance	1	3	30	kd_tree	0,1369565217	6,972873116
77	uniform	1	5	50	ball_tree	0,1434782609	5,796707106
78	uniform	2	5	30	kd_tree	0,9601449275	6,591387892
79	uniform	2	3	30	kd_tree	0,9434782609	6,458456945
80	distance	1	3	10	ball_tree	0,1442028986	5,21879282
81	distance	2	5	40	brute	0,9594202899	0,5651725292
82	uniform	2	5	40	kd_tree	0,9601449275	5,64075141
83	distance	2	5	30	kd_tree	0,9550724638	6,852545643
84	uniform	2	11	40	kd_tree	0,9666666667	6,694555044
85	distance	1	9	50	kd_tree	0,1710144928	6,385271597
86	distance	2	5	50	ball_tree	0,9550724638	6,114038134
87	uniform	2	7	30	brute	0,9637681159	0,604398489
88	distance	1	3	40	ball_tree	0,1442028986	6,075770569
89	distance	1	7	40	kd_tree	0,1702898551	6,446232414
90	uniform	2	3	30	auto	0,9434782609	0,578730011
91	uniform	1	5	50	kd_tree	0,1420289855	6,591904306
92	distance	2	7	50	ball_tree	0,965942029	6,037899446
93	distance	1	5	30	ball_tree	0,1405797101	6,414337254
94	distance	1	11	50	ball_tree	0,1855072464	6,088059282
95	distance	1	3	40	brute	0,1195652174	5,322162533
96	distance	1	3	30	auto	0,1195652174	4,948663378
97	distance	1	7	40	ball_tree	0,1702898551	5,69323082
98	distance	2	7	40	ball_tree	0,965942029	5,604149961
99	distance	1	9	20	ball_tree	0,1710144928	5,751697254
100	distance	1	7	50	ball_tree	0,1702898551	5,344638252

Tabla A.8: Tabla de resultados de K-Nearest Neighbors en Delta

Exp.	**weights**	**p**	**neigh.**	**l_size**	**algorithm**	**mean_score**	**mean_time**
1	distance	2	3	50	ball_tree	0,9480305412	12,16338196
2	distance	1	11	30	auto	0,132390149	12,94494343
3	distance	2	11	10	ball_tree	0,9677221349	14,50810585
4	uniform	1	3	30	auto	0,1712208998	11,66325226
5	uniform	1	3	20	brute	0,1712208998	12,45030022
6	uniform	1	5	10	brute	0,1477176435	11,41426139
7	uniform	1	11	20	brute	0,1219956584	11,11560674
8	distance	2	7	20	brute	0,9600673703	1,199208736
9	uniform	2	5	10	auto	0,9518586721	1,134670973
10	uniform	2	5	40	auto	0,9518586721	1,215837288
11	uniform	2	9	40	brute	0,9638969983	1,174464273
12	uniform	1	5	30	brute	0,1477176435	13,12428112
13	distance	2	7	40	brute	0,9600673703	1,111850977
14	distance	1	5	40	kd_tree	0,1695860469	14,83561463
15	uniform	2	11	30	brute	0,9638955012	1,137507486
16	distance	1	11	40	brute	0,132390149	12,96782808
17	distance	1	3	10	brute	0,2040482072	11,83955626
18	distance	2	9	10	brute	0,9688225167	1,122853041
19	uniform	2	3	50	ball_tree	0,9409147391	12,81862969
20	distance	2	7	50	brute	0,9600673703	1,175942993
21	distance	1	11	40	ball_tree	0,1427801482	13,69624624
22	uniform	1	7	50	brute	0,1258222921	12,5483489
23	distance	2	3	10	brute	0,9480320383	1,185756159
24	uniform	2	7	40	auto	0,9573321356	1,187267447
25	uniform	1	11	10	kd_tree	0,1236290141	23,21613569
26	distance	1	9	20	ball_tree	0,1433206078	13,88194103
27	uniform	1	5	50	auto	0,1477176435	12,84158859
28	uniform	1	5	10	auto	0,1477176435	13,41205168
29	uniform	1	7	40	auto	0,1258222921	12,00837617
30	uniform	1	11	10	auto	0,1219956584	11,57264237
31	uniform	2	9	20	auto	0,9638969983	1,195851803
32	distance	1	7	40	ball_tree	0,1433191107	13,99058375
33	uniform	1	3	50	ball_tree	0,1520787484	12,89160075
34	distance	1	11	30	ball_tree	0,1427801482	12,3786293
35	distance	1	3	10	kd_tree	0,1838101654	24,79016318
36	distance	2	5	50	ball_tree	0,9573321356	14,17471709

Continúa en la página siguiente

Tabla A.8 – Continuación de la tabla de resultados en Delta

Exp.	weights	p	neigh.	l_size	algorithm	mean_score	mean_time
37	distance	1	9	20	brute	0,1405883674	13,00578747
38	distance	2	3	20	brute	0,9480320383	1,313303757
39	distance	2	11	50	brute	0,9688180253	1,31949749
40	distance	2	3	20	kd_tree	0,9485784864	18,27673445
41	distance	1	5	30	ball_tree	0,1515188263	12,77086167
42	uniform	2	3	10	kd_tree	0,9420091324	24,80601163
43	distance	2	7	10	kd_tree	0,9589669885	25,44941545
44	distance	2	11	20	ball_tree	0,9677221349	13,85620189
45	uniform	2	11	10	brute	0,9638955012	1,250550175
46	distance	1	3	40	ball_tree	0,1657489333	16,65107722
47	distance	1	7	20	kd_tree	0,1515457744	18,37374625
48	distance	2	11	40	brute	0,9688180253	1,24819417
49	uniform	1	11	30	ball_tree	0,1329350999	14,43683877
50	distance	1	9	50	ball_tree	0,1433221049	13,29607143
51	uniform	1	9	50	kd_tree	0,1383950895	13,33419833
52	uniform	2	7	10	brute	0,9573321356	1,272943258
53	distance	2	7	30	ball_tree	0,9595149338	14,05022321
54	distance	1	9	10	ball_tree	0,1444164982	17,55196314
55	distance	1	7	30	brute	0,1394984655	14,00517836
56	distance	2	5	30	auto	0,9589774684	1,254765034
57	distance	1	3	30	brute	0,2040482072	13,47180614
58	distance	1	7	10	brute	0,1394984655	12,91402092
59	uniform	1	3	30	brute	0,1712208998	12,68046885
60	uniform	1	3	20	ball_tree	0,1526251965	13,92478967
61	uniform	1	11	40	ball_tree	0,1329350999	12,95254812
62	uniform	1	11	30	kd_tree	0,1208907852	16,18094277
63	distance	1	7	10	ball_tree	0,1444150011	14,22801676
64	uniform	2	5	50	auto	0,9518586721	1,179270267
65	distance	2	3	40	brute	0,9480320383	1,179666233
66	uniform	2	7	30	auto	0,9573321356	1,32448864
67	distance	2	5	20	ball_tree	0,9573321356	15,54744735
68	uniform	1	9	30	ball_tree	0,122536118	14,11226521
69	distance	1	11	50	auto	0,132390149	12,75683951
70	uniform	1	3	10	auto	0,1712208998	12,86865511
71	distance	1	3	50	ball_tree	0,1662968785	12,01606007
72	uniform	2	5	20	ball_tree	0,9502148364	12,99502053
73	uniform	2	3	30	kd_tree	0,9420091324	16,7218153

Continúa en la página siguiente

Tabla A.8 – Continuación de la tabla de resultados en Delta

Exp.	weights	p	neigh.	l_size	algorithm	mean_score	mean_time
74	uniform	1	7	10	kd_tree	0,1406018415	22,06149206
75	distance	2	9	30	ball_tree	0,9677251291	12,40978117
76	uniform	2	7	50	auto	0,9573321356	1,213275909
77	uniform	1	3	50	kd_tree	0,1559128677	15,67175875
78	distance	2	5	10	ball_tree	0,9573321356	15,33594799
79	uniform	2	7	20	ball_tree	0,9562302568	14,32163448
80	uniform	2	9	40	ball_tree	0,962802605	13,5273612
81	distance	1	9	10	brute	0,1405883674	10,94547806
82	uniform	1	7	30	auto	0,1258222921	10,90962
83	uniform	2	5	30	kd_tree	0,9507612845	14,56749582
84	uniform	1	9	30	kd_tree	0,1383935923	14,04539919
85	distance	2	5	20	brute	0,9589774684	1,163062334
86	distance	2	11	50	auto	0,9688180253	1,180666161
87	uniform	2	5	40	ball_tree	0,9502148364	13,84201698
88	distance	2	7	10	ball_tree	0,9595149338	15,38389468
89	uniform	1	3	40	kd_tree	0,1542720263	15,04265895
90	distance	1	5	40	ball_tree	0,1515188263	13,28391705
91	uniform	2	5	20	kd_tree	0,9507612845	18,78740692
92	uniform	1	7	20	brute	0,1258222921	11,97350101
93	uniform	1	7	20	ball_tree	0,1323796691	13,50728951
94	distance	1	3	50	brute	0,2040482072	12,00780401
95	uniform	1	5	20	auto	0,1477176435	11,48224573
96	distance	2	9	30	brute	0,9688225167	1,211060286
97	distance	1	11	40	auto	0,132390149	12,3421011
98	uniform	1	5	50	brute	0,1477176435	11,5677875
99	uniform	1	7	20	kd_tree	0,1389610001	16,59264078
100	uniform	2	3	10	auto	0,9387244554	1,027030897

A.3. Resultados de ejecución Support Vector

Tabla A.9: Tabla de resultados de Support Vector en Alfa

Exp.	shrink.	kernel	γ	deg.	c0	C	mean_score	mean_time
1	TRUE	sigmoid	scale	3	1	1000	0,9583333333	0,06345348358
2	TRUE	sigmoid	scale	2	0	10	0,9583333333	0,06901516914
3	TRUE	rbf	auto	5	1	100	0,9333333333	0,1916432381
4	FALSE	poly	auto	4	0	100	0,3083333333	0,06921491623
5	TRUE	rbf	auto	5	1	0,1	0,9333333333	0,1968435764
6	TRUE	poly	auto	5	-1	0,1	0,9333333333	0,06781601906
7	TRUE	poly	scale	5	0	1	0,5638888889	0,06581568718
8	TRUE	rbf	auto	4	1	0,1	0,9333333333	0,1806408405
9	FALSE	rbf	scale	6	-1	10	0,9583333333	0,1866425037
10	FALSE	linear	scale	2	-1	1000	0,9583333333	0,06861577034
11	TRUE	linear	scale	3	1	0,1	0,9333333333	0,06741566658
12	FALSE	linear	auto	2	-1	1	0,9583333333	0,06761469841
13	TRUE	poly	scale	4	0	1000	0,6972222222	0,06741547585
14	TRUE	sigmoid	auto	2	1	1	0,9333333333	0,06941642761
15	TRUE	poly	scale	6	0	0,1	0,2166666667	0,07161622047
16	FALSE	poly	scale	5	-1	0,1	0,9388888889	0,06720614433
17	FALSE	linear	auto	4	0	1	0,9583333333	0,06914772987
18	TRUE	poly	auto	6	0	0,1	0,9527777778	0,07381706238
19	TRUE	sigmoid	scale	5	-1	0,1	0,9277777778	0,0688158989
20	TRUE	sigmoid	scale	4	-1	10	0,9583333333	0,06541500092
21	FALSE	linear	scale	4	1	1	0,9583333333	0,0676158905
22	FALSE	linear	auto	3	-1	100	0,9583333333	0,06901578903
23	FALSE	rbf	scale	2	0	0,1	0,9277777778	0,1866419315
24	FALSE	linear	auto	2	0	1	0,9583333333	0,0682156086
25	FALSE	sigmoid	auto	4	1	10	0,9333333333	0,0665415287
26	FALSE	poly	auto	3	-1	0,1	0,9333333333	0,06919593811
27	FALSE	rbf	scale	4	0	100	0,9583333333	0,185457468
28	FALSE	sigmoid	scale	3	0	1000	0,9583333333	0,07126579285
29	FALSE	rbf	scale	6	0	1	0,9555555556	0,1772405624
30	FALSE	sigmoid	auto	6	0	1000	0,9333333333	0,0696164608
31	FALSE	poly	auto	4	-1	1000	0	0,06981573105
32	FALSE	poly	scale	5	-1	1000	0,9444444444	0,06021461487
33	TRUE	poly	scale	3	0	100	0,8277777778	0,06881599426
34	FALSE	poly	auto	3	0	10	0,4777777778	0,07041625977

Continúa en la página siguiente

Tabla A.9 – Continuación de la tabla de resultados en Alfa

Exp.	shrink.	kernel	γ	deg.	c0	C	mean_score	mean_time
35	TRUE	sigmoid	auto	6	1	10	0,9333333333	0,06881570816
36	FALSE	sigmoid	auto	5	-1	0,1	0,9333333333	0,06801595688
37	FALSE	poly	scale	4	0	0,1	0,375	0,06781582832
38	TRUE	linear	auto	5	0	0,1	0,9333333333	0,06521511078
39	TRUE	poly	scale	2	-1	100	0	0,07101640701
40	FALSE	sigmoid	scale	6	0	1	0,9555555556	0,06901597977
41	FALSE	linear	scale	2	-1	0,1	0,9333333333	0,06881566048
42	FALSE	poly	scale	3	0	10	0,8277777778	0,07041692734
43	FALSE	sigmoid	scale	2	1	100	0,9583333333	0,07021622658
44	TRUE	sigmoid	auto	6	-1	0,1	0,9333333333	0,06821517944
45	FALSE	rbf	scale	6	-1	1	0,9555555556	0,1790407181
46	TRUE	sigmoid	scale	4	-1	100	0,9583333333	0,07081604004
47	TRUE	sigmoid	auto	2	1	0,1	0,9333333333	0,06941542625
48	FALSE	poly	auto	6	1	1000	0,9583333333	0,0672150135
49	FALSE	poly	scale	2	1	0,1	0,9277777778	0,06881570816
50	FALSE	sigmoid	auto	5	1	0,1	0,9333333333	0,06861543655
51	FALSE	rbf	auto	2	1	1	0,9333333333	0,182241106
52	TRUE	sigmoid	scale	2	-1	0,1	0,9277777778	0,06481509209
53	FALSE	sigmoid	auto	4	1	1000	0,9333333333	0,06941571236
54	FALSE	sigmoid	auto	3	-1	100	0,9333333333	0,06541490555
55	FALSE	sigmoid	auto	4	0	10	0,9333333333	0,07321662903
56	FALSE	rbf	scale	3	-1	1	0,9555555556	0,1876416206
57	FALSE	rbf	scale	2	-1	1	0,9555555556	0,1912430286
58	FALSE	poly	scale	2	0	1000	0,925	0,06681571007
59	TRUE	rbf	auto	4	0	0,1	0,9333333333	0,1814411163
60	FALSE	sigmoid	auto	6	-1	10	0,9333333333	0,06881594658
61	TRUE	linear	auto	4	-1	10	0,9583333333	0,06961622238
62	TRUE	sigmoid	scale	3	-1	1	0,9527777778	0,06861553192
63	TRUE	linear	auto	6	0	1000	0,9583333333	0,06901597977
64	FALSE	poly	auto	3	1	10	0,9333333333	0,06661553383
65	FALSE	sigmoid	auto	4	-1	1000	0,9333333333	0,0680161953
66	FALSE	rbf	auto	2	-1	10	0,9333333333	0,1888341427
67	FALSE	sigmoid	scale	5	1	1	0,9305555556	0,06954555511
68	TRUE	sigmoid	auto	3	-1	0,1	0,9333333333	0,07263455391
69	FALSE	linear	scale	4	-1	1	0,9583333333	0,07361702919
70	TRUE	poly	scale	5	-1	1000	0,9444444444	0,05661258698
71	FALSE	linear	scale	2	0	100	0,9583333333	0,07041540146

Continúa en la página siguiente

Tabla A.9 – Continuación de la tabla de resultados en Alfa

Exp.	shrink.	kernel	γ	deg.	c0	C	mean_score	mean_time
72	FALSE	sigmoid	auto	6	1	1000	0,9333333333	0,07201638222
73	FALSE	rbf	scale	6	1	0,1	0,9277777778	0,1944424629
74	TRUE	sigmoid	scale	5	-1	100	0,9583333333	0,06981630325
75	TRUE	rbf	scale	6	0	1	0,9555555556	0,1782403946
76	TRUE	sigmoid	auto	6	0	0,1	0,9333333333	0,06881532669
77	FALSE	linear	auto	4	-1	10	0,9583333333	0,06717624664
78	FALSE	sigmoid	auto	6	-1	100	0,9333333333	0,07039632797
79	FALSE	sigmoid	auto	4	0	0,1	0,9333333333	0,06771159172
80	FALSE	rbf	scale	4	0	0,1	0,9277777778	0,1892420769
81	FALSE	rbf	scale	5	0	1	0,9555555556	0,1824411869
82	FALSE	poly	scale	5	1	100	0,9583333333	0,07021641731
83	TRUE	sigmoid	scale	2	0	1000	0,9583333333	0,06761512756
84	FALSE	rbf	auto	5	0	10	0,9333333333	0,2002447128
85	FALSE	linear	auto	5	1	1000	0,9583333333	0,0676153183
86	TRUE	rbf	auto	6	1	10	0,9333333333	0,1832412243
87	FALSE	poly	auto	5	-1	0,1	0,9333333333	0,06681513786
88	TRUE	poly	auto	5	-1	1	0,9333333333	0,08021774292
89	TRUE	rbf	scale	4	1	10	0,9583333333	0,2024453163
90	FALSE	rbf	scale	2	0	10	0,9583333333	0,1930121422
91	FALSE	linear	scale	2	1	1	0,9583333333	0,07001609802
92	TRUE	linear	scale	5	-1	10	0,9583333333	0,06941542625
93	TRUE	poly	scale	4	0	1	0,6972222222	0,06821599007
94	TRUE	poly	scale	2	-1	1	0	0,06981663704
95	TRUE	sigmoid	auto	4	0	0,1	0,9333333333	0,06981601715
96	TRUE	poly	auto	2	0	10	0,6972222222	0,07281675339
97	TRUE	poly	auto	6	1	0,1	0,9333333333	0,07021589279
98	FALSE	sigmoid	scale	4	-1	0,1	0,9277777778	0,07541656494
99	FALSE	linear	scale	4	-1	10	0,9583333333	0,07701768875
100	TRUE	linear	auto	2	-1	100	0,9583333333	0,06601481438

Tabla A.10: Tabla de resultados de Support Vector en Beta

Exp.	**shrink.**	**kernel**	γ	**deg.**	**c0**	**C**	**mean_score**	**mean_time**
1	FALSE	linear	scale	3	0	100	0,9540229885	1,32029109
2	FALSE	linear	scale	2	1	0,1	0,1632183908	1,326481962
3	TRUE	rbf	scale	3	0	1000	0,9459770115	2,685349846
4	TRUE	sigmoid	auto	2	-1	10	0,1632183908	1,405862093
5	TRUE	rbf	auto	5	1	100	0,1632183908	2,624591589
6	FALSE	poly	scale	2	-1	10	0,004597701149	1,419624805
7	FALSE	linear	auto	2	-1	1	0,9505747126	1,407552671
8	FALSE	poly	scale	3	1	1000	0,9471264368	1,573466444
9	FALSE	poly	auto	6	0	1	0,1620689655	1,504301405
10	FALSE	sigmoid	scale	6	1	0,1	0,1632183908	1,414087534
11	TRUE	sigmoid	auto	6	0	1000	0,1632183908	1,375825024
12	FALSE	linear	scale	6	-1	0,1	0,1632183908	1,488650036
13	FALSE	poly	auto	4	1	1	0,1632183908	1,578789949
14	TRUE	sigmoid	auto	3	1	10	0,1632183908	1,427391529
15	FALSE	rbf	scale	3	-1	0,1	0,1643678161	2,836239481
16	FALSE	linear	scale	3	1	10	0,9540229885	1,375326061
17	TRUE	poly	auto	4	1	0,1	0,1632183908	1,504375029
18	FALSE	sigmoid	auto	5	-1	1	0,1632183908	1,611781311
19	FALSE	poly	auto	5	-1	1	0,1632183908	1,479136992
20	FALSE	linear	auto	2	0	10	0,9540229885	1,610217381
21	FALSE	linear	scale	4	-1	100	0,9540229885	1,518178368
22	TRUE	poly	scale	4	1	0,1	0,9425287356	1,685204268
23	TRUE	poly	scale	6	1	10	0,9206896552	1,450144815
24	TRUE	sigmoid	auto	5	1	1	0,1632183908	1,488417578
25	TRUE	rbf	auto	3	1	100	0,1632183908	3,134225941
26	FALSE	poly	scale	2	1	100	0,9517241379	1,396301222
27	FALSE	linear	scale	2	0	1000	0,9540229885	1,364012432
28	TRUE	rbf	scale	5	-1	10	0,9459770115	3,194081593
29	TRUE	sigmoid	auto	6	-1	1000	0,1632183908	1,453800488
30	FALSE	poly	scale	4	0	1000	0,1287356322	1,422030878
31	FALSE	rbf	scale	3	-1	10	0,9459770115	3,19273839
32	TRUE	sigmoid	auto	4	0	0,1	0,1632183908	1,429584837
33	TRUE	poly	auto	4	-1	1000	0,06551724138	1,486936426
34	FALSE	linear	auto	5	1	1000	0,9540229885	1,406131077
35	FALSE	sigmoid	auto	6	-1	10	0,1632183908	1,395585155
36	FALSE	linear	scale	2	-1	1000	0,9540229885	1,393904877

Continúa en la página siguiente

Tabla A.10 – Continuación de la tabla de resultados en Beta

Exp.	shrink.	kernel	γ	deg.	c0	C	mean_score	mean_time
37	TRUE	sigmoid	scale	2	1	1000	0,9551724138	1,251482248
38	FALSE	rbf	auto	6	-1	1	0,1632183908	3,283113527
39	FALSE	rbf	auto	5	-1	1000	0,2954022989	3,707603455
40	FALSE	linear	auto	4	1	0,1	0,1632183908	1,549630213
41	TRUE	sigmoid	auto	3	0	10	0,1632183908	1,526214361
42	TRUE	sigmoid	scale	4	-1	1	0,9402298851	1,493581915
43	TRUE	poly	auto	3	1	100	0,1632183908	1,555926991
44	FALSE	linear	auto	2	-1	1000	0,9540229885	1,524253988
45	FALSE	rbf	scale	6	1	1000	0,9459770115	3,163249111
46	FALSE	poly	scale	5	0	1	0,1206896552	1,360796738
47	TRUE	rbf	auto	5	0	1	0,1632183908	3,006075525
48	TRUE	linear	auto	6	-1	0,1	0,1632183908	1,47008872
49	FALSE	linear	auto	3	-1	1000	0,9540229885	1,481791973
50	FALSE	rbf	auto	4	-1	1000	0,2954022989	3,49961257
51	TRUE	rbf	scale	2	0	100	0,9459770115	3,169158697
52	TRUE	sigmoid	auto	6	1	100	0,1632183908	1,397822762
53	TRUE	rbf	auto	2	-1	0,1	0,1632183908	3,206225204
54	FALSE	poly	auto	4	1	100	0,1632183908	1,429927683
55	TRUE	rbf	auto	2	1	10	0,1632183908	3,036358261
56	TRUE	rbf	scale	2	-1	0,1	0,1643678161	3,448949385
57	FALSE	linear	auto	6	1	1000	0,9540229885	1,378270531
58	FALSE	linear	auto	4	0	0,1	0,1632183908	1,399594116
59	FALSE	sigmoid	scale	2	0	1	0,9482758621	1,322508097
60	TRUE	sigmoid	auto	2	-1	0,1	0,1632183908	1,393814039
61	TRUE	rbf	scale	4	1	100	0,9459770115	3,510723734
62	FALSE	rbf	auto	5	-1	1	0,1632183908	3,190446997
63	TRUE	rbf	auto	4	1	1000	0,2954022989	3,273988008
64	FALSE	linear	scale	4	0	10	0,9540229885	1,418523788
65	FALSE	sigmoid	scale	5	1	1000	0,9551724138	1,307613564
66	FALSE	linear	scale	3	-1	100	0,9540229885	1,389607191
67	TRUE	rbf	scale	5	0	10	0,9459770115	3,164281511
68	FALSE	rbf	scale	5	-1	100	0,9459770115	3,230713892
69	FALSE	poly	auto	3	0	1000	0,1183908046	1,478318167
70	TRUE	poly	scale	6	0	10	0,1206896552	1,466443443
71	FALSE	sigmoid	auto	2	0	100	0,1632183908	1,421121931
72	TRUE	rbf	auto	6	0	1000	0,2954022989	3,212716341
73	FALSE	sigmoid	scale	6	-1	100	0,9471264368	1,352504969

Continúa en la página siguiente

Tabla A.10 – Continuación de la tabla de resultados en Beta

Exp.	shrink.	kernel	γ	deg.	c0	C	mean_score	mean_time
74	TRUE	rbf	auto	2	0	1000	0,2954022989	3,310133362
75	TRUE	poly	auto	6	-1	1000	0,04482758621	1,415495491
76	FALSE	linear	scale	5	1	100	0,9540229885	1,375093699
77	FALSE	rbf	auto	6	1	100	0,1632183908	3,231174803
78	TRUE	linear	scale	3	0	100	0,9540229885	1,380874062
79	TRUE	linear	scale	3	1	1	0,9505747126	1,379310703
80	FALSE	rbf	auto	6	0	10	0,1632183908	3,284995174
81	FALSE	poly	scale	2	-1	0,1	0,07011494253	1,404417372
82	TRUE	linear	auto	2	-1	10	0,9540229885	1,371591997
83	FALSE	poly	scale	5	1	0,1	0,9356321839	1,420742893
84	FALSE	rbf	auto	4	0	10	0,1632183908	3,242307425
85	FALSE	sigmoid	scale	4	1	1000	0,9551724138	1,379111385
86	TRUE	rbf	scale	6	-1	0,1	0,1643678161	3,121022224
87	TRUE	rbf	scale	6	1	10	0,9459770115	3,197094584
88	FALSE	rbf	scale	3	-1	1000	0,9459770115	3,121826935
89	TRUE	linear	scale	2	0	1000	0,9540229885	1,401755905
90	FALSE	poly	auto	6	0	100	0,1620689655	1,430161238
91	TRUE	rbf	scale	6	1	1	0,9379310345	3,115562153
92	TRUE	poly	auto	3	1	1000	0,6494252874	1,416830492
93	TRUE	poly	scale	2	-1	1	0,009195402299	1,377468157
94	FALSE	rbf	scale	5	0	1000	0,9459770115	2,971697617
95	TRUE	poly	auto	3	1	1	0,1632183908	1,421865273
96	FALSE	rbf	scale	2	-1	0,1	0,1643678161	3,019894552
97	FALSE	rbf	auto	6	0	1	0,1632183908	2,994412136
98	TRUE	poly	auto	2	0	10	0,1183908046	1,434681797
99	FALSE	sigmoid	scale	2	1	100	0,9551724138	1,180630875
100	TRUE	linear	scale	3	0	0,1	0,1632183908	0,973272419

Tabla A.11: Tabla de resultados de Support Vector en Gamma

Exp.	**shrink.**	**kernel**	γ	**deg.**	**c0**	**C**	**mean_score**	**mean_time**
1	TRUE	rbf	scale	3	-1	1000	0,9652173913	12,09467535
2	TRUE	rbf	auto	2	1	1000	0,4137681159	11,71711936
3	FALSE	rbf	auto	5	0	10	0,1369565217	11,91672025
4	TRUE	linear	auto	3	-1	1000	0,9695652174	5,556842375
5	TRUE	poly	scale	5	0	1000	0,1246376812	5,98359766
6	TRUE	rbf	scale	2	0	100	0,9652173913	11,61971755
7	FALSE	rbf	auto	6	-1	0,1	0,1369565217	11,91348801
8	FALSE	rbf	scale	6	1	100	0,9652173913	12,09041214
9	FALSE	sigmoid	scale	2	0	0,1	0,1753623188	5,99448657
10	TRUE	linear	auto	4	-1	100	0,9695652174	5,73383503
11	TRUE	sigmoid	scale	4	0	1	0,9666666667	5,428470516
12	FALSE	poly	scale	2	-1	1000	0,002898550725	5,937518454
13	FALSE	rbf	auto	6	1	1	0,1369565217	12,23749628
14	FALSE	linear	scale	4	-1	100	0,9695652174	5,605578041
15	FALSE	poly	scale	6	0	1	0,115942029	5,977476597
16	TRUE	linear	scale	3	0	1000	0,9695652174	5,71581378
17	TRUE	rbf	auto	4	0	10	0,1369565217	11,81291571
18	TRUE	poly	auto	3	-1	0,1	0,1369565217	6,059334373
19	TRUE	rbf	auto	4	1	100	0,1369565217	12,28299289
20	FALSE	sigmoid	auto	6	1	1	0,1369565217	6,097672844
21	TRUE	linear	auto	5	-1	10	0,9695652174	5,791585732
22	TRUE	poly	auto	2	-1	100	0,09202898551	6,307043791
23	FALSE	linear	auto	3	1	1	0,968115942	5,857344103
24	FALSE	poly	scale	6	1	10	0,9615942029	6,181931162
25	FALSE	linear	auto	5	0	1000	0,9695652174	5,718623066
26	TRUE	linear	auto	5	1	10	0,9695652174	5,770557022
27	TRUE	sigmoid	scale	2	1	100	0,968115942	5,050360394
28	TRUE	sigmoid	scale	5	0	1000	0,9702898551	5,61284008
29	FALSE	poly	scale	4	0	0,1	0,1144927536	6,164115286
30	FALSE	rbf	scale	2	1	1	0,9630434783	12,23366265
31	TRUE	sigmoid	scale	5	-1	100	0,968115942	6,186164474
32	FALSE	linear	scale	5	0	0,1	0,1804347826	6,27365098
33	FALSE	poly	auto	3	0	10	0,1144927536	6,119505548
34	TRUE	rbf	scale	5	0	1	0,9630434783	12,30232372
35	FALSE	sigmoid	scale	4	-1	1000	0,968115942	6,189010382
36	TRUE	rbf	scale	6	-1	100	0,9652173913	12,88464918

Continúa en la página siguiente

Tabla A.11 – Continuación de la tabla de resultados en Gamma

Exp.	shrink.	kernel	γ	deg.	c0	C	mean_score	mean_time
37	TRUE	sigmoid	scale	3	0	10	0,9702898551	5,644326496
38	FALSE	sigmoid	auto	3	-1	1	0,1369565217	6,067708921
39	TRUE	poly	auto	5	1	100	0,1369565217	6,078437138
40	TRUE	poly	scale	2	-1	10	0,004347826087	6,068691778
41	FALSE	sigmoid	auto	5	1	1	0,1369565217	6,193624735
42	FALSE	sigmoid	scale	2	1	1000	0,968115942	4,992940712
43	TRUE	linear	auto	4	-1	0,1	0,1804347826	6,235285139
44	TRUE	rbf	scale	6	0	1000	0,9652173913	12,75329485
45	FALSE	poly	scale	6	0	10	0,1166666667	6,05068388
46	FALSE	linear	scale	4	0	0,1	0,1804347826	6,051510429
47	TRUE	rbf	auto	4	0	1	0,1369565217	12,73914804
48	FALSE	sigmoid	scale	6	-1	1000	0,968115942	6,020086861
49	FALSE	sigmoid	scale	5	-1	0,1	0,1376811594	6,222086906
50	TRUE	rbf	scale	6	1	100	0,9652173913	12,43355269
51	TRUE	poly	scale	4	1	0,1	0,9644927536	6,003585005
52	FALSE	sigmoid	auto	4	0	0,1	0,1369565217	6,030617809
53	FALSE	linear	scale	5	-1	100	0,9695652174	5,687225199
54	TRUE	poly	auto	5	0	10	0,1376811594	5,983960819
55	FALSE	sigmoid	auto	6	0	100	0,1369565217	6,010395956
56	TRUE	rbf	auto	4	0	100	0,1369565217	12,79688182
57	TRUE	rbf	auto	2	1	10	0,1369565217	12,49930596
58	FALSE	sigmoid	scale	4	-1	1	0,9630434783	5,950531769
59	FALSE	poly	scale	2	-1	100	0,002898550725	6,023114872
60	TRUE	sigmoid	auto	6	0	1	0,1369565217	6,07231102
61	TRUE	poly	scale	5	1	1000	0,9630434783	6,101879072
62	TRUE	poly	scale	4	1	1	0,9644927536	5,995160532
63	FALSE	poly	scale	6	1	100	0,9615942029	5,97032485
64	TRUE	sigmoid	auto	3	0	100	0,1369565217	6,022227001
65	TRUE	rbf	auto	4	-1	10	0,1369565217	12,67675962
66	TRUE	linear	auto	6	-1	0,1	0,1804347826	6,06125102
67	TRUE	poly	auto	5	0	0,1	0,1376811594	6,095764971
68	TRUE	rbf	scale	2	0	1	0,9630434783	12,85623431
69	TRUE	sigmoid	auto	5	0	0,1	0,1369565217	6,191565943
70	TRUE	poly	scale	3	0	0,1	0,1144927536	6,11269455
71	FALSE	rbf	auto	2	-1	1	0,1369565217	12,67038832
72	FALSE	rbf	scale	3	0	0,1	0,1376811594	12,81440349
73	FALSE	sigmoid	scale	6	1	1000	0,968115942	4,923499393

Continúa en la página siguiente

Tabla A.11 – Continuación de la tabla de resultados en Gamma

Exp.	shrink.	kernel	γ	deg.	c0	C	mean_score	mean_time
74	FALSE	linear	scale	2	-1	1	0,968115942	5,76819849
75	TRUE	poly	scale	4	0	10	0,1623188406	6,044114971
76	FALSE	sigmoid	auto	4	1	10	0,1369565217	6,047916651
77	TRUE	sigmoid	auto	5	1	0,1	0,1369565217	6,022803164
78	FALSE	linear	scale	4	0	100	0,9695652174	5,735382318
79	FALSE	sigmoid	scale	6	0	100	0,9702898551	5,613148832
80	TRUE	rbf	auto	6	-1	1000	0,4137681159	13,08371043
81	TRUE	linear	auto	4	-1	1000	0,9695652174	5,745640135
82	TRUE	rbf	auto	6	0	0,1	0,1369565217	12,64999394
83	FALSE	linear	scale	4	0	1000	0,9695652174	5,666201067
84	FALSE	rbf	scale	4	0	0,1	0,1376811594	12,66476159
85	FALSE	rbf	auto	2	-1	0,1	0,1369565217	12,71936908
86	TRUE	linear	auto	5	1	1000	0,9695652174	5,633950233
87	FALSE	sigmoid	auto	5	-1	0,1	0,1369565217	5,994280958
88	TRUE	rbf	auto	6	0	1000	0,4137681159	12,72510548
89	FALSE	linear	auto	2	0	1000	0,9695652174	5,79567709
90	FALSE	rbf	scale	3	0	100	0,9652173913	13,42073755
91	FALSE	linear	auto	4	-1	10	0,9695652174	6,127027988
92	FALSE	rbf	scale	6	1	1	0,9630434783	13,20144963
93	FALSE	poly	auto	3	1	1	0,1369565217	6,195274544
94	TRUE	rbf	auto	5	0	1	0,1369565217	12,88106656
95	FALSE	poly	auto	4	-1	1000	0,08550724638	6,060430241
96	FALSE	rbf	auto	3	-1	1000	0,4137681159	12,85193796
97	TRUE	poly	auto	6	-1	1	0,09202898551	6,065441465
98	TRUE	sigmoid	auto	2	1	1	0,1369565217	6,107284355
99	TRUE	linear	auto	6	0	1000	0,9695652174	5,826881933
100	TRUE	sigmoid	scale	3	-1	0,1	0,1376811594	4,893372107

Tabla A.12: Tabla de resultados de Support Vector en Delta

Exp.	**shrink.**	**kernel**	γ	**deg.**	**c0**	**C**	**mean_score**	**mean_time**
1	FALSE	sigmoid	scale	5	1	0,1	0,389626469	11,51506772
2	TRUE	rbf	auto	6	1	10	0,3890785238	23,42590308
3	TRUE	sigmoid	auto	2	0	100	0,3890785238	11,51189237
4	TRUE	linear	scale	3	1	100	0,9677206378	10,60237293
5	TRUE	linear	scale	6	-1	1000	0,9677206378	10,43806028
6	TRUE	rbf	scale	6	-1	1	0,9666277416	23,65274215
7	TRUE	rbf	scale	2	1	1	0,9666277416	23,86719894
8	FALSE	rbf	scale	5	1	10	0,9688150311	23,76078606
9	FALSE	sigmoid	auto	3	0	0,1	0,3890785238	11,36179128
10	TRUE	linear	scale	4	1	1	0,9660812935	10,41946859
11	TRUE	rbf	auto	6	-1	0,1	0,3890785238	23,56372929
12	FALSE	linear	auto	5	1	100	0,9677206378	10,47405105
13	FALSE	linear	auto	4	0	1000	0,9677206378	10,48497448
14	TRUE	linear	scale	2	-1	10	0,9677206378	10,41131716
15	TRUE	sigmoid	auto	6	0	0,1	0,3890785238	11,48075194
16	TRUE	poly	scale	4	0	1000	0,3206317838	11,53692884
17	FALSE	poly	scale	5	0	1	0,1773096789	11,68694448
18	TRUE	linear	auto	6	-1	1	0,9660812935	10,5598392
19	TRUE	sigmoid	auto	4	0	10	0,3890785238	11,62805448
20	FALSE	sigmoid	scale	3	0	10	0,9655333483	9,974755764
21	TRUE	rbf	auto	6	0	10	0,3890785238	24,57770367
22	FALSE	sigmoid	scale	5	0	0,1	0,4514289992	11,97124581
23	FALSE	sigmoid	auto	5	1	1000	0,3890785238	11,59300108
24	TRUE	poly	scale	4	0	100	0,3206317838	11,70799823
25	TRUE	poly	auto	3	1	1000	0,8392095217	11,66333466
26	FALSE	sigmoid	auto	5	-1	1	0,3890785238	11,64714351
27	TRUE	linear	auto	3	1	100	0,9677206378	10,73891864
28	TRUE	poly	auto	3	0	0,1	0,136803653	11,66940761
29	TRUE	linear	scale	5	0	1000	0,9677206378	10,76480393
30	FALSE	linear	auto	3	-1	1	0,9660812935	10,72214732
31	TRUE	linear	auto	6	-1	10	0,9677206378	10,8067184
32	FALSE	linear	auto	2	0	0,1	0,4601811513	11,63818855
33	TRUE	linear	auto	4	-1	0,1	0,4601811513	11,75498466
34	TRUE	rbf	scale	6	1	10	0,9688150311	25,96361318
35	FALSE	linear	auto	3	-1	0,1	0,4601811513	12,10107589
36	FALSE	rbf	auto	5	1	0,1	0,3890785238	26,38915267

Continúa en la página siguiente

Tabla A.12 – Continuación de la tabla de resultados en Delta

Exp.	shrink.	kernel	γ	deg.	c0	C	mean_score	mean_time
37	FALSE	poly	scale	5	0	100	0,1773096789	11,81027799
38	FALSE	linear	auto	4	1	100	0,9677206378	10,80552244
39	FALSE	sigmoid	auto	4	1	0,1	0,3890785238	11,84979143
40	FALSE	sigmoid	auto	3	0	1000	0,3890785238	11,76529989
41	TRUE	poly	scale	3	1	1000	0,9693629763	12,09932146
42	FALSE	poly	auto	5	-1	0,1	0,3890785238	11,80834246
43	FALSE	rbf	scale	3	1	100	0,9688150311	26,25772128
44	FALSE	linear	scale	4	-1	1	0,9660812935	10,78028088
45	FALSE	poly	scale	4	1	100	0,9677221349	11,72379193
46	FALSE	rbf	scale	5	1	100	0,9688150311	26,58261633
47	FALSE	poly	scale	6	-1	1000	0,001092896175	12,17426915
48	TRUE	rbf	scale	3	-1	100	0,9688150311	26,44064398
49	FALSE	poly	scale	6	-1	10	0,001092896175	11,78042583
50	TRUE	linear	auto	2	-1	1000	0,9677206378	10,86786489
51	TRUE	rbf	scale	5	1	0,1	0,3874376825	26,60568123
52	TRUE	linear	scale	3	0	100	0,9677206378	10,8458653
53	FALSE	poly	auto	5	1	1	0,3890785238	12,18927722
54	TRUE	poly	scale	2	-1	10	0,00164084138	11,8630043
55	TRUE	linear	scale	6	0	0,1	0,4601811513	11,89285874
56	TRUE	poly	scale	5	-1	1	0,9606123213	6,384427261
57	TRUE	sigmoid	auto	4	-1	1000	0,3890785238	11,60051756
58	TRUE	poly	auto	3	0	1000	0,1400913242	11,67260489
59	TRUE	linear	auto	5	0	1	0,9660812935	10,66869116
60	TRUE	rbf	scale	3	1	1000	0,9688150311	25,49242997
61	TRUE	poly	scale	5	0	1	0,1773096789	11,60712004
62	TRUE	sigmoid	scale	2	0	0,1	0,4514289992	11,65178289
63	TRUE	poly	auto	5	0	10	0,407139756	11,73099566
64	FALSE	linear	auto	4	1	1000	0,9677206378	10,58913803
65	TRUE	sigmoid	auto	2	-1	1	0,3890785238	11,79667959
66	FALSE	sigmoid	auto	2	0	0,1	0,3890785238	11,56612902
67	TRUE	sigmoid	auto	6	1	10	0,3890785238	11,60964069
68	TRUE	linear	scale	6	0	10	0,9677206378	10,65602288
69	FALSE	linear	scale	4	-1	10	0,9677206378	10,61193085
70	TRUE	linear	scale	3	-1	100	0,9677206378	10,64772897
71	FALSE	linear	scale	3	0	100	0,9677206378	10,73649158
72	FALSE	poly	auto	3	1	1000	0,8392095217	11,61196647
73	FALSE	rbf	auto	6	-1	1	0,3890785238	25,8028564

Continúa en la página siguiente

Tabla A.12 – Continuación de la tabla de resultados en Delta

Exp.	shrink.	kernel	γ	deg.	c0	C	mean_score	mean_time
74	TRUE	linear	scale	5	1	10	0,9677206378	10,76685634
75	FALSE	sigmoid	scale	2	0	1000	0,9655333483	10,05398502
76	FALSE	poly	scale	3	1	1000	0,9693629763	11,52319455
77	FALSE	linear	scale	2	1	100	0,9677206378	10,6537992
78	TRUE	poly	scale	6	-1	1000	0,001092896175	11,56432776
79	TRUE	linear	auto	6	1	1000	0,9677206378	10,67824554
80	TRUE	poly	auto	6	1	1	0,3890785238	11,717061
81	FALSE	linear	scale	4	1	1	0,9660812935	11,08233342
82	TRUE	linear	auto	6	-1	0,1	0,4601811513	11,86296701
83	FALSE	rbf	auto	4	0	0,1	0,3890785238	27,55002551
84	FALSE	rbf	scale	6	-1	1000	0,9688150311	27,76664577
85	TRUE	sigmoid	scale	6	0	100	0,9655333483	11,50899959
86	FALSE	rbf	auto	6	0	100	0,3890785238	27,02945333
87	TRUE	rbf	auto	2	1	1000	0,5384040722	27,89832683
88	TRUE	poly	auto	3	1	10	0,3890785238	12,7527966
89	TRUE	sigmoid	scale	6	1	100	0,9633445617	10,03159556
90	TRUE	poly	auto	3	-1	100	0,3890785238	13,41088133
91	FALSE	sigmoid	scale	2	1	10	0,9633445617	9,477033806
92	TRUE	rbf	scale	2	1	0,1	0,3874376825	26,47317653
93	FALSE	rbf	auto	2	0	0,1	0,3890785238	26,90845623
94	FALSE	linear	auto	5	0	1	0,9660812935	11,35320516
95	FALSE	linear	scale	5	0	0,1	0,4601811513	12,26572809
96	TRUE	sigmoid	scale	4	0	1000	0,9655333483	10,69949169
97	TRUE	linear	auto	2	-1	10	0,9677206378	11,55373039
98	TRUE	rbf	scale	4	1	1000	0,9688150311	26,9984292
99	TRUE	linear	auto	2	-1	0,1	0,4601811513	12,68990583
100	TRUE	sigmoid	auto	6	1	1	0,3890785238	10,23019338