Integración de ERPs para Finanzas y RRHH en entornos cloud

Asier Cardoso Sánchez

Doble grado en Ingeniería Informática y Matemáticas
Facultad de Informática
Universidad Complutense de Madrid

Trabajo de Fin de Grado

15 de junio de 2017

Directores:

Gonzalo Méndez Pozo
Javier Delgado Taretto

Dedicado a
mi familia

Agradecimientos

Me gustaría agradecer a las personas que han hecho posible este trabajo:

En primer lugar a Javier Delgado por permitirme la posibilidad de realizar este trabajo con un proyecto de *Business Network Builders* (BNB). También a Gonzalo Méndez que cuando le propuse ser mi tutor en el trabajo, aceptó encantado.

A Carmen Fernández que fue la primera profesora con la que contacté para intentar realizar el trabajo en empresa. Gracias por resolverme esas dudas iniciales.

Cómo no, también quiero mostrar mi agradecimiento a mi familia y a mi novia, que han sabido soportarme en estos años de carrera y siempre que lo he necesitado han estado ahí para ayudarme.

También quiero agradecer a todos los profesores que durante la carrera se han esforzado en enseñarnos la belleza de las matemáticas y la informática.

Por último pero no por ello menos importante, quiero hacer un agradecimiento especial a mis compañeros, o mejor dicho amigos, de carrera, con los que he compartido memorables experiencias durante estos cinco años. Sin duda alguna es una de las cosas más valiosas que me ha brindado la Universidad.

Gracias

Resumen

Es una buena práctica intentar tener unificadas en una aplicación múltiples funcionalidades. Sin embargo hay ocasiones en las que esto no es posible, o poco conveniente. En estos casos nos vemos obligados a usar más de una aplicación que necesitan comunicarse y relacionarse entre sí.

En estas situaciones, a pesar de que las aplicaciones suelen proporcionar herramientas que facilitan este trabajo de integración, todavía se necesita desarrollar esta caja negra que sirva de puente entre ambas aplicaciones.

Este trabajo de fin de grado se ha realizado en colaboración con la empresa *Business Network Builders (BNB)*. En este trabajo de fin de grado, se han integrado dos aplicaciones *cloud*. La primera de ellas es el *Enterprise Resource Planning* (ERP) *Workday*, que la empresa pretende empezar a usar. La segunda es el *Customer Relationship Management* (CRM) *HubSpot* que se quiere continuar usando, el cual permite llevar un seguimiento de las oportunidades de negocio de la empresa. El trabajo consiste en el desarrollo de una Interfaz que actúe de puente entre ambas aplicaciones y conseguir la integridad de los datos entre las dos.

Para ello se ha desarrollado la Interfaz de integración entre *HubSpot* y *Workday*. Esta Interfaz escucha los mensajes enviados desde *HubSpot*, los trata y realiza las operaciones de integración correspondientes en *Workday*.

Adicionalmente en el trabajo se ha realizado un estudio de la plantilla de empresa. Con las conclusiones de este estudio y usando técnicas de *Machine Learning* se ha construido un modelo basado en la información de los empleados. Este modelo es capaz de predecir futuros casos de empleados que abandonan el trabajo.

Palabras clave
Workday, Hubspot, Integración, *Enterprise Resource Planning* (ERP), *Customer Relationship Management* (CRM), Cloud, Análisis de datos, Machine learning.

Abstract

It's a good practice to have several functionalities in one application, instead of having them distributed in multiple ones. But there are some cases in which it's not possible to accomplish that, or maybe it's simply not recommended. In these situations we are forced to use multiple applications that should be able to interact with each other.

Applications usually provide services to make these integrations easier. However a lot of work is still necessary, in order to build this black box that will function as bridge between the applications.

This project has been carried out in collaboration with the company *Business Network Builders* (BNB). In this final degree project, I have integrated two cloud applications. One of them is *Workday* a relatively recent cloud based *Enterprise Resource Planning* (ERP). And the other is *HubSpot*, a cloud based *Customer Relationship Management* (CRM). Which allows users to keep track of the different business opportunities a company has.

I have developed the Interface *HubSpot-Workday* that listens to the automatic messages sent by Hubspot, processes them and then delivers the corresponding messages to *Workday*. When a deal or business opportunity is created or modified in *HubSpot*, our interface receives and processes the message, and then sends *Workday* requests to execute the corresponding operations.

As a complementary part of the project, I have studied the data of a company's personnel. With the conclusion of this study and different *Machine Learning* techniques, I have built a model that is able to predict future cases of job attrition based on the employee's information.

Keywords
Workday, Hubspot, Integration, *Enterprise Resource Planning* (ERP), *Customer Relationship Management* (CRM), Cloud, Data analysis, Machine learning.

Índice

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Motivaciones

Este trabajo de fin de grado se ha realizado en colaboración con la empresa *Business Network Builders* (BNB). BNB es una empresa de consultoría que proporciona a los clientes soluciones para distintas aplicaciones de negocio: *SAP*, *Workday*, *PeopleSoft*, *JD Edwards*.

Este trabajo de fin de grado es parte de un proyecto, concretamente es una de las integraciones del proyecto. El proyecto parte de la idea de mejorar el *software* que se usa internamente para la parte de recursos humanos y finanzas. Se requiere facilitar las labores internas. Comenzar a usar un ERP es una forma de conseguir reunir las funcionalidades de recursos humanos y finanzas en una aplicación. Dado que *Workday* es un ERP moderno y opera en la nube, se trata de la opción que más encaja para la empresa y la que se eligió.

Una vez fijado el proyecto surgen infinidad de tareas. Configurar *Workday* con todas las opciones y funcionalidades que desea tener la empresa, migrar los datos de aplicaciones previas como *QlikView* o *CM Plan*, integrar *Workday* con otras aplicaciones con las que tiene que coexistir cuando se salga a producción...

Las aplicaciones que se deben integrar con *Workday* son *Mantis Bug Tracker* y *HubSpot*.
Este trabajo de fin de grado consiste en la integración de *Workday* con *HubSpot*.
HubSpot es el CRM que usa actualmente la empresa y quiere mantenerlo tras la salida a producción.

El proceso de salida a producción debe ser rápido, ya que la empresa debe continuar con sus actividades empresariales. Los procesos de migración y puesta en marcha de las integraciones tienen que hacerse en los días previos a la salida a producción. Por ello, uno de los grandes retos que supone el proyecto es la salida a producción con el menor impacto posible en las actividades de la empresa.

Adicionalmente, por parte de la empresa se me solicitó hacer un prototipo de aplicación capaz de predecir la rotación de plantilla de la empresa, y así ser capaz de prever cuando un empleado va a abandonar la empresa.

Con el desarrollo de este prototipo se quiere tomar las acciones necesarias para realizar una mejor predicción.

1.2. Objetivos

En esta sección vamos a establecer los objetivos a cumplir.

- Desarrollar un microservicio que realice la integración entre *HubSpot* y *Workday*.
- Conseguir que al introducir datos en *HubSpot*, automáticamente se sincronicen con *Workday*.
- Que al modificar datos en *HubSpot*, se modifiquen en *Workday* de forma automática.
- Que la integración sea rápida, y el tiempo transcurrido entre la introducción o cambio de datos en *HubSpot* y los cambios en *Workday* sea el mínimo posible.
- La integración ha de ser segura. Estar provista de mecanismos para evitar posibles ataques de terceras personas.
- La integración debe ser totalmente transparente para el usuario.
- El programa no debe abortar su ejecución de manera inesperada.
- En la medida de lo posible evitar errores en la introducción de datos. Evitar que se cree información duplicada.
- El servicio que realiza la integración tiene que estar en ejecución ininterrumpida.
- Garantizar la sincronización de aquellos datos que cumplen los requisitos para ser integrados.
- Localmente se debe llevar la cuenta de los datos que se encuentran integrados.
- El servicio debe soportar ser reiniciado.
- El prototipo predictor debe ser capaz de calcular la probabilidad de que un empleado abandone la empresa.
- Tras el estudio se debe concluir que características son más importantes para predecir si un empleado va a abandonar el trabajo.
- Poder tomar medidas de acuerdo a las conclusiones obtenidas del estudio que den lugar a mejores predicciones.

1.3. Plan de trabajo

Para la realización del proyecto de transición a *Workday* se ha usado una metodología ágil. Cinco personas formábamos parte del equipo de proyecto y nos reuníamos al menos una vez por semana. El proceso ha durado unos tres meses y todos los integrantes realizábamos paralelamente tareas en otros proyectos.

En estas reuniones, se informaba de dónde nos encontrábamos y cuáles eran los siguientes pasos. Por turnos se intervenía para explicar los avances que habíamos hecho hasta el momento y los problemas a los que nos estábamos enfrentando. Las reuniones también servían para tomar decisiones sobre el proyecto.

En cuanto al desarrollo de la Interfaz de integración entre *HubSpot* y *Workday* se fue realizando de forma iterativa e incremental. Además, al tratarse de un grupo reducido, cualquier consulta o duda sobre los requisitos de la Interfaz podía ser solventada rápidamente.

Durante las últimas semanas se planificó mucho más detalladamente las tareas de cada uno, así como el momento en el que estas tareas debían ejecutarse. Fue en estas últimas semanas cuando se necesitó la colaboración de otros compañeros de trabajo en el proyecto.

El desarrollo de la Interfaz de integración entre *HubSpot* y *Workday* se ha realizado de forma incremental.

En cada reunión se comunicaban los avances realizados y se proponían los objetivos a cumplir para el próximo encuentro. Tras la reunión se comprobaban las nuevas funcionalidades implementadas.

En las semanas posteriores a la salida a producción, gracias a las observaciones e incidencias anunciadas por los usuarios, se ha continuado modificando la Interfaz *HubSpot-Workday*.

Para mantener un historial de los cambios y controlar las versiones de la Interfaz *HubSpot-Workday* he usado le herramienta de control de versiones *Git*.

Para la realización del estudio de los datos y la elaboración del modelo predictor he trabajado de forma individual. He ido realizando las pruebas y código con el editor interactivo *Jupyter Notebook*. A la hora de decidir el modelo a usar, he seguido una mecánica de ensayo y error, comparando los resultados arrojados por los distintos modelos.

Capítulo 2

Trabajo relacionado

En este capítulo voy a describir el contexto en el que se encuentra este trabajo de fin de grado. Explicaremos qué es un *Enterprise Resource Planning* (ERP) y algunos de los más usados. También explicaremos qué es un *Customer Relationship Management* (CRM) y varios ejemplos. Por último se da una introducción a *Machine Learning*.

2.1. Enterprise Resource Planning

Un *Enterprise Resource Planning* (ERP) o Sistema de planificación de recursos empresariales es una aplicación usada para recoger, almacenar, interpretar y tratar grandes cantidades de información sobre actividades empresariales. Podríamos decir que un ERP reúne en una aplicación las funcionalidades necesarias para desarrollar las distintas actividades empresariales de una organización o empresa.

Además este sistema ayuda a evaluar y controlar de manera más fácil un negocio. También permite la automatización de tareas repetitivas y una mejor comunicación entre las distintas áreas que componen la empresa.

Los ERP brindan información que permite a las empresas tomar decisiones que ayuden a cumplir los objetivos.

Sobre los años 60 y principios de los 70, antes de que los ERP surgieran, los sistemas utilizados por las empresas eran los llamados MRP y MRP II. Los *Materials Requirement Planning* (MRP) son sistemas de planificación y administración de la producción de una empresa. Estos tipos de sistemas se centraban en aspectos relacionados con el inventario y no llegaban a abarcar muchas de las áreas importantes en una empresa.

El crecimiento de la información y las tecnologías de comunicaciones llevó a la creación de sistemas más robustos para las organizaciones. A finales de los años 80 y principios de los 90 comenzó a aparecer en el mercado un nuevo sistema software conocido como ERP, destinado principalmente a grandes y complejas organizaciones de negocios. Estos sistemas se caracterizaban por ser propietarios, caros, complejos y con gran potencial. Los ERPs son sistemas que requieren de consultoría para implementar soluciones a medida, que se basen en las necesidades de la empresa u organización.

Los ERP no surgieron de la noche a la mañana, este proceso llevó tiempo y todavía siguen evolucionando. Si por algo se han caracterizado los ERP es por su transformación y continuo cambio. [HRP03, Wik17, Bur15]

2.1.1. SAP ERP

SAP SE (*Systems, Applications and Products in Data Processing*) es una compañía alemana fundada en 1972. Conocida principalmente por el desarrollo del ERP *SAP*.

SAP fue construido partiendo de un *software* previo, llamado *SAP R/3* y fue lanzado el 6 de julio de 1992. En torno al año 2000 *SAP* se convirtió en la tercera compañía más grande dedicada al sector de los *ERPs*.

Sus principales objetivos es dar soluciones a empresas pertenecientes a los sectores químico, servicios públicos, la venta al por menor, farmacéuticas...

Los puntos fuertes de *SAP* com ERP es proporcionar un software a las fábricas y a la parte administrativa de las empresas.

El ERP *SAP* se trata de un *software On-premise*[1]. Sin embargo uno de sus enfoques está siendo dar importancia a las soluciones *cloud*, y prueba de ello es el proyecto *SAP HANA*.

Un inconveniente es que su implantación supone un riesgo, por la inversión económica y la transición a la nueva aplicación. La amortización de *SAP* puede llevar mucho tiempo.

2.1.2. PeopleSoft

PeopleSoft fue fundado por David Duffield en 1987 y adquirido por *Oracle* en 2005. Es un ERP moderno especializado en los módulos de recursos humanos y finanzas.

PeopleSoft es un ERP *On-premise*. Originalmente *PeopleSoft* ofrecía soluciones para recursos humanos y finanzas.

Peoplesoft permite tener las funcionalidades separadas en módulos independientes, pero también facilita que estos módulos trabajen conjuntamente. A diferencia de *SAP*, *PeopleSoft* puede ser usado para la implementación de un único módulo, como por ejemplo administración de estudiantes o gestión de capital humano.[And01]

En sus últimas versiones cuenta con FLUID, que aporta la posibilidad de tener una interfaz de usuario *responsive* dedicada especialmente a dispositivos móviles y *tablets*.

2.1.3. Workday

Workday fue fundado en 2005 por David Duffield, fundador y actual CEO del ERP *PeopleSoft*.

Se trata de un ERP totalmente *Cloud*. Permite a las empresas y organizaciones despreocuparse de los equipos informáticos que ejecutan el software. Se trata claramente de un aspecto diferenciador frente a ERP *on-premise* como *Peoplesoft*. *Workday* está destinado al mismo tipo de clientes que *SAP*, y por tanto se trata de una de las alternativas.

Workday abarca los campos de finanzas y recursos humanos. Cuenta con una interfaz simple y moderna, con facilidades para su uso en dispositivos móviles.

Con *Workday* siempre se hace uso de la última versión y va siendo actualizado constantemente.

[1]Software On-premise es instalado y ejecutado en el edificio (*on the premises*) de la persona u organización que hace uso del software.

2.2. Customer Relationship Management

Un *Customer Relationship Management* (CRM) es una aplicación o sistema usado para satisfacer las necesidades de los clientes durante cualquier interacción con los mismos.

Los CRM ayudan a las empresas a aumentar su rentabilidad e ingresos, logrando atraer y retener clientes. Permiten un seguimiento de las relaciones de la empresa con los clientes, así como las reuniones, acuerdos...

2.2.1. Salesforce

La empresa *Salesforce* fue fundada en marzo de 1999 por un antiguo ejecutivo de Oracle Marc Benioff. La empresa es conocida por su famoso CRM *Salesforce*. *Salesforce* es un CRM totalmente *cloud*. Por tanto se puede acceder a *Salesforce* desde un navegador. Gracias a *Salesforce* se consiguen automatizar numerosos procesos que ayudan a administrar las relaciones con contactos y empresas.

La configuración inicial del CRM *Salesforce* puede ser complicado y requerir gran cantidad de tiempo.

2.2.2. HubSpot

HubSpot es un CRM fundado por Brian Halligan y Dharmesh Shah en 2006. Su objetivo está orientado a proporcionar herramientas para aumentar la rentabilidad de las oportunidades con los clientes. Proporciona mecanismos para la integración con otras herramientas como el correo electrónico o las redes sociales en el ámbito empresarial. *HubSpot* también actúa como gestor de contenido y como plataforma para poder analizar los datos.

Inicialmente estaba pensado para pequeñas empresas, pero poco a poco se ha ido adaptando a empresas más grandes.

2.3. Introducción a *Machine Learning*

Para muchos problemas existen algoritmos con los cuales podemos resolverlos. Sin embargo existen otros tipos de problemas para los cuales no contamos con un algoritmo. *Machine Learning* o Aprendizaje automático es una rama de la inteligencia artificial que explora la construcción de algoritmos que pueden aprender y realizar predicciones sobre los datos.

Tiene como objetivo predecir información de datos desconocidos habiendo sido entrenado con un conjunto de datos.[sci, SSBD16, Alp10]

Los problemas de aprendizaje se clasifican en dos grandes grupos dependiendo de si contamos en los datos con la información que se pretende predecir.

- **Aprendizaje supervisado**: En este tipo de aprendizaje existe una característica con la información que se quiere predecir. Dentro de este tipo de aprendizaje se encuentran dos tipos de problemas:

  - **Clasificación**: Este problema consiste en clasificar cada muestra en una clase. Si solo existen dos clases, el problema se dice de clasificación binaria. En otro caso se denomina problema multiclase.

    Un ejemplo de problema de clasificación es el siguiente: Dada una foto de una persona, determinar si la persona está sonriendo. Se trataría de un problema supervisado pues suponemos que contaríamos con datos iniciales donde se indicase si la persona está o no sonriendo.

  - **Regresión**: Problemas en los que la característica que se quiere predecir es continua, no toma valores discretos. Un ejemplo de problema de regresión sería el siguiente: dada una foto de una persona averiguar la edad de ésta.

- **Aprendizaje no supervisado**. En este tipo de aprendizaje los datos de entrenamiento no cuentan con una característica que indique la información que se quiere predecir. Puede usarse con varios objetivos, aunque el más común es clasificar los datos en grupos con similitudes.

Capítulo 3

Integración de *HubSpot* y *Workday*

En este capítulo vamos a ver como funciona el sistema de integración. Para ello explicaremos en detalle cada uno de los componentes del sistema.

Si nos fijamos en la Figura 3.1 podemos ver como la Interfaz *HubSpot-Workday* actúa como puente entre *HubSpot* y *Workday*.

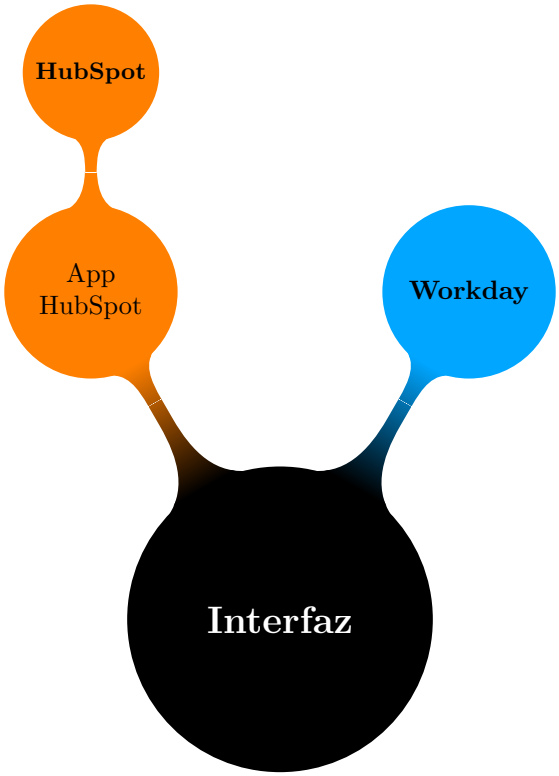


Figura 3.1: Diagrama general del sistema
Modificación de ejemplo de *TikZ* (*Till Tantau*)

Explicaremos como funcionan *HubSpot* y *Workday*. Como están organizados y cual es su estructura. También explicaremos los métodos de integración con los que cuenta cada una de las aplicaciones. Por último, explicaremos en detalle el diseño de la Interfaz *HubSpot-Workday* así como sus características.

La Interfaz *HubSpot-Workday* esta conectada tanto a *HubSpot* como a *Workday*.

A *HubSpot* se conecta a través de una aplicación que necesitamos crear en *HubSpot* y la comunicación puede ser bidireccional. De la Interfaz *HubSpot-Workday* hacia *HubSpot* se hace

mediante peticiones HTTP a la API de *HubSpot*. La comunicación en la otra dirección se realiza mediante los *webhooks* declarados en la aplicación de *HubSpot*

Por otro lado está *Workday* cuya comunicación es unidireccional. Únicamente existe comunicación propiamente dicha de la Interfaz *HubSpot-Workday* hacia *Workday*. *Workday* sí que responde a algunas peticiones HTTP con información, pero nunca inicia un mensaje hacia la Interfaz *HubSpot-Workday*.

3.1. *HubSpot*

En esta sección vamos a describir toda la información que necesitamos saber sobre *HubSpot* para entender la Integración.

Primero debemos tener en cuenta que *HubSpot* se encuentra organizado en objetos. Y toda la información almacenada en *HubSpot* pertenece a uno de los siguientes tipos de objetos:

- *Contacts* o Contacto
- *Companies* o Compañía
- *Deals* u Oportunidad

Cada objeto tiene una serie de propiedades por defecto a las que se pueden añadir propiedades personalizadas. Es importante tener en cuenta las relaciones posibles entre los distintos tipos de objetos.

- *Contacts*
  - Un contacto puede ser asociado con un único objeto compañía.
  - Un contacto puede ser asociado con múltiples objetos oportunidad.
- *Companies*
  - Una compañía puede estar asociada a múltiples contactos.
  - Una compañía puede estar asociada a múltiples oportunidades.
- *Deals*
  - Un *deal* puede estar asociado a múltiples objetos contacto y a múltiples objeto compañía.

En *HubSpot* un *deal* puede estar asociado a una o más *companies*. Para el caso de nuestra integración a lo sumo habrá un *company* asociado a cada *deal*.

Los objetos que nos interesan para nuestra integración son *deal* y su *company* asociado. La creación o modificación de un *deal* iniciaran procesos de integración.

3.1.1. Descripción del objeto *Deal*

Es un objeto de *HubSpot* que representa una oportunidad de negocio con cierto cliente. Por defecto en *HubSpot* tenemos las siguientes propiedades asociadas a un *deal*:

- **Deal Id**: Identificador unívoco del *deal* de *HubSpot*. Esta propiedad se genera automáticamente cuando un *deal* es creado.
- **Amount**: Cantidad de dinero para la oportunidad de negocio.

- **Close Date**: Se trata de la fecha en la que se ha cerrado el acuerdo.
- **Closed Lost Reason**: Comentario explicando la razón de la pérdida de la oportunidad.
- **Closed Won Reason**: Comentario explicando la razón por la que se ha ganado la oportunidad.
- **Create Date**: Fecha de creación del *deal*.
- **Deal Description**: Información describiendo el *deal*.
- **Deal Name**: Nombre del *deal*.
- **Deal Stage**: Estado en el que se encuentra el *deal*.

  Esta propiedad puede tomar los siguientes valores.

  - *Appointment Scheduled*: Cuando se ha fijado una cita con el cliente.
  - *Initial Contact*: Si ya se ha tenido un primer contacto con el cliente.
  - *Opportunity Identfied*: En caso de que se haya identificado una oportunidad de negocio con el cliente.
  - *Preparing Proporsal*: Si se está preparando una proposición de oferta al cliente.
  - *Proporsal Sent*: Una vez se haya enviado la proposición de oferta al cliente.
  - *Decision Maker Bought In*: Una vez la oferta ya esta hecha y el cliente tiene la posibilidad de decidir si la acepta o no.
  - *Contract Sent*: Cuando el contrato se ha enviado al cliente.
  - *Closed Won*: Si la oportunidad de negocio se ha cerrado favorablemente.
  - *Closed Lost*: En caso de que se haya perdido la oportunidad con el cliente.
- **Deal Type**: Tipo de *deal*.
- **Hubspot Owner**: *Customer Manager* al que le pertenece la oportunidad de negocio.
- **Last Activity Date**: Fecha de la última actividad realizada.
- **Last Modified Date**: Fecha de la última modificación.
- **Number of Contacts**: Número de contactos asociados al *deal*.
- **Opp number**: Identificador manual de la oportunidad. Puede ser usado con distintos fines.
- **Owner Assigned Date**: Fecha en la que el *deal* se asigno a un *Customer Manager*.
- **Pipeline**: Serie de estados por los que pasa un *deal* con la probabilidad de éxito en cada estado. En la empresa solo hay un *pipeline* definido.

Adicionalmente en la empresa se han creado propiedades personalizadas:

- **Practice**: Practica a la que pertenece el *deal*. Esta propiedad puede tomar los siguientes valores: *PeopleSoft*, *SAP*, *Workday*...
- **Transaction Amount**: Cantidad de dinero en la transacción del *deal*.
- **Transaction Currency**: Moneda utilizada para la oportunidad de negocio.

Es de especial importancia la propiedad *Deal Stage*, ya que dependiendo de su valor se realizará o no la sincronización. La propiedad *Deal Stage* representa el estado en el que se encuentra un *deal*.

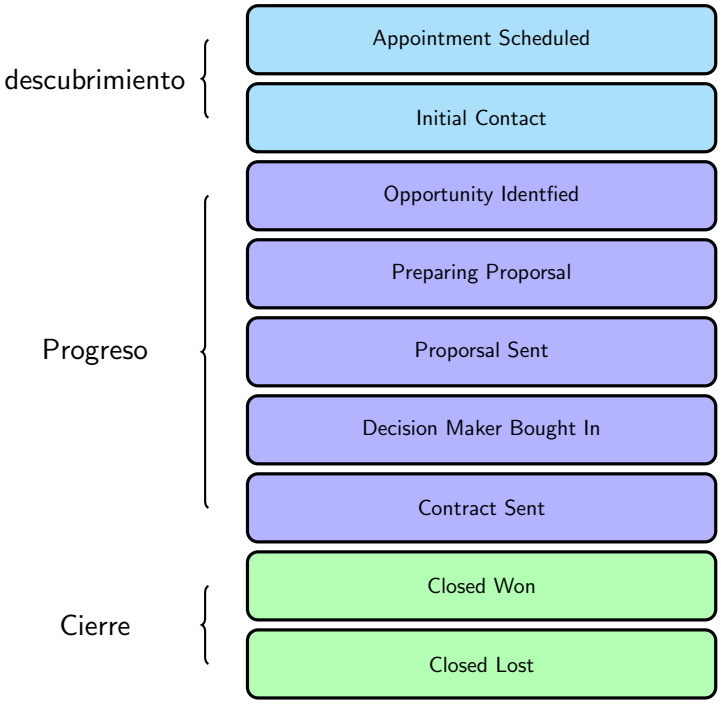


Figura 3.2: Estados por los que pasa un deal
Modificación de ejemplo de *TikZ* (*Erno Pentzin*)

Un *deal* pasa por tres fases.

1. **Fase de descubrimiento**: Fase inicial del *deal*. Cuando un *deal* se encuentra en esta fase, no se sincroniza con *Workday*. La información del *deal* solo reside en *HubSpot*.

2. **Fase de progreso**: fase intermedia del proceso en la que todavía esta por decidir si la oportunidad se va a ganar o no. Todos los *deals* en esta fase se mantienen sincronizados con el correspondiente proyecto de *Workday*.

3. **Fase de cierre**: Fase final del proyecto. Una vez un *deal* ha alcanzado esta fase, se excluye la oportunidad y no se continua sincronizando. Desde este momento se trata de manera independiente el proyecto en *Workday*, y el *deal* no se modifica más.

La fase a la que pertenece el *deal* depende de su estado. En la Figura 3.2 se puede ver a que fase pertenece cada estado.

Todo *deal* empieza con un estado perteneciente a la fase de descubrimiento, con el paso del tiempo puede ir pasando a la fase de progreso y finalmente todo *deal* termina en uno de los dos estados de la fase de cierre: *Closed Won* y *Closed Lost*.

Además de propiedades, los objetos pueden tener asociaciones. En el caso del deal puede tener una o más *companies* asociadas. La Interfaz *HubSpot-Workday*, dará por hecho que cada *deal* está asociado a lo sumo a una *company*.

Desde el portal de *HubSpot* es posible tanto crear nuevos *deals* como modificar cualquier propiedad a excepción del *Deal Id*.

Cuando seleccionamos la opción de creación de un *deal* nos aparece un formulario como el de la Figura 3.3.

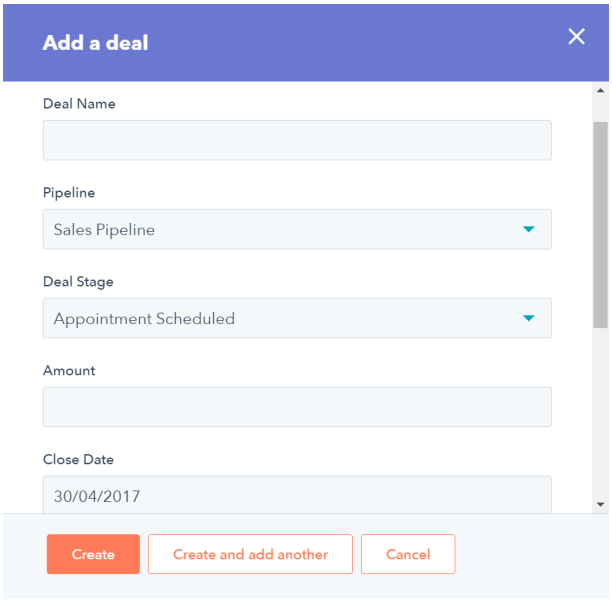


Figura 3.3: Creando un *deal* desde el portal de *HubSpot*

3.1.2. Descripción del objeto *Company*

Es un objeto de *HubSpot* que representa cualquier tipo de organización o empresa. Este objeto cuenta con multitud de propiedades, que describen información de la empresa, como puede ser información de contacto, localización...

Desde el portal de *HubSpot* es posible tanto crear nuevas *companies* como modificar cualquiera de las ya existentes. Podemos ver el formulario de creación de una *company* en la Figura 3.4.

Sin embargo la creación o modificación de *companies* no iniciarán procesos de sincronización. Solo se integrarán aquellas *companies* que estén asociadas a un *deal*. De esta forma evitamos que *companies* con las que no existe un proyecto estén sincronizadas.

3.1.3. Métodos de integración de *HubSpot*

HubSpot cuenta con varios métodos para realizar las integraciones, cada uno de ellos pensado para distintas situaciones.

Integración con *API keys*

Para utilizar este método simplemente hay que consultar la clave de la API de *HubSpot* (*HAPIKey*), accesible desde el portal de *HubSpot* y añadirla como parámetro en las peticiones que enviemos al servicio REST de *HubSpot*.

Este método es idóneo para realizar pruebas. Sin embargo, su falta de seguridad impide su uso comercial.

Además con este método solo es posible la comunicación en un sentido, desde la Interfaz *HubSpot-Workday* hacia *HubSpot*.

Integración con *OAuth*

Para poder usar este método de integración es necesario crearse una cuenta de desarrollador. Con la cuenta de desarrollador podemos crear una **aplicación de *HubSpot*** y acceder a su panel de control.

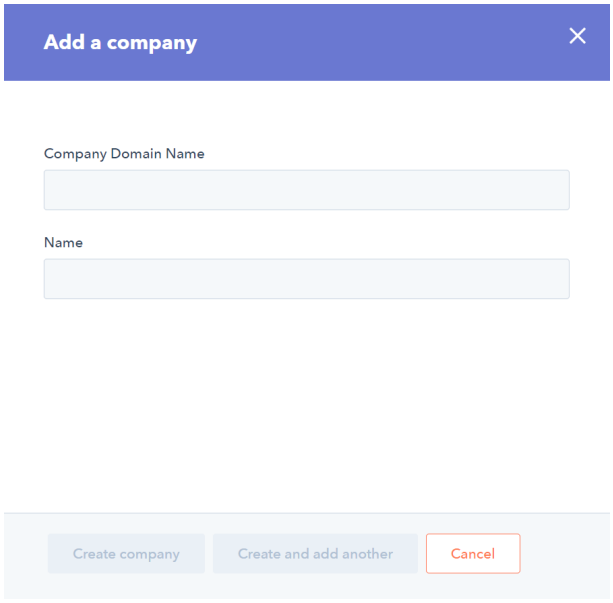


Figura 3.4: Creando una *company* desde el portal de *HubSpot*

Desde el panel de control podemos ver toda la información de la aplicación. La información que necesitaremos es el identificador de la aplicación, la clave `Client Id` y la clave `Client Secret`.

Otra funcionalidad del panel de control es la posibilidad de configurar las suscripciones *Webhook* de *HubSpot*, que permiten enviar de mensajes desde *HubSpot* a destinos externos cuando ciertos eventos ocurren.

También podemos hacer un seguimiento del funcionamiento de la aplicación desde el panel de control.

Lo primero que necesitamos es instalar la aplicación en el portal de *HubSpot*.

Para instalar la aplicación en el portal de *HubSpot* hay que seguir un proceso de autorización llamado *OAuth 2.0*[1]. Este proceso se ha automatizado e integrado en la Interfaz *HubSpot-Workday* creada. Los pasos que hay que seguir son los siguientes [hsa]:

1. Dirigirse a la *url* de autorización[2]. Iniciar sesión en el portal de *HubSpot* y así autorizar la instalación de la aplicación. Cabe destacar que solo se puede acceder a dicha *url* y por tanto instalar la aplicación si se cuenta con el `Client ID`. también se necesita un usuario del portal con permisos para poder instalar aplicaciones. Haciendo público el `Client ID` de la aplicación, esta puede ser instalada por cualquier persona en portales a los que tengan acceso. En nuestro caso este valor permanecerá en privado y solo nosotros haremos uso de la aplicación.
2. Tras dar acceso al portal se te redirige a una página en la que aparece como parámetro de la *url* un código.
3. Mediante este código, el `Client ID` y el `Client secret` podemos obtener un *token* de sesión con una llamada a la API. Tras hacer la petición a la API de *HubSpot*, nos devolverá un mensaje JSON con los siguientes valores.

[1] OAuth 2.0 es un protocolo estándar de la industria para la autorización. OAuth 2.0 proporciona un flujo de autorización específico para aplicaciones web, de escritorio, dispositivos móviles...

[2] `https://app.hubspot.com/oauth/authorize?client_id=<client_id>&scope=contacts&redirect_uri=https://www.hubspot.com/` sustituyendo `<client_id>` por el su valor

- `access_token` clave para realizar las peticiones a la API de *HubSpot*.
- `refresh_token` clave para conseguir nuevo par (`access_token`, `refresh_token`) cuando el `access_token` ha expirado.
- `expires_in` tiempo en milisegundos de validez del `access_token`, por defecto son seis horas.

4. Cuando el `access_token` haya expirado podremos llamar a otro *end point* especificando el `refresh_token` y recibiremos un mensaje de vuelta con otro nuevo `access_token` y `refresh_token`.

Usando un *token* válido podremos realizar peticiones a los distintos *endpoint* REST de *HubSpot*, es decir, la comunicación de la Interfaz *HubSpot-Workday* a *HubSpot*.

Con este método se garantiza que los *token* van variando como mínimo cada seis horas a diferencia del método de integración con *API keys* en el que la clave es fija y solo varía si lo hacemos manualmente.

La aplicación instalada sirve como paso intermedio para las comunicaciones entre *HubSpot* y la Interfaz *HubSpot-Workday*.

La comunicación en el otro sentido, de *HubSpot* hacia la Interfaz *HubSpot-Workday* se hace gracias a la configuración de los *webhook* de *HubSpot*.

Desde el portal podemos configurar las suscripciones a los *webhook*. Es necesario especificar la dirección (*host* y puerto) donde deseamos que los mensajes sean enviados ante los diferentes eventos.

En el caso de nuestra aplicación está suscrita a los siguientes eventos:

- **deal.creation**: Cuando un nuevo *deal* es creado en *HubSpot* se envía un mensaje.
- **deal.propertyChanged**
  - `dealstage`: Cuando en un *deal* de *HubSpot* se modifica el estado.
  - `practice`: Cuando en un *deal* de *HubSpot* se modifica la práctica.
  - `hubspot_owner_id`: Cuando en un *deal* de *HubSpot* se modifica su propietario.
  - `closedate`: Cuando en un *deal* de *HubSpot* se modifica la fecha de cierre.
  - `description`: Cuando en un *deal* de *HubSpot* se modifica la descripción.
  - `dealname`: Cuando en un *deal* de *HubSpot* se modifica el nombre.
  - `transaction_currency`: Cuando en un *deal* de *HubSpot* se modifica la moneda.
  - `legal_entity`: Cuando en un *deal* de *HubSpot* se modifica la entidad.

La versión que se debe usar de *OAuth* es *OAuth2.0* ya que *HubSpot* en un futuro no dará soporte a las integraciones con *OAuth1.0*.

3.2. *Workday*

En esta sección vamos a describir la información que necesitamos saber sobre *Workday*.

Workday es un ERP que internamente se encuentra organizado en objetos.

Los objetos que van a ser de interés para realizar la integración son: *Project*, *Customer* y *Hierarchy*.

Además cada uno de estos objetos cuenta con una serie de campos, que pueden incluso hacer referencia a otros objetos. Un *project* tiene un *customer* puede tener asociado y también puede tener dos *hierarchies* asociadas, una opcional y una principal.

Para realizar las integraciones, *Workday* proporciona un catálogo de servicios web basados en SOAP (*Workday Web Services* [wws]). Mediante estos servicios podemos hacer que la Interfaz *HubSpot-Workday* interactúe con *Workday*.

Cada servicio cuenta con un conjunto de operaciones que pueden ser llamadas mediante su *end point*.

Lo siguiente que necesitamos saber es el formato de los mensajes que se deben enviar a estos *end point*. Para ello contamos con el fichero WSDL de cada servicio que especifica el formato de los mensajes SOAP que se deben enviar para realizar cada operación.

El envío de los mensajes se realiza por HTTP.

Una herramienta que se ha utilizado para probar los servicio es *SOAP UI*.

Además es necesario proporcionar credenciales en el mensaje para que *Workday* realice las operaciones deseadas.

Ante estas solicitudes *Workday* siempre responde indicando si la operación se ha realizado correctamente o el error que ha sucedido.

3.2.1. Descripción del objeto *Project* en *Workday*

Es un objeto de *Workday* y existe una instancia por cada proyecto. Dentro de nuestra integración, el objeto *Project* de *Workday* se corresponde con el objeto *Deal* de *HubSpot*. De forma que por lo general las acciones sobre el objeto *Deal* en *HubSpot* darán lugar a otros eventos sobre el objeto *Project* de *Workday*.

Un objeto *Project* puede ser creado de manera manual, pero en nuestra integración, esto se realizará mediante los servicios web. Para crear o modificar un proyecto ya existente hay que enviar un mensaje SOAP por HTTP. Para ello hay que usar la operación `Submit_Project` incluida en el servicio `Resource_Management`.

El objeto *Project* cuenta con multitud de campos, son de especial importancia los siguientes:

- ***Project Name***: Nombre del proyecto, está sincronizado con el nombre del *deal* correspondiente.
- ***Start Date***: Fecha de inicio, está sincronizado con con la propiedad `close_date` del *deal*.
- ***Status***: Estado del proyecto, existe una correspondencia con la propiedad `deal_stage` del deal en *HubSpot*.
- ***Description***: Descripción del proyecto. Este campo se encuentra sincronizado con la propiedad `description` del *deal*.

Otros campos que hacen referencia a objetos son estos:

- ***Project Hierarchy***: La Jerarquía principal del proyecto. Este campo está en directa relación con la propiedad personalizadas `practice` de *HubSpot*.
- ***OptionalProject Hierarchy***: La jerarquía opcional del proyecto. Este campo se encuentra directamente relacionado con la propiedad `hubspot_owner_id` del correspondiente *deal*.
- ***Customer***: El cliente del proyecto. Existe una correspondencia entre el cliente y la asociación `associated_company` del *deal* en *HubSpot*.

3.2.2. Descripción del objeto *Customer* en *Workday*

Se trata de un objeto de *Workday* que representa a los clientes de la empresa. este objeto de *Workday* se encuentra relacionado con el objeto *company* de *HubSpot*. Gracias a la Interfaz *HubSpot-Workday*, cuando se realice la integración de un *deal*, también se integrará la *company* asociada.

Existen múltiples métodos para crear o modificar un *Customer* en *Workday*, de forma manual, hojas de calculo para carga de datos ...

En nuestro caso la creación y modificación de un cliente se hace desde la Interfaz *HubSpot-Workday*, mediante la operación `Put_Customer` del servicio web `Revenue_Management`.

3.2.3. Descripción del objeto *Hierarchy* en *Workday*

El objeto *Hierarchy* de *Workday* se encarga de representar jerarquías. Es decir, representa información que se encuentra estructurada en forma de árbol.

En nuestro caso tenemos una jerarquía principal que organiza los proyectos según a que práctica pertenecen. (En el caso de nuestra empresa las distintas prácticas son: *PeopleSoft*, *SAP*, *Cloud* ...).

También existen jerarquías opcionales para organizar los proyectos según su *Customer Manager* y su *Project Manager*.

3.2.4. Servicios web de *Workday*

Para hacer uso de los servicios web de *Workday*, primero debemos de contar con una cuenta del *tenant* (o entorno) con permisos para el uso de los servicios web que se vayan a usar.

Una vez tengamos dicho usuario, podremos realizar las operaciones.

Los pasos para realizar una operación concreta son los siguientes:

1. Determinar en que servicio se encuentra la operación deseada.
2. Conseguir el archivo WSDL. En este archivo podremos encontrar el formato de mensaje correcto para cada una de las operaciones disponibles en dicho servicio.
3. Modificar la plantilla de mensaje con los parámetros deseados. Para comprobar que el mensaje realiza la operación deseada correctamente podemos usar la aplicación *SoapUI*[1]. *SoapUI* permite abrir esquemas WSDL, modificar directamente la plantilla del mensaje para cualquier operación del servicio y enviar el mensaje.
4. Enviar el mensaje SOAP por HTTP con el método *POST*.

[1] `https://www.soapui.org/`

5. comprobar que no ha habido errores con el mensaje de respuesta proporcionado por *Workday*.

Para el tratamiento de los mensajes SOAP se ha realizado del siguiente modo. Se ha usado la librería de *python lxml*[1].

1. Se genera el archivo *xml* con ayuda de la librería *lxml*. Este *xml* cuanta con los parametros necesarios para la llamada al servicio web de *Workday*.
2. Con el archivo XSLT correspondiente se transforma el XML en el mensaje SOAP.

Cuando *Workday* responde con un mensaje SOAP, la Interfaz *HubSpot-Workday* recupera los valores del XML con ayuda de la librería *lxml*.

3.3. Interfaz *HubSpot-Workday*

Para el desarrollo de la Interfaz *HubSpot-Workday* he usado el lenguaje de programación **python**, con el *IDE* **PyCharm**. La librería principal, bajo la que se sustenta el servidor es ***web.py***. Alternativas a esta librería pueden ser los *frameworks Flask* o *Django*. *Flask* es un *microframework*, sería una buena alternativa si hubiese que elegir algo distinto a ***web.py***.

Django es idónea para proyectos mucho más grandes. Requiere de un proceso más avanzado antes de poder empezar a desarrollar la aplicación.

La ventaja de *web.py* es lo ligera que es y la facilidad de ponerla en marcha.

La Interfaz *HubSpot-Workday* se encuentra ejecutándose de manera continua en un servidor de *amazon* y está escuchando en un puerto los distintos mensajes que puedan llegar de *HubSpot*. En el panel de control de la aplicación de *HubSpot*, explicado en la sección 3.1.3, debemos especificar el *host* y puerto del servidor que está ejecutando la Interfaz *HubSpot-Workday*.

En esta sección vamos a explicar detalladamente el funcionamiento de la Interfaz *HubSpot-Workday*.

3.3.1. Estructura de la Interfaz *HubSpot-Workday*

Vamos a describir como se encuentra estructurada nuestra Interfaz *HubSpot-Workday*. Para ello podemos ver la Figura 3.5.
El proyecto de *python* se encuentra organizado en diferentes archivos subdirectorios.

- ***service.py***: Se encuentra en el directorio raíz y se encarga de ejecutar la aplicación y en este modulo está definida la función que se ejecuta al recibir un mensaje *POST*. Desde este módulo se hacen llamadas a distintas funciones de *handler.py*.
- ***handler.py***: También se encuentra en el directorio raíz y en este módulo se encuentra la lógica de programa para cada evento recibido.
- ***hubspot***: En este directorio se encuentran todos los módulos de *python* que implementan funcionalidades para interactuar con *HubSpot*.
  - ***deal.py***: En este módulo se encuentra la clase *Deal*, encargada de representar en la Interfaz *HubSpot-Workday* el objeto *deal* de *HubSpot*.
  - ***company.py***: En este módulo se encuentra la clase *Company*, que se encarga de representar al objeto *company* de *HubSpot*.

[1] `http://lxml.de/`

  - ***authorization.py***: En este módulo se encuentra la clase *Authorization* encargada de facilitar el flujo de aprobación *OAuth 2.0* de *HubSpot* y la renovación de los *token* de sesión.

- **workday**: En este directorio están los módulos encargados de la interacción con *Workday*.

  - ***project.py***: En este módulo se encuentra la clase *Project* que representa en la Interfaz *HubSpot-Workday* el objeto *Project* de *Workday*.
  - ***custommer.py***: En este módulo se encuentra la clase *Customer* que representa en la Interfaz *HubSpot-Workday* el objeto *Customer* de *Workday*.
  - ***hierarchy.py***: En este módulo se encuentra la clase *Hierarchy* que representa en la Interfaz *HubSpot-Workday* el objeto *Hierarchy* de *Workday*.

- **modules**: En este directorio se encuentran agrupados módulos con funcionalidades diversas.

  - ***configuration.py***: En este módulo es donde inicializamos los *Parser* de cada uno de los archivos de configuración.
  - ***database.py***: Módulo para inicializar de las tablas de la base de datos y para la realización de operaciones en estas tablas.
  - ***log.py***: Módulo encargado de implementar la salida a ficheros registro.
  - ***mail.py***: Este módulo se encarga de la funcionalidad necesaria para enviar *emails*.

- **cfg**: En este directorio se encuentran todos los archivos de configuración. Por lo general permanecen invariantes. A excepción de aquellos que almacenan *tokens* de sesión que van cambiando.

- **xslt**: En este directorio se encuentran todas las plantillas de transformación necesarias para enviar los mensajes a *Workday*. Partiendo de un archivo XML, podemos transformarlo con estas plantillas y generar un mensaje SOAP para su posterior envío a *Workday*.

3.3.2. Almacenamiento persistente

En la Interfaz *HubSpot-Workday* existen dos tipos de almacenamiento persistente: Estático y dinámico.

- **Estático**: Se trata de los archivos de configuración ubicados en el directorio *cfg* que se pueden ver en la Figura 3.5. En estos archivos se guarda información como usuarios, contraseñas, *token* de sesión, direcciones *url*... De esta forma se evita que estos datos se encuentren escritos directamente en el código como constantes. Nos facilita su modificación e incluso la posibilidad de excluirlos fácilmente cuando se añada el proyecto a repositorios públicos. Está información es fija (a excepción de los *token* de sesión). Solo se cambiará cuando sea necesaria otra configuración.

  Para su implementación he usado el módulo *ConfigParser*[1] parte de una librería estándar. *ConfigParser* facilita la lectura de archivos de configuración. A estos archivos se les suele poner la extensión *ini*. Estos archivos de configuración se organizan en secciones formadas por listas de parejas clave-valor. Una ventaja al usar *ConfigParser* es que a la hora de definir una pareja clave-valor se puede referenciar un valor definido en la sección *DEFAULT*.

  Otras alternativas para los archivos de configuración son *yaml*, *xml*, *json*...

[1] `https://docs.python.org/2/library/configparser.html`

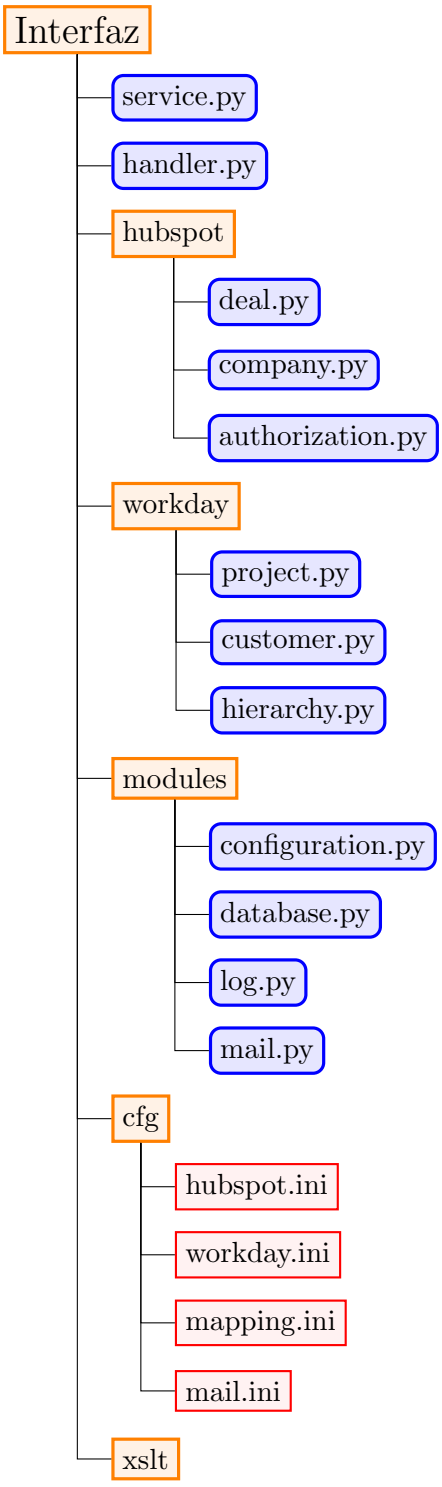


Figura 3.5: Estructura de la interfaz

  - ***hubspot.ini***: En este archivo se guarda la información referente a los *tokens* de sesión, necesarios para comunicarse con el portal de *HubSpot*. Este archivo cambia cada vez que se actualiza un *token*.
  - ***workday.ini***: En este archivo de configuración se guardan las credenciales de acceso a *Workday*, así como distintas direcciones url para evitar su repetición en el código.
  - ***mapping.ini***: En este archivo de configuración se encuentran todas las correspondencias entre *HubSpot* y *Workday*.
  - ***mail.ini***: En este archivo de configuración se encuentran las credenciales necesarias para el acceso a la cuenta de correo.

- **Dinámico**: Se trata de la base de datos local *SQLite3*[1] para *python*. En esta base de datos se almacenan las distintas correspondencias entre *HubSpot* y *Workday*.

  Existen tres tablas en la base de datos: `deals_excluded`, `deal_project` y `company_customer`.

  - `deals_excluded`: En esta tabla se almacena los identificadores de los *deals* de *HubSpot* que se encuentran excluidos para la integración. Todos los *deals* que se encuentren en la fase de cierre estarán excluidos.
  - `deal_project`: En esta tabla se encuentra la correspondencias entre identificadores de *deals* de *HubSpot* y los identificadores de los *projects* en *Workday*, para todos los *deals* que han sido integrados.
  - `company_customer`: En esta tabla se encuentra la correspondencia entre las *company* de *HubSpot* y los *customers* de *Workday* para todos los *companies* que hayan sido integrados.

deals_excluded
deal id

deals_excluded	
deal id	project id

company_customer	
company id	customer id

Tabla 3.1: Tablas en la base de datos local

3.3.3. Seguridad

La Interfaz *HubSpot-Workday* se encuentra escuchando los mensajes que le son enviados. Pero ¿Cómo podemos garantizar que el mensaje que recibimos proviene realmente de *HubSpot*? Para ello existe un campo en la cabecera del mensaje recibido, que es `HUBSPOT_SIGNATURE`. Este campo es el formado tras aplicar la función *hash* ***sha256*** a la concatenación del `Client Secret` de nuestra aplicación de *HubSpot* con el cuerpo del mensaje HTTP recibido.

Basta hacer esta comprobación para garantizar la procedencia del mensaje, ya que el `Client Secret` solo es conocido por nosotros y por *HubSpot*.

Además cabe destacar que cualquier persona que interceptase el mensaje, no podría deducir el `Client Secret`, gracias a las propiedades de las funciones *hash*.

Como podemos comprobar en la Figura 3.6, aquellos mensajes que no cumplen los requisitos, se ignoran. Esto protege a nuestra Interfaz *HubSpot-Workday* de aquellos mensajes no autorizados.

3.3.4. Transformación del *deal* de *HubSpot* en el *project* de *Workday*

Como se ha comentado anteriormente si un *deal* u oportunidad de negocio en *HubSpot* alcanza la fase de progreso, entonces se crea el correspondiente *project* en *Workday*.

[1] `https://docs.python.org/2/library/sqlite3.html`

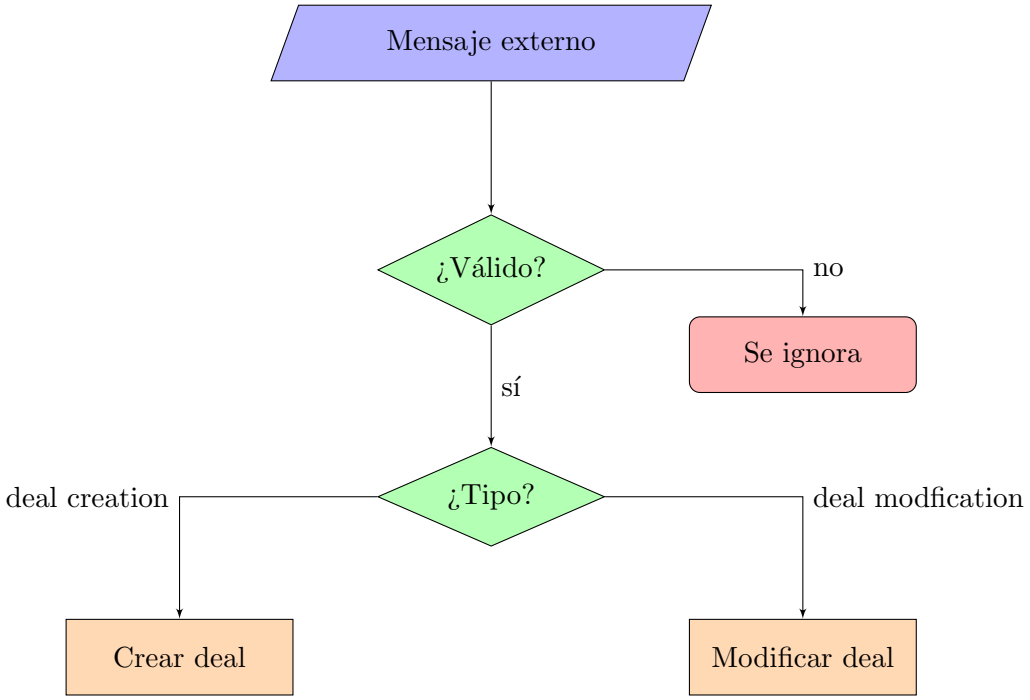


Figura 3.6: Validación de los mensajes recibidos por la Interfaz

Para poder hacer esto es necesario saber como cierto *deal* se transforma en *project*. En la Tabla 3.2 podemos ver la traducción de las propiedades del *deal* de *HubSpot* en los campos del *project* de *Workday*.

HubSpot	***Workday***
Deal Name	*Project Name*
Close Date	*Start Date*
Deal Stage	*Status*
Deal Description	*Description*
Practice	*Project Hierarchy*
Hubspot Owner	*Optional Project Hierarchy*
company	*Customer*

Tabla 3.2: Transformación de las propiedades del *deal* en los campos del *project*.

- El nombre del *deal* en *HubSpot* se corresponde con el nombre del *project* en *Workday*, como es lógico.

- La fecha en la que se ha cerrado el acuerdo de la oportunidad de negocio (*Close Date*) se corresponde con la fecha en la que comienza a gestionarse el *project* en *Workday*.

- El *Deal Stage* de hubspot se traduce en el *Status* del *project* en *Workday*. La correspondencia se puede ver en la Tabla 3.3.

Código 3.1: Ejemplo de mensaje recibido por la Interfaz *HubSpot-Workday*

```
[{"objectId":92604011,
  "changeFlag":"NEW",
  "changeSource":"",
  "eventId":2763628519,
  "subscriptionId":1794,
  "portalId":1234,
  "appId":5678,
  "occurredAt":1485865855549,
  "subscriptionType":"deal.creation",
  "attemptNumber":0}]
```

Deal Stage	***Project Status***
Opportunity Identified	*Opportunity Identified*
Preparing Proporsal	*Preparing Proporsal*
Proporsal Sent	*Proporsal Sent*
Decision Maker Bought In	*Contract Sent*
Contract Sent	*Contract Sent*
Closed Won	*Active*
Closed Lost	*Closed Lost*

Tabla 3.3: Transformación del *Deal Stage* en el *Project Status*.

- La descripción del *deal* se corresponde con la descripción del *project*.
- Según la práctica a la que pertenezca el *deal*, el *project* es asignado a la correspondiente jerarquía principal.
- Dependiendo del propietario del *deal*, el *project* es asignado a la correspondiente jerarquía opcional.
- Por último la *company* asociada a cierto *deal* en *HubSpot* se corresponde con el campo *customer* del objeto *project* en *Workday*.

3.3.5. Flujo del programa

Como se ha mencionado anteriormente, nuestra aplicación de *HubSpot* está suscrita a diferentes eventos de los *deals* de *HubSpot*. Cuando uno de estos eventos se produce, la aplicación de *HubSpot* envía un mensaje HTTP (*POST*) a la Interfaz *HubSpot-Workday*. Este mensaje es un JSON con información sobre el evento ocurrido. Podemos ver un mensaje de ejemplo en el Código 3.1.

Para probar el correcto funcionamiento de la interfaz sin realizar cambios en *HubSpot*, se ha usado la herramienta *cURL*[1]. Con *cURL* podemos simular las peticiones HTTP enviadas desde *HubSpot*.

Una vez recibido el mensaje se comprueba su validez verificando su procedencia como hemos visto en la Sección 3.3.3. Aquellos mensajes inválidos se ignoran como se muestra en la Figura 3.6.

Tras verificar que un mensaje es válido, se comprueba si dicho *deal* se encuentra en la tabla de excluidos de base de datos local (`deals_excluded`). Esta comprobación sirve como precaución

[1] `https://curl.haxx.se/`

para evitar la sincronización de proyectos no deseados en *Workday*.

Después dependiendo del valor del campo *subscriptionType* comienza la ejecución del proceso de creación de un *deal* o de modificación.

- **Creación de un *deal***: Si el valor del atributo *subscriptionType* es *deal.creation*, éste nos indica que un *deal* ha sido creado. En el mensaje JSON también se especifica el identificador del *deal* creado (*objectId*) y otros datos de relevancia como el identificador del portal, identificador de la aplicación, momento en el que ocurrió el evento...

  Los pasos que se siguen para el procesamiento del mensaje son los siguientes:

  1. Se comprueba si el *deal* está excluido, si es así entonces la petición no se procesa.

  2. Se comprueba si el *deal* tiene una entrada en la tabla de correspondencias entre los *deals* y *projects* (`deals_project`). En la tabla `deals_project` se guarda la correspondencia entre los identificadores de los *deals* de *HubSpot* y los identificadores de los *projects* de *Workday* que están integrados. En caso de que el *deal* esté en la tabla de correspondencias, quiere decir que ese *deal* se ha creado previamente y por tanto la petición se ignora y para el proceso.

     Si por el contrario no se encuentra en la tabla de correspondencias entonces se continua con el siguiente paso.

  3. Se realiza una petición a la API de *HubSpot* para obtener el resto de información del *deal* creado.

  4. Con todos los datos del *deal* que se ha creado se comprueba si este se encuentra en la fase de progreso, en caso contrario dicho *deal* no se integra y se para el proceso.

  5. Si el *deal* se encuentra en la fase de progreso, entonces se comprueba si el *company* asociado al *deal* existe en la tabla de correspondencias entre los *companies* y los *customers* (`company_customer`). En caso de que no exista, se crea el correspondiente *customer* en *Workday*, y se actualiza la tabla `company_customer` con la nueva entrada.

  6. Se crea el *project* asociándolo con el *customer* previamente creado o existente y se actualiza la tabla `deals_project` con la nueva correspondencia.

  Podemos ver un diagrama del proceso en la Figura 3.7

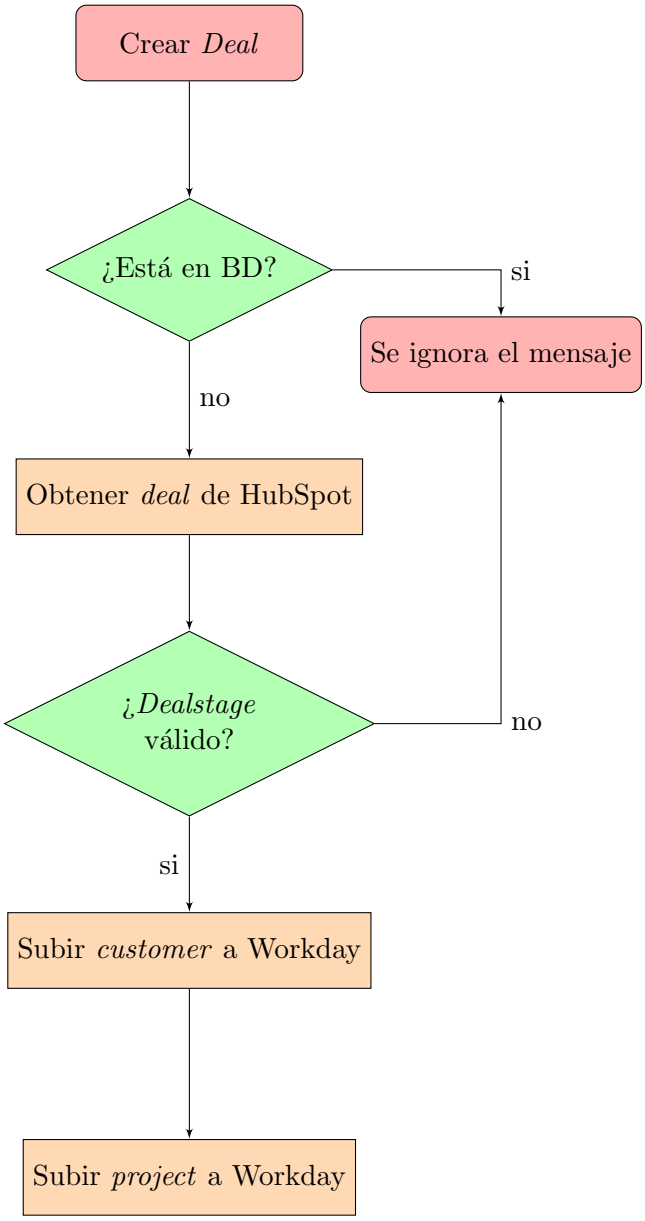


Figura 3.7: Proceso Crear Deal

- **Modificación de un *deal***:

  Si atributo *subscriptionType* en el mensaje JSON recibido tiene el valor *deal.propertyChange*, quiere decir que se ha producido el cambio de alguna propiedad en un *deal* de *HubSpot*. En estos casos el mensaje JSON contiene otro atributo *propertyName* que indica qué propiedad ha cambiado.

  Los pasos que se siguen ante este tipo de mensajes son los siguientes:

  1. Se comprueba si dicho *deal* se encuentra en la tabla `deals_excluded` de base de datos local. En tal caso, se para el proceso y no se sincronizan las modificaciones en *Workday*.
  2. Se comprueba si el *deal* existe en la tabla de correspondencias `deal_project`. De no ser así se procede a la creación del *deal* con el proceso detallado anteriormente.
  3. Se obtiene el *deal* de *HubSpot* con todas su propiedades mediante una llamada a la API de *HubSpot*. También se obtiene el objeto *project* de *Workday*. Dependiendo de

la propiedad del *deal* que haya cambiado se actualizan localmente los correspondientes campos del *project*. Por último se actualiza el objeto *project* correspondiente de *Workday* enviando un mensaje SOAP.

Podemos ver un diagrma del proceso en la Figura 3.8

Cuando un *deal* es modificado y pasa a un estado perteneciente a la fase de cierre, el *deal* se añade a la tabla de excluidos (`deals_excluded`).

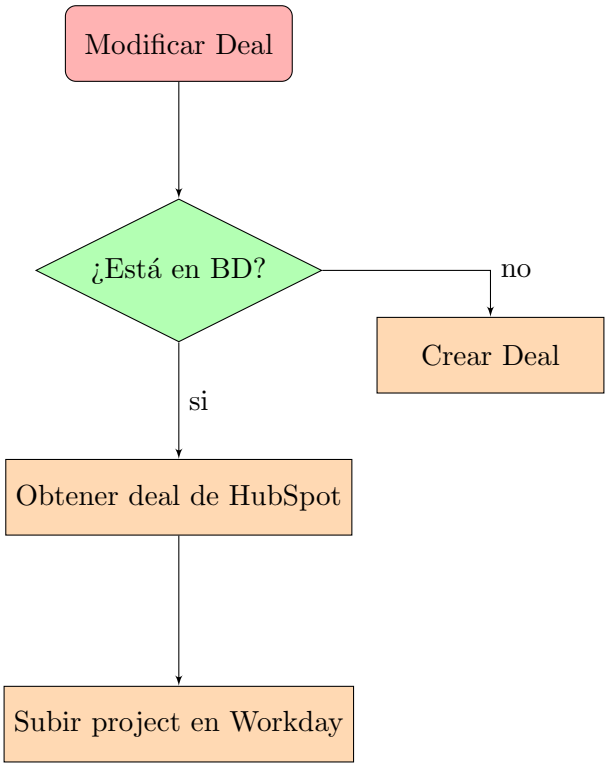


Figura 3.8: Proceso Modificar Deal

De esta forma conseguimos gracias a la Interfaz *HubSpot-Workday* que los *deals* de *HubSpot* se encuentren sincronizados con los *projects* de *Workday*. Este proceso ocurre de forma totalmente transparente al usuario de *HubSpot*

Capítulo 4

Predicción de abandono en el trabajo

La empresa me solicitó realizar un prototipo que, partiendo de los datos de una plantilla de trabajadores, fuese capaz de predecir futuros casos de abandono en el trabajo. Tras su desarrollo se verían las posibilidades de integrarlo con *Workday*.

En este capítulo explicaremos en detalle los datos que vamos a utilizar. Estudiaremos estos datos y construiremos un modelo capaz de predecir cuando un trabajador va a abandonar el trabajo.

El estudio de los datos se ha hecho en *python*. La librería que nos permite usar distintas técnicas de *Machine Learning* es *scikit-learn*[1].

También se ha usado las librerías *pandas*[2] para la transformación y análisis de datos y la librería *matplotlib*[3] para la visualización de los datos.

4.1. Datos y problema a resolver

Se ha realizado un estudio usando los datos con los que cuenta el proyecto de *IBM*, *Watson Analytics*[4]. Se trata de una tabla con información sobre los empleados de una empresa. Estos datos pertenecen a *IBM* y son los que usan para las predicciones analíticas del robot de *Inteligencia artificial Watson*. *IBM* ha desarrollado una solución al problema que queremos resolver, pero este proyecto de *IBM* no es de código abierto.

Para el uso de los datos he descargado la hoja de cálculo y así poder trabajar localmente. Después la he convertido a CSV para poder manejar los datos de manera cómoda desde *python*.

Se trata de una única tabla, donde cada fila hace referencia a la información de un empleado y donde cada columna nos da información sobre una característica.

Hay un total de 1470 empleados (filas) con 35 características (columnas).

Algunas de las características con las que contamos son las siguientes:

- **Age**: *Número*. indicando la edad.

[1] `http://scikit-learn.org/stable/`

[2] `http://pandas.pydata.org/`

[3] `https://matplotlib.org/`

[4] Fuente de datos proyecto de *IBM* de *Watson Analytics* `https://www.ibm.com/communities/analytics/watson-analytics-blog/hr-employee-attrition/` [Internet; descargado 4-junio-2017]

- **Attrition**: *Texto.* indicando si ha abandonado el trabajo o no.
- **BusinessTravel**: *Texto.* indicando la frecuencia con la que viaja.
- **DailyRate**: *Número.* indicando la tasa diaria del trabajador.
- **Department**: *Texto.* con el departamento al que pertenece el empleado.
- **DistanceFromHome**: *Número.* con la distancia de casa al trabajo.
- **Education**: *Número.* indicando el nivel de educación
- **EducationField**: *Texto.* con la rama de educación del trabajador.
- **EmployeeCount**: *Número.* que siempre es uno.
- **EmployeeNumber**: *Número.* que etiqueta con un identificador a cada empleado.
- **EnvironmentSatisfaction**: *Número.* que indica el nivel de satisfacción del trabajador.
- **Gender**: *Texto.* especificando el género
- **HourlyRate**: *Número.* indicando la tasa por hora del empleado.
- **JobInvolvement**: *Número.* indicando el grado de involucración en el trabajo.
- **JobLevel**: *Número.* especificando el nivel en el trabajo.
- **JobRole**: *Texto.* indicando el puesto de trabajo del empleado.
- **JobSatisfaction**: *Número.* especificando el nivel de satisfacción
- **MaritalStatus**: *Texto.* indicando el estado civil.
- **MonthlyIncome**: *Número.* Salario mensual.
- **NumCompaniesWorked**: *Número.* Cantidad de empresas en las que ha trabajado anteriormente.
- **Over18**: *Texto.* Si el trabajador es mayor de edad, todas las filas tienen el valor *Y*.
- **OverTime**: *Texto.* Si el trabajador realiza horas extra.
- **PercentSalaryHike**: *Número.* Porcentaje de aumento de salario.
- **PerformanceRating**: *Número.* Desempeño del trabajador.
- **RelationshipSatisfaction**: *Número.* Nivel de satisfacción con las relaciones en el trabajo.
- **TotalWorkingYears**: *Número.* Años totales trabajados.
- **TrainingTimesLastYear**: *Número.* Cantidad de veces que has recibido formación en el último año.
- **WorkLifeBalance**: *Número.* Conciliación de la vida laboral y familiar.
- **YearsAtCompany**: *Número.* Años en la compañía.
- **YearsInCurrentRole**: *Número.* Años en el puesto actual.
- **YearsSinceLastPromotion**: *Número.* Años desde el último ascenso.
- **YearsWithCurrManager**: *Número.* Años con el actual jefe.

En este estudio vamos a realizar un aprendizaje supervisado, ya que contamos con una columna (*Attrition*) que nos indica si el empleado en cuestión ha dejado o no el trabajo. Es decir, nos podemos basar en esa columna para decidir como de bueno es el modelo que estamos construyendo, y garantizar un buen comportamiento ante muestras fuera de nuestra población inicial.

Todos los pasos que he ido realizando están accesibles en un *Jupyter Notebook*[1]: Vamos a ir explicando en detalle cada uno de los pasos realizados.

Primero tenemos que deshacernos de aquellas características que no nos aporten información

- **Over18** que siempre tiene el valor *Y*.
- **EmployeeCount** que siempre tiene el valor 1.
- **EmployeeNumber** que no aporta información del empleado al ser solo un identificador.
- **StandardHours** que siempre tiene un valor de 80.

Después debemos transformar los datos para que tengan un formato que *scikit-learn* pueda entender. En definitiva que *numpy* pueda entender ya que *scikit-learn* esta construido sobre *numpy*. Así que tenemos que pasar las características de formato texto a numérico.

- **Attrition** contiene los valores *Yes* y *No* y los convertimos en 1 y 0 respectivamente.
- **BusinessTravel** puede tomar los valores *Travel_Rarely*, *Travel_Frequently* y *NonTravel*. Convertimos esta columna en dos columnas numéricas. Las columnas resultantes serian *BusinessTravel_Travel_Frequently BusinessTravel_Travel_Rarely* y podrían tomar los valores 0 y 1. En caso de que ambas estén a 0, representarían el valor *NonTravel*.
- **Department** puede tomar los valores *Sales*, *Research & Development*, *Human Resources*. Por tanto queda también transformada en dos columnas.
- **EducationField** tiene como posibles valores *Human Resources*, *Life Sciences*, *Marketing*, *Medical*, *Technical Degree* y *Other*. Y por ello esta columna se transforma en en 5 columnas numéricas.
- **Gender** contiene los valores *Male* y *Female* y los convertimos en 1 y 0 respectivamente en una nueva columna *Gender_Male*. En caso de tomar el valor 1 significa que el trabajador es un hombre y 0 una mujer.
- **JobRole** se transforma en 8 columnas numéricas, ya que tenemos 9 categorias posibles.
- **MaritalStatus** puede tener los valores *Single*, *Married*, *Divorced* y por tanto se crean dos columnas numéricas.
- **OverTime** contiene los valores *Yes* y *No* y los convertimos en 1 y 0 respectivamente.

4.2. Análisis de los datos

En esta sección vamos a analizar los datos y ver si podemos crear más variables que nos puedan servir de utilidad [Bow15]. Es importante saber que todas las columnas a excepción de **Attrition** podrán ser usadas para la construcción del modelo. La columna **Attrition** se usará para el entrenamiento y comprobación de la precisión del modelo.

El primer dato en el que nos queremos fijar lógicamente es el abandono. De los 1470 empleados un total de 237 han abandonado la empresa, mientras que 1233 han permanecido en ella. Esto

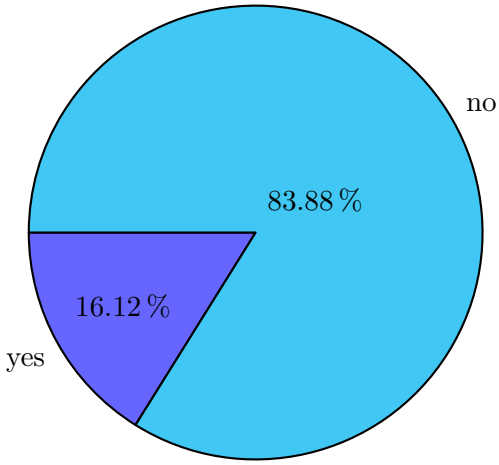
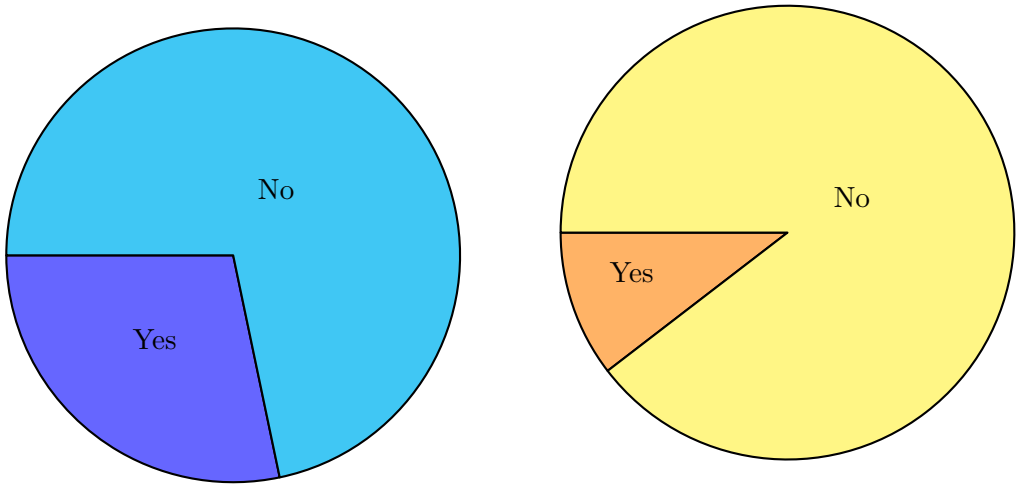


Figura 4.1: Tasa de abandono



(a) Con horas extra (b) Sin horas extra

Figura 4.2: Tasa de abandono según horas extra

supone que en nuestros datos hay un 16,12 % de abandono que podemos apreciar en la Figura 4.1.

La primera variable que suponemos tendrá una gran relación con con la predicción del abandono en el trabajo es **OverTime**, es decir, si el trabajador realiza horas extra. De los 1470 empleados 1054 no realizan horas extra y 416 sí, es decir, un 28,29 % realizan horas extra. De aquellos que realizan horas extra 127 han abandonado la empresa mientras que 289 no, lo que supone un abandono del 30,52 %. En cuanto a aquellos que no realizan horas extra 944 no han abandonado la empresa mientras que 110 sí. Nos encontramos ante una tasa de abandono del 10,43 %. En la figura 4.2 podemos ver reflejados todos estos datos.

Esta variable nos puede aportar información valiosa para realizar las predicciones.

Dado que a diferentes personas les puede afectar de forma distinta el hacer horas extra, he creado una nueva variable que puede ser de utilidad. Esta variable trata de reunir información de dos columnas diferentes: **OverTime** y **WorkLifeBalance**. Como *WorkLifeBalance* toma

[1] `http://jupyter.org/`

valores entre 1 y 4, donde valores más altos significan una mejor conciliación entre el trabajo y la vida familiar.

Con la siguiente fórmula podemos crear una nueva variable **OverTime_Balance**.

$$OverTime\_Balance = OverTime(5 - WorkLifeBalance\_Balance)$$

Cuando **OverTime_Balance** tome valores altos significa que realiza horas extra y tiene mala conciliación, mientras que valores bajos representan que no hace horas extra o si las hace tiene una buena conciliación.

La siguiente variable que vamos a observar es **Gender**. En los datos 882 empleados son hombres y 588 mujeres. Si comprobamos la tasa de abandono en hombres es de un 17 % y en mujeres de un 14,79 %. Esto nos indica que la variable **Gender** no aporta gran información.

Ahora vamos a estudiar la característica **YearsAtCompany**. La mediana de esta característica es 5 años. Podemos comprobar que si separamos los datos en aquellos mayores y menores al valor de la mediana, encontramos que la proporción de abandono es mayor en aquellos empleados que llevan menos de 5 años en la empresa. Dada esta situación voy a crear una nueva variable **AtCompany_Over** que tome el valor 1 si el empleado lleva más de 5 años en la empresa y 0 en caso contrario.

Relacionando la variable **YearsAtCompany** con la variable **YearsInCurrentRole** podemos crear otra variable **YearsAtCurrRole_Rate** que refleje la proporción de tiempo que lleva el empleado en el puesto actual en relación con el tiempo que lleva en la empresa. La siguiente fórmula lo consigue:

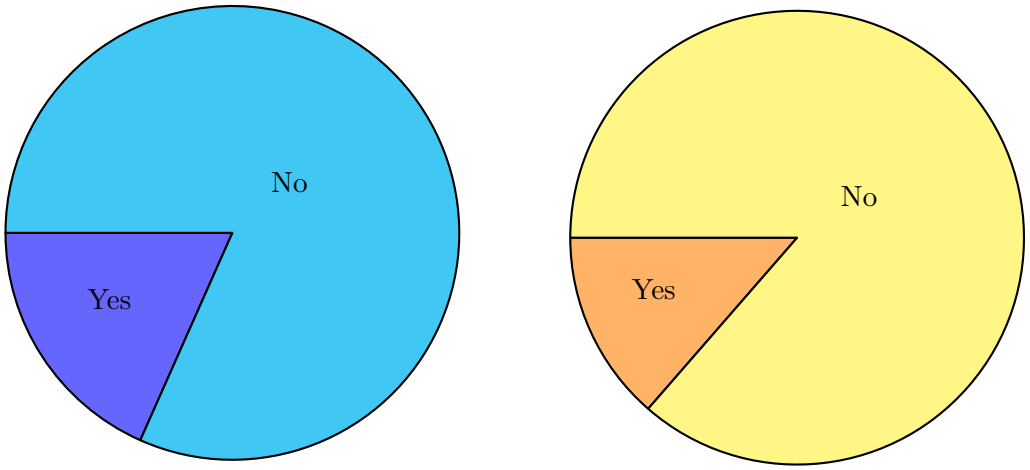
$$YearsAtCurrRole\_Rate = \frac{YearsInCurrentRole}{YearsAtCompany}$$

Vamos a fijarnos en la características **MonthlyIncome**. Si dividimos los empleados en aquellos que tienen un salario superior a la mediana de los que no, nos encontramos que la tasa de abandono en los que tienen un menor salario es mayor. Estos presentan un abandono del 21,76 % mientras que los de mayor salario tienen un 10,47 % de abandono. Puede ser interesante tener en cuenta esta característica para realizar las predicciones.

Dado que no es lo mismo que dos empleados que llevan distinto tiempo en la empresa estén cobrando el mismo salario. Vamos a crear una nueva característica que refleje la relación entre las características **MonthlyIncome** y **YearsAtCompany**.

Poniendo atención en la característica **DistanceFromHome** podemos apreciar que aquellos trabajadores que se encuentran a mayor distancia de sus casas son más propensos a abandonar el trabajo. Por ejemplo existe un abandono del 18,35 % en los trabajadores con una distancia superior a 7, mientras que un 13,6 % lo hace cuando la distancia es inferior a 7. En la figura 4.3 podemos visualizar estos datos. Como podemos observar no es una diferencia muy significativa, pero puede aportar información.

En la variable **Age**, podemos comprobar que la tasa de abandono es mayor en aquellos empleados más jóvenes frente a los de mayor edad. Por ejemplo existe un abandono del 21,94 % en los empleados menores a 36 años, mientras que en los mayores de 36 es de 10,39 %.



(a) Más de 7 (b) Menos de 7

Figura 4.3: Tasa de abandono según *DistanceFromHome*

Otro buen indicador de que un empleado tiene más posibilidades de marcharse de una empresa es su trayectoria previa. La variable **NumCompaniesWorked** nos da una idea de la trayectoria empresarial del trabajador. Sin embargo, por si sola parece que esta característica puede ser insuficiente. Por ello podemos combinarla con la variable **TotalWorkingYears** para conseguir una nueva variable **YearsPerCompany** que muestre la media de años trabajados por empresa.

Podemos lograr fácilmente este objetivo con la siguiente fórmula:

$$YearsPerCompany = \frac{TotalWorkingYears}{NumCompaniesWorked}$$

4.3. Construcción del modelo

Una parte importante a la hora de construir un modelo es elegir un buen estimador. Para ello nos vamos a ayudar de la Figura 4.4.

En nuestro caso contamos con más de 50 muestras, queremos predecir a que categoría pertenecen y contamos con los datos etiquetados gracias a la columna *Attrition*. Por tanto, nos encontramos en la burbuja *classification* de la Figura 4.4. Y vamos a elaborar un proceso para decidir cual de ellos nos conviene más.

Para construir un modelo es necesario determinar las características que se van a usar, con que estimador se va a realizar la predicción y bajo que parámetros.

4.3.1. Cálculo de la precisión de un modelo

Una vez elegido el modelo existen distintas formas de evaluar dicho modelo. Esto es posible en problemas supervisados, como es nuestro caso en el que conocemos el resultado para ciertos datos.

Con el modelo elegido es necesario entrenarlo con un conjunto de datos. Si lo entrenamos con todos los datos con los que contamos, solo podremos realizar predicciones con muestras que pertenecen al conjunto de entrenamiento. Lo cual no es muy revelador, pues es normal que el

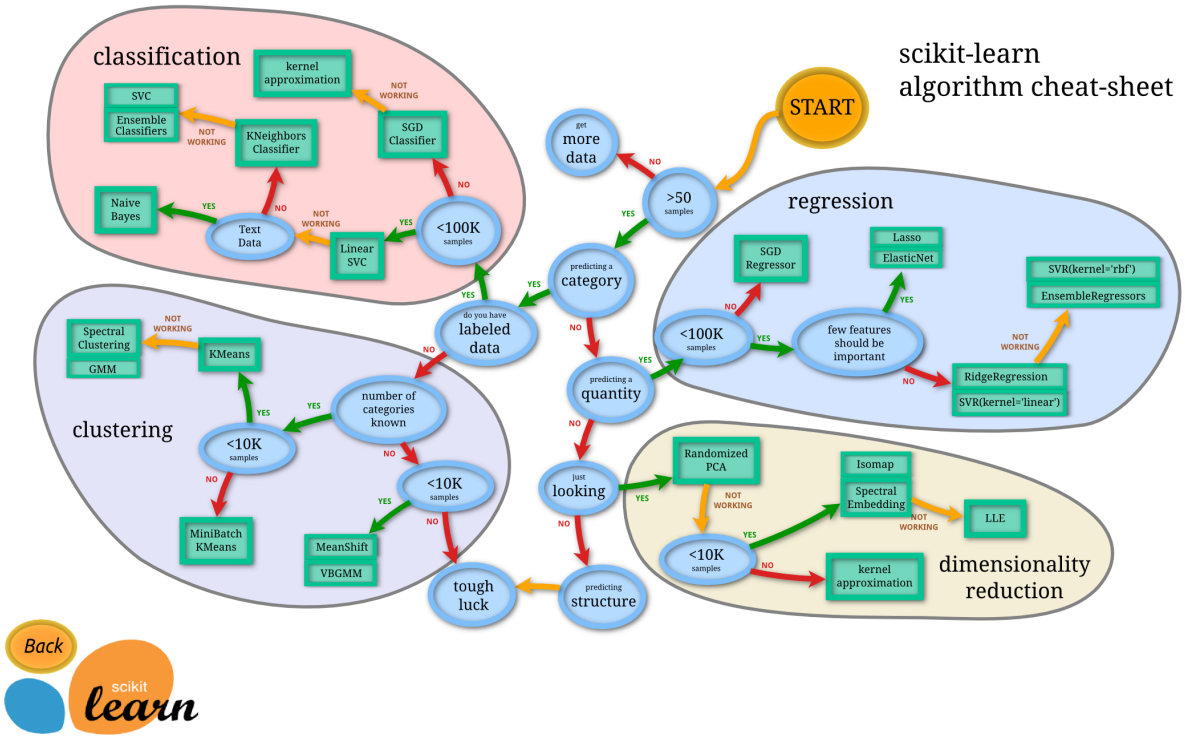


Figura 4.4: Esquema scikit-learn para la elección de un estimador

modelo se comporte bien ante muestras con las cuales se le ha entrenado.

Para evitar caer en este error existe el método de separación de los datos en dos conjuntos, uno de entrenamiento y otro de pruebas. Este método es conocido como **Split Training**. Podemos ver esta separación en la Figura 4.5.

Este método consiste en separar los datos en dos conjuntos: uno de entrenamiento (*train*), y otro de pruebas (*test*). Se entrena al modelo con el conjunto de entrenamiento y se realiza la predicción con los datos de prueba. Una vez realizada la predicción se compara con el resultado correcto utilizando alguna sistema de puntuación (accuracy, f1, roc_auc . . .).

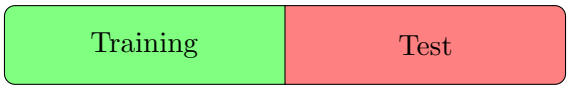


Figura 4.5: Split training

Sin embargo, puede que al hacer la separación en estos dos conjuntos se de un caso óptimo o por el contrario un caso fatal. Es por eso que se introduce otro nuevo método de evaluación, **Cross Validation** o Validación Cruzada.

Este método consiste en dividir los datos en una partición con k conjuntos. Y realizar k veces el método *Split Training*. En cada iteración, el conjunto *test* va siendo un elemento distinto de la partición y el conjunto *train* lo forman los $k - 1$ elementos restantes de la partición. Una vez finalizadas cada una de las k evaluaciones con *Split training* se realiza la media de las puntuaciones obtenidas, dando lugar a la puntuación con *Cross Validation*. Podemos ver un ejemplo en la Figura 4.6.

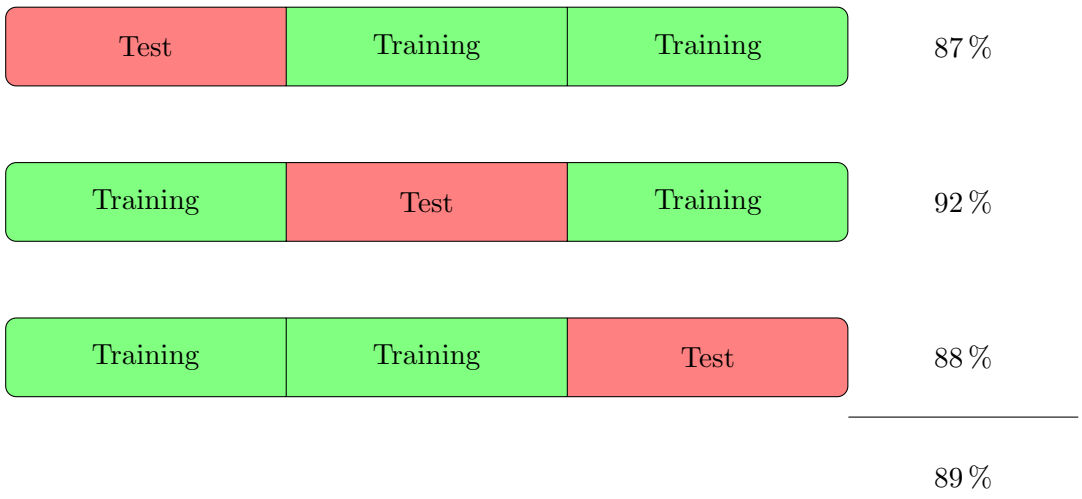


Figura 4.6: Evaluación con el método *Cross Validation* con $k = 3$

4.3.2. Selección de características

Esta fase consiste en la elección de aquellas características que aporten más información, para poder ser usadas en los distintos estimadores. Una pregunta común sería, ¿Por qué no usamos todas las características para realizar la predicción? La respuesta es sencilla, y es que en este caso, más no significa mejor. Usar características que no aportan información e introducen ruido en los datos repercute negativamente en la precisión del modelo.

La librería *scikit-learn* proporciona mecanismos para la selección automática de características. Estos son los que hemos usado:

- **Recursive Feature Elimination**. Dado un estimador que asigne pesos a las características *RFE* comienza con todas las características y recursivamente va considerando conjuntos más pequeños de características. Si usamos la clase `sklearn.feature_selection.RFECV` de *scikit-learn* podemos realizar este proceso de eliminación encontrando el número óptimo de características.
- **Feature selection**. Únicamente puede ser usado con estimadores que asigna importancia a las características tras ser entrenado. Podemos hacer uso de este método de selección de características gracias a la clase `sklearn.feature_selection.SelectFromModel` de *scikit-learn*.

4.3.3. Optimización de parámetros

Una vez elegido un estimador, es necesario configurar los parámetros para obtener el mejor resultado posible con dicho estimador. Para ello existe una clase en *scikit-learn* llamada `sklearn.model_selection.GridSearchCV` a la cual le podemos pasar como argumentos un diccionario de *python* con los valores que debe explorar para cada uno de los parámetros del estimador. Y lo que hace *gridSearchCV* es ejecutar *Cross Validation* para las distintas combinaciones de parámetros. Una vez finalizadas, podemos acceder tanto a la puntuación más alta como a los parámetros del estimador que han dado lugar a dicha puntuación.

4.3.4. Selección del estimador

Los diferente estimadores que he probado son:

- **Linear Support Vector Classification**. Perteneciente a la familia de estimadores *lineales*, cuyo objetivo es la construcción de un hiperplano en el espacio donde se encuentran

los datos. De manera que un punto se clasificará dependiendo de a que lado del hiperplano se encuentre.

- **K-vecinos más cercanos o K nearest neighbors Classifier**. Este estimador tiene como parámetros un valor k que indica el número de vecinos en los que debe fijarse y otro parámetro *weight_options* que puede tomar los valores *uniform* y *distance*. A la hora de clasificar un punto el estimador se fija en los k puntos de entrenamiento más cercanos, detecta que clase es la más común en esos k puntos, y esa clase es el resultado de la predicción. Cuando el parámetro *weight_options* tiene el valor *distance* se ponderan los valores de los k vecinos con el inverso de la distancia al punto que deseamos clasificar.
- **Regresión logística** se trata de un tipo de análisis de regresión usado para predecir el resultado de una variable categórica.

4.3.5. Resultado final

El resultado de todo el proceso ha dado lugar al modelo. Este modelo esta formado por un conjunto de características, un estimador y sus parámetros.

- **Características**: El conjunto de características final es el obtenido por el método *Recursive Feature Elimination*. Resultando ser 42 el número óptimo de características.

  Las características que han quedaron excluidas tras el método de *Recursive Feature Elimination* son: *Age*, *DailyRate*, *DistanceFromHome*, *Education*, *HourlyRate*, *MonthlyIncome*, *MonthlyRate*, *PercentSalaryHike* y *TotalWorkingYears*. Cabe destacar que si que se han utilizado *AgeOver*, *DistanceOver* e *IncomeOver*.

- **Estimador**: Tras comparar varios estimadores, se eligió el que daba lugar a un mejor resultado. Este estimador fue el de regresión logística.
- **Parámetros**: La búsqueda de los parámetros con *GridSearchCV* ha dado lugar a la siguiente configuración del Código 4.1.

Código 4.1: Declaración del estimador

```
LogisticRegression(penalty='l2',C=1 ,fit_intercept=True,
    intercept_scaling=0.1, tol=0.001, class_weight=None)
```

Esta configuración nos proporciona una precisión del 89,38 % estimada mediante el método de *Cross Validation*.

Ahora que hemos elegido el modelo que vamos a usar para realizar las predicciones, tenemos que entrenarlo con todos los datos con los que contamos (con los 1470 empleados). De este modo podremos obtener el mejor resultado al realizar predicciones sobre datos que estén fuera de nuestra población inicial.

Capítulo 5

Conclusiones

El desarrollo de la Interfaz *HubSpot-Workday* se ha realizado satisfactoriamente, sirviendo como proyecto interno de integración para BNB.

La salida a producción de *Workday* se realizó el 4 de abril de 2017. Con ello los sistemas de integración, incluyendo la Interfaz *HubSpot-Workday* se pusieron en funcionamiento.

Gracias al desarrollo de la Interfaz *HubSpot-Workday* he podido comprobar la versatilidad del lenguaje *python* a la hora de construir microservicios que se comunican con varias aplicaciones. También he podido entender que no es posible tener todo unificado en una misma aplicación. Por ello es necesario la integración de múltiples aplicaciones, para conseguir que los datos estén sincronizados.

He visto la variedad de metodologías utilizadas por distintas aplicaciones para facilitar las integraciones: servicios web, APIs...

También he comprobado las diferentes formas de permitir el acceso a las funcionalidades de integración: OAuth2.0, acceso por usuario y contraseña...

Ahora vamos a ir comprobando si se han cumplido los objetivos establecidos para la Interfaz *HubSpot-Workday* en el capítulo de introducción.

- Desarrollar un microservicio que realice la integración entre *HubSpot* y *Workday*.

  Este objetivo se ha conseguido.

- Conseguir que al introducir datos en *HubSpot*, automáticamente se sincronicen con *Workday*.

  Una vez que se crea un *deal* en *HubSpot*, si corresponde se crea automáticamente un proyecto en *Workday*.

- Que al modificar datos en *HubSpot*, se modifiquen en *Workday* de forma automática.

  Cuando un *deal* es modificado en *HubSpot*, y este tiene un *project* asociado en *Workday*, el *project* asociado también se ve modificado. Esta sincronización ocurre con la modificación de cualquiera de las propiedades de un *deal*, incluso si modificamos el objeto *company* asociado.

- Que la integración sea rápida, y el tiempo transcurrido entre la introducción o cambio de datos en *HubSpot* y los cambios en *Workday* sea el mínimo posible.

Se ha conseguido que la integración sea instantánea gracias a las suscripciones a los *webhook* de *HubSpot*. Se ha evitado así la creación de un proceso recurrente que compruebe periódicamente si ha habido algún cambio en *HubSpot*.

- La integración ha de ser segura. Estar provista de mecanismos para evitar posibles ataques de terceras personas.

  Se ha conseguido garantizar la seguridad del sistema, añadiendo comprobaciones sobre los mensajes recibidos, para asegurar su procedencia.

- La integración debe ser totalmente transparente para el usuario.

  El usuario no necesita saber como funciona la integración. Simplemente creando o modificando objetos en *HubSpot* consigue que la información se sincronice en *Workday*. Sin embargo los usuarios deben tener presente en que fases un *deal* es sincronizado y en que fases se excluye de la sincronización.

- El programa no debe abortar su ejecución de manera inesperada.

  Gracias al *framework* utilizado *web.py*, no es posible que el programa finalice su ejecución de forma inesperada. Ya que internamente *web.py* captura todas las excepciones y las muestra por consola. En cualquier caso, se ha intentado en la medida de lo posible controlar todos los tipos de errores y excepciones que pudiesen ocurrir.

- En la medida de lo posible evitar errores en la introducción de datos. Evitar que se cree información duplicada.

  Este objetivo no se ha podido llevar a cabo, ya que la duplicación de la información en *HubSpot* es un error del usuario. Si el usuario no se da cuenta de la existencia de un *deal* o *company*, puede crear un duplicado y que consecuentemente se sincronice, cuando eso no es el resultado esperado.

- El servicio que realiza la integración tiene que estar en ejecución ininterrumpida.

  La Interfaz *HubSpot-Workday* se encuentra en ejecución en un servidor de *amazon*, activo todo el tiempo. Esta solución es idónea, pues evita los inconvenientes que puede conllevar mantener un servidor interno en marcha.

- Garantizar la sincronización de aquellos datos que cumplen los requisitos para ser integrados.

  Para garantizar esto se ha creado un proceso que revisa todos los *deals* e integraría aquellos *deals* que no están integrados. Ya sea debido a una interrupción del servicio o la pérdida de algún mensaje.

  Sin embargo no se ha podido comprobar el correcto funcionamiento de este proceso.

- Localmente se debe llevar la cuenta de los datos que se encuentran integrados.

  Se ha conseguido mediante el uso de una base de datos sencilla. Sin embargo no está totalmente protegida de errores de los usuarios, pues por ejemplo si un *deal* se pase a la fase de cierre por equivocación, se para la sincronización del mismo. De manera que es necesario un cambio manual en la tabla de *deals* excluidos.

- El servicio debe soportar ser reiniciado.

  Gracias al almacenamiento persistente de los *token*, es posible reiniciar el sistema sin problemas de funcionamiento. No obstante es posible que problemas de pérdidas de datos tengan lugar y sea necesario tomar medidas(Como por ejemplo ejecutar el proceso que revisa todos los *deals*).

En la parte del estudio de los datos de los trabajadores de una empresa, he podido comprobar las ventajas que presenta el lenguaje *python* tanto para el análisis de datos como para realizar *Machine Learning*. A pesar de no haber usado lenguajes como *R* o *Scala* puedo asegurar que *python* es una buena opción cuando se trata con proyectos de *Machine Learning*.

Ahora vamos a ir comprobando uno a uno los objetivos establecidos para el prototipo predictor en el capítulo de introducción, para ver si se han cumplido.

- El prototipo predictor debe ser capaz de calcular la probabilidad de que un empleado abandone la empresa.

  Este objetivo se ha cumplido, aunque quizá se podría haber alcanzado mejores resultados haciendo uso de datos más fiables.

- Tras el estudio se debe concluir que características son más importantes para predecir si un empleado va a abandonar el trabajo.

  Hemos podido averiguar que datos son claros indicadores a la hora de predecir las rotaciones en la plantilla de una empresa. Por ejemplo hacer horas extra, el salario o la distancia del trabajo a casa son características importantes para la predicción del abandono en una empresa.

- Poder tomar mediadas de acuerdo a las conclusiones a las que se llegue para elaborar mejores predicciones.

  Gracias a la información obtenida con el estudio y la predicción, se puede recomendar medidas de actuación que reduzcan estos sucesos en las empresas. Por ejemplo la realización de una encuesta puede ayudar a recabar información desconocida sobre los empleados, como la conciliación de la vida laboral con la familiar, la satisfacción en el trabajo...

Capítulo 6

Trabajo futuro

En un futuro se mejorará la Interfaz *HubSpot-Workday* portandola a la plataforma de desarrollo propia de *Workday*, *Workday Studio*. Aún así será necesario mantener un servicio intermedio que reenvíe los mensajes desde *HubSpot* hacia *Workday*.

Hasta que finalmente se haga dicha migración será necesario realizar algunas mejoras.

A continuación se citan algunas ideas sobre el trabajo futuro que se podría realizar en la Interfaz *HubSpot-Workday*.

- Implementar una funcionalidad capaz de detectar la introducción de información duplicada. Cuando un usuario introduzca por error información duplicada en *HubSpot* se debe notificar al usuario sobre el problema. De esta forma se evitaría la sincronización de información duplicada en *Workday*.

  Por ejemplo en la creación de un nuevo *deal*, buscar si existe algún *deal* con el nombre igual o muy parecido al que se esta creando y avisar al usuario para que en última instancia decida como resolver el problema.

- Realizar las pruebas necesarias en el proceso recurrente que comprueba la integridad de los datos entre *HubSpot* y *Workday*. De manera que pueda ser añadido al entorno de producción.

- Diseñar una interfaz de usuario para poder realizar ciertas acciones mientras la Interfaz *HubSpot-Workday* está en ejecución. Esta interfaz de usuario por ejemplo podría ser accedida a través de una *url* y permitiese lanzar procesos, visualizar información sobre la base de datos local, actualizar la base de datos manualmente o comprobar que todos los datos están correctamente sincronizados.

Ahora vamos a ver el trabajo futuro que se podría hacer en el prototipo predictor.

La idea principal sería implementar el modelo predictor en un entorno de producción. En dicho entorno debería operar en tiempo real, notificando con un aviso a las personas correspondiente cuando la probabilidad de abandono calculada para un empleado supere cierto umbral.

Para conseguirlo habría que ir completando una serie de tareas.

- Una de las primeras tareas sería desarrollar un modelo utilizando los datos reales de la empresa. Para ello podría ser necesario recabar información que se desconozca sobre los empleados. Por ejemplo la conciliación de la vida laboral con la familiar, número de empresas en las que ha estado el empleado...

- En este desarrollo del modelo predictor, se haría un estudio profundo sobre las características y un proceso de *Feature engineering*.

- Como trabajo futuro también se podría tener en cuenta más estimadores o técnicas para elegir estimadores, como por ejemplo *Ensemble methods*.

- Por último habría que integrar este modelo con el sistema de *Workday*. De manera que en tiempo real el modelo fuese entrenandose y realizando predicciones.

Bibliografía

[Alp10] Ethem Alpaydin. *Introduction to machine learning.* The MIT Press, 2010.

[And01] Lynn Anderson. *Understanding PeopleSoft 8.* Sybex, 2001.

[Bow15] Michael Bowles. *Machine Learning in Python: essential techniques for predictive analysis.* John Wiley & Sons, Inc., 2015.

[Bur15] Roberto Núñez Burgos. *Software ERP análisis y consultoría de software empresarial.* Createspace Independent Publishing Platform, 2015.

[HRP03] Liaquat Hossain, Mohammad A. Rashid, and Jon David. Patrick. *Enterprise resource planning: global opportunities and challenges.* Idea Group Pub., 2003.

[hsa] Api overview — hubspot api. `https://developers.hubspot.com/docs/overview`. [Internet; descargado 4-junio-2017].

[sci] An introduction to machine learning with scikit-learn. `http://scikit-learn.org/stable/tutorial/basic/tutorial.html`. [Internet; descargado 4-junio-2017].

[SSBD16] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning:from theory to algorithms.* Cambrige University Press, 2016.

[Wik17] Wikipedia. Sistema de planificación de recursos empresariales — wikipedia, la enciclopedia libre. `https://es.wikipedia.org/w/index.php?title=Sistema_de_planificaci%C3%B3n_de_recursos_empresariales&oldid=99393524`, 2017. [Internet; descargado 4-junio-2017].

[wws] Workday web services (wws) directory (v28.1). `https://community.workday.com/custom/developer/API/index.html`. [Internet; descargado 4-junio-2017].

Acrónimos

API *Application Programming Interface*. 26, 29–31, 40, 41, 53

BNB *Business Network Builders*. 5, 7, 9, 17, 53

CEO *Chief Executive Officer*. 22

CRM *Customer Relationship Management*. 7, 9, 17, 21, 23

CSV *Comma-Separated Values*. 43

ERP *Enterprise Resource Planning*. 7, 9, 17, 21, 22, 31

HTTP *Hypertext Transfer Protocol*. 26, 32, 33, 37, 39

JSON *JavaScript Object Notation*. 30, 39–41

MRP *Materials Requirement Planning*. 21

MRP II *Manufacturing Resources Planning*. 21

REST *Representational State Transfer*. 29, 31

SOAP *Simple Object Access Protocol*. 32–35, 42

WSDL *Web Services Description Language*. 32, 33

XML *Extensible Markup Language*. 34, 35

XSLT *Extensible Stylesheet Language Transformations*. 34