UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

ESTUDIO DE TÉCNICAS DE RECOMENDACIÓN APLICADA A DATOS TURÍSTICOS DE UNA RED SOCIAL

José Luis Romero Lluch
Tutor: Alejandro Bellogín Kouki
Ponente: Pablo Castells Azpilicueta

FEBRERO 2018

ESTUDIO DE TÉCNICAS DE RECOMENDACIÓN APLICADA A DATOS TURÍSTICOS DE UNA RED SOCIAL

AUTOR: José Luis Romero Lluch
TUTOR: Alejandro Bellogín Kouki
PONENTE: Pablo Castells Azpilicueta

Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Enero de 2018

Resumen (castellano)

Este Trabajo Fin de Grado presenta todo el proceso de desarrollo de la aplicación TweetAndTrip: un sistema de recomendación de eventos turísticos a partir de los datos disponibles en Twitter. Esta aplicación se basa en que una buena forma de conocer a los usuarios es mediante lo que expresan en redes sociales. Para ello, se ha decidido utilizar la red social Twitter; donde en tan solo unas pocas líneas, los usuarios sintetizan una opinión, un sentimiento o una experiencia. Por tanto, desde el inicio de TweetAndTrip, se ha intentado reflejar todos estos comentarios en nuestro proyecto: si un usuario escribe asiduamente sobre un tema, esperamos poder recomendarle destinos relacionados con ese tema y actividades que poder realizar en él. En particular, queremos explorar cómo se pueden explotar los datos demográficos del usuario para mejorar sus recomendaciones.

Para conseguir este objetivo se ha desarrollado una aplicación cuyo propósito es recomendar viajes y actividades a los usuarios de Twitter. Para que sea posible, previamente se han recolectado datos mediante el uso de diferentes APIs de redes sociales, estos datos han sido analizados para ver realmente cuáles son los importantes que dictan lo que a un usuario le gusta. Intencionadamente, se ha elegido Twitter por la posibilidad de explotar los *hashtag*, es decir, aquellas palabras que empiezan por # y que los usuarios escriben en sus tweets como identificadores o palabras clave del mismo. En este proyecto los hemos utilizado como síntesis de los mensajes.

Tras conseguir suficientes datos, se ha construido un modelo y se ha evaluado mediante algoritmos clasificadores. Además, a las recomendaciones de destinos que hemos obtenido, se le asocian y añaden actividades, las cuales obtenemos a partir de los hashtags de los usuarios.

Finalmente, TweetAndTrip cuenta con una aplicación web, donde el lado del servidor se compone de una API REST que está escuchando las peticiones que nos hagan los usuarios de Twitter mediante la parte frontal de la aplicación.

Palabras clave (castellano)

Red social, Twitter, hashtag, API, clasificador, recomendación, aplicación web.

Abstract (English)

This Bachelor Thesis describes the whole development process of the TweetAndTrip application: a recommendation system tailored to tourism events based on user data available on Twitter. This application assumes that a good way to know the users is through what they express on social networks. To achieve this, we have decided to use the Twitter social network; where in few lines, users synthesize an opinion, a feeling, or an experience. Therefore, since the beginning of TweetAndTrip, we have tried to reflect all those comments in our project: if a user often writes about a topic, we expect the recommended destinations to be related with such topic and with those activities that can be done there. In particular, we aim to explore how demographic data from the user can be exploited to improve the recommendations.

To achieve this goal, we have developed an application whose purpose is to recommend trips and activities to the users of Twitter. To make this possible, we have previously collected data while using different APIs from social networks, these data have been analyzed and checked to discern which features are more important in order to learn what the user is really interested in. We selected Twitter on purpose because it is possible to exploit the *hashtags*; i.e., those words that start with a # and that are written by the users in their tweets as identifiers or keywords of that tweet. We will use them in this project as a summary of the messages.

After collecting enough data, we have built and evaluated a model by using classifier algorithms. Besides, once the destination recommendations were obtained, we added venues to these recommendations based on the users' hashtags.

Finally, TweetAndTrip also includes a web application, where in the server side includes an API REST that listens to the Twitter users' requests made from the front-end application.

Keywords (inglés)

Social network, Twitter, hashtag, API, classifier, recommender, web application.

Agradecimientos

A toda mi familia, que nunca han dejado de apoyarme y de creer que el final era posible en este largo camino.

A Miri, que me ha dado el último empujón que necesitaba para terminar esta etapa.

A mis amigos, en especial a los de la universidad, con los que tengo y tendré grandes recuerdos dentro y fuera de clase.

A Alejandro, mi tutor, por su profesionalidad y por haber sabido guiarme durante todo el desarrollo del proyecto.

INDICE DE CONTENIDOS

1 Introducción 1
1.1 Motivación 1
1.2 Objetivos 1
1.3 Organización de la memoria 2
2 Estado del arte 3
2.1 Redes sociales y la información que proporcionan 3
2.1.1 Introducción 3
2.1.2 Aplicaciones similares basadas en la utilización de APIs de redes sociales 4
2.1.2.1 WeePlaces (API de Foursquare) 4
2.1.2.2 FriendShuffle (APIs de Facebook y Twitter) 4
2.1.2.3 Paper.li (API de Twitter) 4
2.1.2.4 FriendAnalyzer (API de Facebook) 5
2.1.3 Conclusiones 5
2.2 Contexto técnico del proyecto 5
2.2.1 Tecnologías relevantes para la implementación del proyecto 5
2.2.1.1 Back-End del proyecto 5
2.2.1.2 Base de Datos 6
2.2.1.3 Software de clasificación 7
2.2.1.4 Front-End del proyecto 7
2.2.2 Conclusiones 8
3 Análisis y Diseño 11
3.1 Análisis 11
3.1.1 Requisitos Funcionales 11
3.1.1.1 Recolección de datos 11
3.1.1.2 Evaluación de un modelo 11
3.1.1.3 API REST 11
3.1.1.4 Interfaz gráfica del Proyecto 12
3.1.1.5 Otros 12
3.1.2 Requisitos No Funcionales 12
3.1.2.1 Seguridad 12
3.1.2.2 Interfaz 12
3.1.2.3 Disponibilidad 12
3.1.2.4 Rendimiento 13
3.1.2.5 Mantenibilidad y portabilidad 13
3.1.2.6 Recursos 13
3.1.2.7 Fiabilidad 13
3.1.3 Planificación del proyecto 13
3.2 Diseño 14
3.2.1 Arquitectura del proyecto 14
3.2.2 Estructura de datos 15
3.2.3 Diseño de clases 17
3.2.4 Casos de uso 20
4 Desarrollo 21
4.1 Introducción 21
4.1.1 Conceptos generales 21
4.1.2 Software utilizado 22
4.2 Desarrollo del proyecto 23
4.2.1 Recolección de datos 23
4.2.1.1 Elección de librería para la API de Twitter 23

4.2.1.2 Conexión a base de datos 24
4.2.1.3 Interacción con base de datos desde Java 24
4.2.1.4 Integración de los datos de Twitter en la base de datos 24
4.2.2 Construcción y evaluación del modelo 25
4.2.2.1 Construcción del modelo 25
4.2.2.2 Evaluación del modelo 26
4.2.3 Implementación de una API REST 27
4.2.4 Aplicación Web 28
4.2.4.1 Creación de una aplicación en Angular 5 28
4.2.4.2 Librerías de Componentes: Bootstrap y Angular Material 28
4.2.4.3 Componentes 29
4.2.4.4 Routing 30
4.2.4.5 Llamada HTTP 30
4.2.4.6 Resultado final 30
4.3 Problemas encontrados 32
4.3.1 Límite de memoria en Windows al entrenar modelos con Weka 32
4.3.2 Eliminación de la descripción y versión single-tag del modelo 32
4.3.3 Volcado de base de datos en CSV, eliminar espacios, \n, \t, caracteres especiales, etc 33
4.3.4 Género y edad en Twitter 33
5 Integración, pruebas y resultados 35
5.1 Introducción 35
5.1.1 Pruebas unitarias 35
5.1.2 Pruebas de integración 35
5.1.3 Pruebas de sistema 36
5.2 Resultados 37
6 Conclusiones y trabajo futuro 41
6.1 Conclusiones 41
6.2 Trabajo futuro 41
Referencias 43
Glosario 45
Anexos ..... - 2 -
A Manual de uso ..... - 2 -

INDICE DE FIGURAS

FIGURA 2-1: RANKING DE LAS REDES SOCIALES MÁS FAMOSAS POR NÚMERO DE USUARIOS ACTIVOS (EN MILLONES) EN 2017 [3] 3

FIGURA 2-2: REACT, ANGULAR Y POLYMER, TECNOLOGÍAS FRONT-END 8

FIGURA 3-1: PLANIFICACIÓN TEMPORAL DEL PROYECTO 13

FIGURA 3-2: ARQUITECTURA DEL PROYECTO 14

FIGURA 3-3: DISEÑO DE LA BASE DE DATOS 15

FIGURA 3-4: DIAGRAMA DE CLASES DE LA APLICACIÓN 17

FIGURA 3-5: DIAGRAMA DE CASOS DE USO DE LA APLICACIÓN 20

FIGURA 4-1: ARQUITECTURA CLIENTE-SERVIDOR 21

FIGURA 4-2: ATRIBUTOS DEL MODELO EN WEKA 25

FIGURA 5-1: OUTPUT DEL PROGRAMA JAVA 36

FIGURA 5-2: OUTPUT DE LOS RESULTADOS PARA N=5 36

FIGURA 5-3: OUTPUT PARA UN N FUERA DE RANGO 36

FIGURA 5-4: RESULTADOS EN WEKA PARA CLASIFICACIÓN PARA 10 PARTICIONES 37

FIGURA 5-5: DETALLES DE RESULTADOS PARA 10 PARTICIONES 37

FIGURA 5-6: RESULTADOS EN WEKA PARA CLASIFICACIÓN POR PORCENTAJE 80% 38

FIGURA 5-7: DETALLES DE RESULTADOS PARA PORCENTAJE 80% 38

FIGURA 5-8: EVALUACIÓN DEL MODELO CON DISTINTAS N 38

INDICE DE TABLAS

TABLA 2-1: COMPARATIVA DE LAS PRINCIPALES TECNOLOGÍAS SQL 7

TABLA 5-1: RESULTADOS PARA USUARIOS CONCRETOS 39

1 Introducción

1.1 Motivación

A día de hoy las redes sociales siguen creciendo, y con el auge laboral de tecnologías como la minería de datos, Big Data y la inteligencia artificial, resulta comprensible que las empresas estén muy interesadas en la utilización de las redes sociales como fuente de información para sus propios objetivos.

Por este motivo, nace la idea de realizar el proyecto que llamaremos **TweetAndTrip** y que se describe en este documento. En este proyecto se quiere experimentar sobre datos reales y crear una aplicación que los utilice con algún propósito concreto. En este caso, el propósito son los viajes y las actividades; en muchas ocasiones nos encontramos sin tener del todo claro qué lugar nos gustaría visitar ni qué posibilidades nos ofrece. Esto último es importante, ya que podemos querer ir a un destino concreto del que nos han hablado, pero no tenemos la posibilidad de descubrir nada nuevo de lo que no tengamos referencia.

Además, existe una fuerte motivación personal por ver la importancia que pueden llegar a tener datos sociales y demográficos como la localización o el idioma del usuario. Resulta muy interesante ver que a una persona por el simple hecho de ser de un país y tener unos determinados gustos, podría llegar a encajarle mejor una visita a un país o a otro con diferentes actividades.

Por último, a todo esto, hay que sumarle otro motivo personal, y es la pasión por viajar y conocer otras culturas, es por ello que se han elegido los viajes como tema central de este trabajo.

1.2 Objetivos

Desde el principio del proyecto se ha tenido clara la idea de realizar una aplicación *end-to-end*, con el objetivo de darle una experiencia lo más completa posible al usuario final que utilice la aplicación. Además, existe el objetivo personal de conseguir construir una aplicación intentando unir conceptos de distintas asignaturas de la carrera, con todas las partes necesarias para que pudiera funcionar y ser usada por cualquier persona.

Por tanto, la lista de objetivos principales es la siguiente:

- Ser capaz de interactuar con las diferentes redes sociales que se utilizarán en el proyecto de forma que se consiga la información necesaria para la recomendación de viajes y actividades.

- Gestionar un entorno con diferentes conexiones, servidores y bases de datos.

- Evaluación y construcción de un modelo de datos para su posterior explotación. Además, se hará uso de algoritmos clasificadores para conseguir las recomendaciones y, a su vez, evaluar nuestro modelo.

- Crear una Aplicación Web completa para que los usuarios puedan recibir recomendaciones, la cual utilizará todo el proyecto y se esperará que los resultados tengan realmente una relación entre el usuario y el destino recomendado.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- Estado del arte: En esta sección se realizará un repaso al contexto actual del proyecto, donde veremos el estado de las redes sociales y otros proyectos similares de los que podremos tomar ideas. Además, se verá un apartado de contexto técnico donde veremos las alternativas de implementación que podríamos utilizar.

- Análisis: Se realizará un análisis de los requisitos funcionales y no funcionales de nuestra aplicación. Con el objetivo de tener claro desde el principio al punto que se quiere llegar.

- Diseño: Se explicará el diseño elegido para la aplicación y todas sus partes. Desde la base de datos hasta la implementación realizada en Java.

- Pruebas y resultados: Se explicarán las pruebas que se han realizado, así como los resultados que hemos ido obteniendo durante el desarrollo del proyecto.

- Conclusiones: Breve síntesis de todo lo que nos ha aportado el proyecto, así como de su estado actual. También se hará un repaso a posibles mejoras.

2 Estado del arte

2.1 Redes sociales y la información que proporcionan

En este apartado se realizará un resumen del estado actual de las diferentes aplicaciones que tienen como principal objetivo utilizar los datos obtenidos de redes sociales para diferentes propósitos.

2.1.1 Introducción

A día de hoy, a pesar de tener la sensación de que todo está inventado, las redes sociales siguen evolucionando y desarrollando diferentes métodos para captar más usuarios.

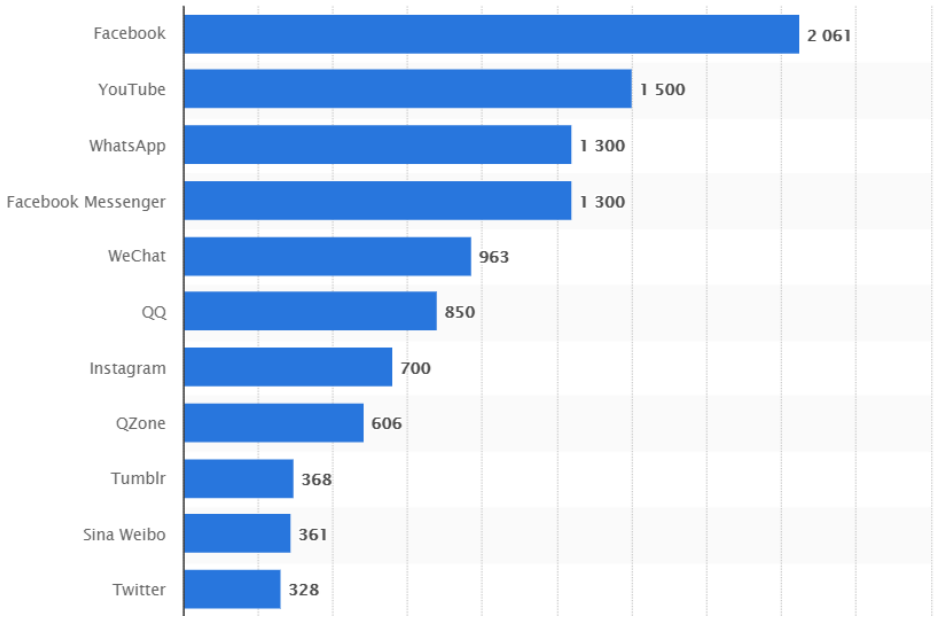


Figura 2-1: Ranking de las redes sociales más famosas por número de usuarios activos (en millones) en 2017 [3]

Para entender mejor el gráfico, podemos ver que en la actualidad Facebook cuenta con más de 2.000 millones de usuarios activos, lo que supone un incremento del 16% respecto a 2016 [7]. En el caso de Twitter, el incremento es de un 5% respecto al año anterior.

Además, lo fascinante de estos sistemas no es solamente la cantidad creciente de gente que los utilizan, si no el uso que le dan. Las redes sociales son, actualmente, la fuente de información más fiable que existe para obtener datos de una sociedad. Para contrastar esto, solo hace falta ver algunos de los datos principales relacionados con Twitter [8]:

- 500 millones de Tweets diarios (2017)
- 100 millones de usuarios diarios activos (2017) [4]
- 500 millones de personas visitan Twitter sin tener cuenta mensualmente (2016)

- La cuenta con más seguidores del mundo es la de Katy Perry con 108 millones (2017) [1]
- 80% de los usuarios que utilizan Twitter es a través del móvil (2017)

No es complicado imaginarse, a través de esta información, lo conectados que estamos y la facilidad que existe con las redes sociales de compartir prácticamente cualquier cosa en cualquier momento.

2.1.2 Aplicaciones similares basadas en la utilización de APIs de redes sociales

Viendo la información que nos proporcionan las redes sociales, es lógico que aparezcan aplicaciones que quieran beneficiarse de estos datos. Para ello, suelen proporcionar una API para que los desarrolladores puedan acceder a cierta información pública de forma que sea más sencillo explotarla en una aplicación.

Algunos ejemplos que podemos encontrar son:

2.1.2.1 WeePlaces (API de Foursquare)

Foursquare es una red social para encontrar lugares y actividades basados en los filtros que el usuario quiera aplicar. En todas las actividades se pueden dejar consejos, reseñas, etiquetas, fotos y otra información útil, por lo que existe una interacción indirecta y real entre los usuarios.

WeePlaces lleva esta API a otro nivel, permitiendo que los usuarios puedan ver las rutas que han ido realizando entre las diferentes actividades a las que se han apuntado. Es una forma objetiva para ver los intereses que tenemos a través de un mapa.

2.1.2.2 FriendShuffle (APIs de Facebook y Twitter)

Tanto la API de Facebook como la de Twitter proporcionan información relacionada con los usuarios que la utilizan. Esta información siempre es pública, y suelen ser datos personales como el nick, el nombre, el género o la localización del perfil del usuario. Además de esta información personal, proporcionan el método para conseguir las publicaciones de cada persona.

FriendShuffle es una aplicación muy sencilla, la cual se centra en unificar las dos redes sociales en una sola página. De forma que podamos ver la actividad de nuestros amigos independientemente de si se ha producido en Twitter o en Facebook.

2.1.2.3 Paper.li (API de Twitter)

Esta aplicación se basa en filtrar la información que el usuario lee diariamente. En vez de tener que leer todos los perfiles que el usuario sigue, Paper.li realiza un resumen basándose en los intereses del perfil para mostrarle solamente lo que podría considerar relevante.

2.1.2.4 FriendAnalyzer (API de Facebook)

FriendAnalyzer tiene como objetivo encontrar entre los amigos del usuario, al perfil más similar con el principal. A simple vista, puede parecer una aplicación con poca utilidad ya que, si tienes una cantidad de amigos normal, es posible que no requieras de esta funcionalidad. Pero si tienes una red muy grande, o mejor aún, si eres una empresa, es posible que te resulte muy interesante saber con quién tienes que contactar.

2.1.3 Conclusiones

A pesar de la valiosa información que nos proporcionan las diferentes APIs de cada red social, no parece que exista un número de aplicaciones destacable que quieran beneficiarse de las mismas. Esto probablemente se deba a que cada empresa realiza su propio sistema para explotar esta información. Hoy en día, es muy común encontrar ofertas de trabajo donde se pida experiencia en el uso de APIs, Big Data y minería de datos.

También hay que considerar que estas APIs tienen sus restricciones, como puede ser el número de llamadas diarias o los datos privados que no pueden proporcionar.

Por tanto, hay que tener muy claro el objetivo al que se quiere llegar antes de realizar una aplicación que involucre la utilización de cualquiera de estos sistemas.

En el caso de la aplicación desarrollada en este proyecto, se supo desde el principio que se querían utilizar los datos personales de perfil de usuario y las publicaciones que realizaran (Twitter o Facebook) así como recomendar actividades y lugares que visitar (Foursquare).

La balanza terminó a favor de Twitter debido a los tags que presentan los Tweets. Estos tags generalmente son información concreta y valiosa, por lo que combina perfectamente con la API de Foursquare para conseguir lo que buscamos: recomendar destinos con actividades.

2.2 Contexto técnico del proyecto

En esta sección se va a realizar un breve vistazo sobre las diferentes tecnologías que se podrían haber utilizado como alternativa para la realización de un proyecto con APIs de redes sociales y recomendadores.

2.2.1 Tecnologías relevantes para la implementación del proyecto

2.2.1.1 Back-End del proyecto

Para poder utilizar la API de Twitter encontramos que en la propia sección de desarrollador nos recomiendan distintas librerías en diferentes lenguajes.

A pesar de no estar dentro de las principales redes sociales en cuanto a usuarios activos, podemos ver que las posibilidades que tenemos son muy amplias y variadas.

Dentro del listado, dividen las librerías en dos secciones (la lista de Twitter está algo desactualizada y algunas librerías no tienen un link correcto, sin embargo, la mayoría siguen activas si las buscamos en Google):

Librerías desarrolladas por Twitter

- **Twitter Kit (Android)**: Librería muy completa que se compone de diferentes módulos: Un Core para realizar funciones de autenticación mediante SSO. Vistas para presentar la información correctamente en la app. Composer para que los usuarios puedan interactuar con la aplicación.
- **Horsebird Client (Java)**: Librería para capturar el streaming de Twitter.
- **Twitter Kit (iOS)**

Librerías desarrolladas por la comunidad de Twitter

- **Temboo (Multiplataforma**): Framework para trabajar con Twitter a través de diferentes plataformas entre las que encontramos iOS, Android, Java, PHP, Python, Ruby y Node.js.
- **Twitcurl (C++)**: Alternativa para los desarrolladores de C++. Funciona correctamente en cualquier sistema operativo que soporte cURL.
- **Twitter4J (Java)**: Librería que además de tener la posibilidad de capturar el streaming de los tweets, integra todas las funcionalidades que la API de Twitter nos ofrece.
- **Swifter (Swift**): Alternativa para los desarrolladores de Swift.

2.2.1.2 Base de Datos

Para este tipo de proyectos es imprescindible contar con una base de datos donde poder almacenar toda la información que obtenemos de las redes sociales.

Es importante, antes de empezar a desarrollar, estudiar las distintas alternativas que podemos implementar en un sistema similar.

En primer lugar, realizamos un análisis sobre si al proyecto le vendría mejor una base de datos SQL o una noSQL.

En el caso de TweetAndTrip, como veremos en el apartado de diseño, existen diferentes tablas relacionadas con las que operaremos, por lo que es más lógico utilizar una base de datos relacional SQL.

Por otro lado, noSQL tiene un punto importante a su favor, y es el rendimiento. Sin embargo, nuestra interacción con la base de datos se usará solamente en la primera fase del proyecto, donde llenaremos las tablas de información para posteriormente clasificar. Como este rendimiento no es estrictamente necesario para nuestro ámbito, SQL parece ser la opción más adecuada [5].

Tomada una primera decisión, es importante conocer algunas ventajas y desventajas que pueden ofrecernos los tres sistemas principales de SQL:

Programa	Ventajas	Desventajas
PostgreSQL	• Gratuito • Mayor seguridad	• Velocidad de respuesta más lenta
MySQL	• Gratuito • Instalación sencilla • Mayor velocidad	• Poco intuitivo
ORACLE	• El más utilizado • Fácil de usar	• De pago

Tabla 2-1: Comparativa de las principales tecnologías SQL

2.2.1.3 Software de clasificación

- **Weka**: Plataforma *open source* de software para el aprendizaje automático y la minería de datos escrito en Java. Tiene además una librería escrita en Java la cual es sencilla de implementar. Cuenta con una comunidad grande detrás, lo que hace fácil resolver los problemas más básicos.

- **Scikit-learn**: Librería *open source* en Python para *Machine Learning*, contiene una amplia lista de algoritmos para clasificación y regresión. Uno de sus principales problemas es que no soporta modelos demasiado grandes [2].

- **MLLib Spark**: Librería de Spark para *Machine Learning*, se puede utilizar tanto en Scala, como en Java o Python. Al contrario que en la librería anterior, no existe esa limitación en el tamaño del modelo, por lo que, si pensamos trabajar con modelos de gran tamaño, esta opción parece la más adecuada.

- Dentro del campo de *Machine Learning*, encontramos librerías especializadas en el procesamiento del lenguaje, algunas de ellas son: **Standford NLP** (Java), **MALLET** (Java), que también incluye técnicas de clasificación y **Natural Language Toolkit** (Python).

2.2.1.4 Front-End del proyecto

Desde el comienzo del proyecto, se tuvo como intención principal realizar una aplicación completa. Esto implica realizar tanto la base de datos, como el programa que explote la información obtenida de Twitter, y la aplicación para utilizar una API REST que construiremos. Esta aplicación tendrá un frontal donde poder mostrar los datos que nos devuelve nuestra API.

El concepto de nuestra aplicación final es muy similar al de una SPA (Single Page Application), por lo que prácticamente con cualquier tecnología nos sería suficiente. Un buen candidato a utilizar habría sido Backbone.js con Javascript. Es un framework muy ligero cuya única dependencia es Underscore.js. Lo cual es perfecto cuando el objetivo es realizar páginas sencillas sin demasiadas funcionalidades.

A pesar de conocer las ventajas de utilizar un framework más simple, durante el desarrollo del proyecto se ha buscado también utilizar las últimas tecnologías que nos ofrece el mercado, con el fin de explorar nuevas alternativas.

Por este motivo se eligió entre las diferentes librerías y frameworks basados en componentes que están en auge en la actualidad:

Figura 2-2: React, Angular y Polymer, tecnologías Front-End

- React: Biblioteca para crear interfaces de usuario mantenida por Facebook. Es rápido, fácil de aprender y puede instanciarse con el patrón Modelo-Vista-Controlador.
- Angular 5: Framework mantenido por Google que incluye todo lo necesario para utilizar formularios, routing y llamadas HTTP. Respaldado por una amplia comunidad y documentación.
- Polymer: Biblioteca desarrollada y mantenida por Google. Un desarrollo enfocado principalmente a componentes aislados. Con una curva de aprendizaje más compleja.

2.2.2 Conclusiones

Habiendo realizado un repaso general de las diferentes posibilidades que existen actualmente para implementar un proyecto de este tipo, es hora de elegir las herramientas para su correcto diseño y desarrollo.

Si seguimos el orden en el que hemos ido definiendo el estado del arte tecnológico de este proyecto, empezamos por el Back-End, que además tendrá el mayor peso entre las distintas partes. Para esta parte hemos decidido realizar el proyecto en Java junto con la librería Twitter4J. El primero por la facilidad de desarrollo y documentación que podemos encontrar hoy en día. Y la librería porque dentro de las alternativas que existen, es la única que ofrece un streaming de los tweets (funcionalidad que usaremos más adelante) junto con métodos para utilizar todos los endpoints de la API de Twitter. Además, veremos que la elección de Java no ha sido casual, si no que todas las piezas siguientes se compenetrarán a la perfección al ser un lenguaje tan usado.

En el apartado de base de datos, al echar el primer vistazo a las tecnologías que podíamos utilizar, quedamos en utilizar SQL, por lo que, llegados a este punto, hay que considerar si utilizar ORACLE, PostgreSQL o MySQL. Cualquiera de las tres tecnologías es más que válida para el propósito del proyecto. En el caso de TweetAndTrip, se ha elegido MySQL por una cuestión personal: utilizar la única tecnología que no había implementado nunca.

El siguiente paso consistía en elegir una herramienta de clasificación que encajara con Java de forma sencilla, y dada la experiencia que obtenemos en el uso de Weka durante la carrera, es sin duda la elegida. Probablemente la librería de Spark nos hubiese dado también resultados muy positivos, pero como digo, la experiencia previa con Weka ha inclinado la balanza a su favor.

Por último, teniendo todo el proyecto estructurado a nivel tecnológico, falta la parte frontal, donde la intención es aprovechar esta API para usar alguna tecnología moderna de front-end. En este caso se ha elegido Angular 5, ya que nuestra aplicación va a ser una aplicación sencilla (SPA) y, además, podremos utilizar el cliente que ofrece para crear rápidamente un proyecto al que añadir nuestras vistas. También ha sido elegida por las utilidades que ofrece a la hora de realizar llamadas HTTP e implementar enrutamiento. Por último, cuenta con mucha documentación y comunidad, lo que seguro hará más fácil su desarrollo y aprendizaje.

3 Análisis y Diseño

3.1 Análisis

En esta sección se realizará un análisis de los requisitos que encontramos en la aplicación. Con un correcto desglose de las características que debe tener el proyecto, será mucho más fácil comenzar con un correcto desarrollo.

Recordemos que el proyecto consiste en la implementación de una aplicación que, mediante los datos recopilados de distintos usuarios de Twitter, sea capaz de recomendar viajes y actividades.

3.1.1 Requisitos Funcionales

3.1.1.1 Recolección de datos

- RF1: El sistema será capaz de rellenar la base de datos con los datos de los usuarios que hayan escrito sobre la palabra 'travel'.

- RF2: El sistema será capaz de obtener todos los tags de los tweets de cada usuario e insertarlos en la base de datos.

- RF3: El sistema será capaz de obtener los destinos que los usuarios hayan mencionado en sus tweets.

- RF4: El sistema será capaz de añadir un género a cada usuario basado en su nombre mediante la utilización de la API Genderize.

- RF5: En caso de que el nombre del usuario sea ambiguo, el sistema deberá ser capaz de establecer un género neutro en base de datos.

3.1.1.2 Evaluación de un modelo

- RF6: El sistema deberá contar con la funcionalidad necesaria para realizar una evaluación de un modelo de datos con un clasificador Naive Bayes con la posibilidad de especificar un número N de clases resultantes.

3.1.1.3 API REST

- RF7: El sistema será capaz de levantar una API REST para recomendar destinos y actividades.

- RF8: Dado un nombre de usuario, el servidor deberá poder construir un usuario con los datos de perfil de Twitter.

- RF9: El servidor será capaz de asociar un género a cada usuario.

- RF10: El servidor deberá crear un modelo a partir de los datos recolectados del usuario.

- RF11: Se limitará el número de destinos recomendados a 5.

- RF12: Se limitará el número de actividades asociadas a un destino a 3.

3.1.1.4 Interfaz gráfica del Proyecto

- RF13: Se realizará una pantalla de inicio donde el usuario pueda escribir su nick de Twitter junto con un botón para encontrar destinos.

- RF14: En la página de inicio no se le permitirá al usuario enviar nombres vacíos.

- RF15: Se mostrarán en forma de listado los resultados que devuelva la llamada a la API REST, tanto los destinos como las actividades.

- RF16: En caso de error o falta de permisos, se le mostrará un aviso al usuario.

- RF17: Se realizará una cabecera en la página con tres links: Página de inicio, cuenta de Twitter de TweetAndTrip y cuenta con los repositorios de TweetAndTrip en Github.

3.1.1.5 Otros

- RF18: El servidor del proyecto deberá tener un menú por línea de comandos en el que ofrezca la posibilidad de recolectar datos, lanzar la API o realizar una evaluación por top N.

3.1.2 Requisitos No Funcionales

3.1.2.1 Seguridad

- RNF1: Las propiedades relacionadas con la base de datos estarán externalizadas de forma segura y se alojarán solamente en el servidor sin la necesidad de subirlas al repositorio externo de Github.

- RNF2: Las propiedades relacionadas con la autenticación en la API de Twitter estarán externalizadas de forma segura y se alojarán solamente en el servidor sin la necesidad de subirlas al repositorio externo de Github.

- RNF3: Las propiedades relacionadas con la autenticación en la API de FourSquare estarán externalizadas de forma segura y se alojarán solamente en el servidor sin la necesidad de subirlas al repositorio externo de Github.

3.1.2.2 Interfaz

- RNF4: La interfaz gráfica de la aplicación será sencilla e intuitiva para el usuario. Contará con elementos básicos como botones, inputs o enlaces para no complicar la experiencia del usuario.

3.1.2.3 Disponibilidad

- RNF5: El sistema será capaz de utilizar una API alternativa de género en caso de que la primera llegue al límite diario de llamadas.

3.1.2.4 Rendimiento

- RNF6: El sistema deberá controlar el tiempo de refresco necesario para obtener nuevos datos del streaming de la API de Twitter.

- RNF7: El sistema deberá controlar el tamaño y el número de los tags a insertar para evitar sobrecargas innecesarias en la base de datos.

3.1.2.5 Mantenibilidad y portabilidad

- RNF8: El proyecto deberá ser exportable a un ejecutable de tipo .jar para poder interactuar con el menú de consola.

- RNF9: El proyecto deberá seguir un versionado en un repositorio público de Github.

3.1.2.6 Recursos

- RNF10: No se necesitarán requisitos especiales para poder utilizar la página web.

- RNF11: Para utilizar el programa (API REST, clasificador y la recolección de datos) se necesitará un ordenador con Java.

3.1.2.7 Fiabilidad

- RNF12: El sistema deberá ser capaz de identificar el fallo en caso de que exista algún problema.

3.1.3 Planificación del proyecto

Desde el comienzo del proyecto se acordaron plazos con un trabajo iterativo e incremental de entregas u objetivos.

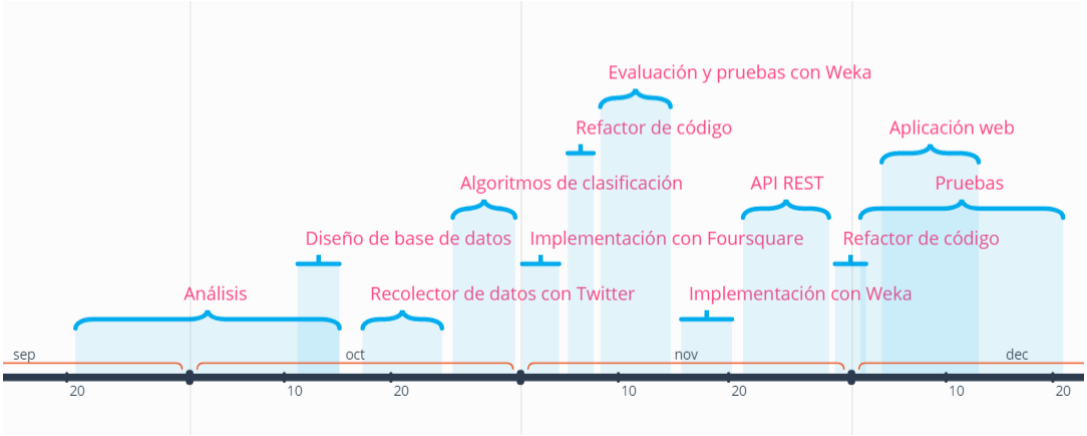


Figura 3-1: Planificación temporal del proyecto

En el gráfico anterior se puede observar una planificación aproximada de lo que fue el proyecto. En general los plazos se cumplieron con normalidad y gracias a realizar un planning inicial, se pudo avanzar y reaccionar a posibles imprevistos que surgieron.

También es muy recomendable realizar una fase generosa de análisis; cuanto más desglosado esté cada elemento del proyecto, luego será mucho más sencillo el desarrollo y la resolución de problemas que deriven del primero.

3.2 Diseño

3.2.1 Arquitectura del proyecto

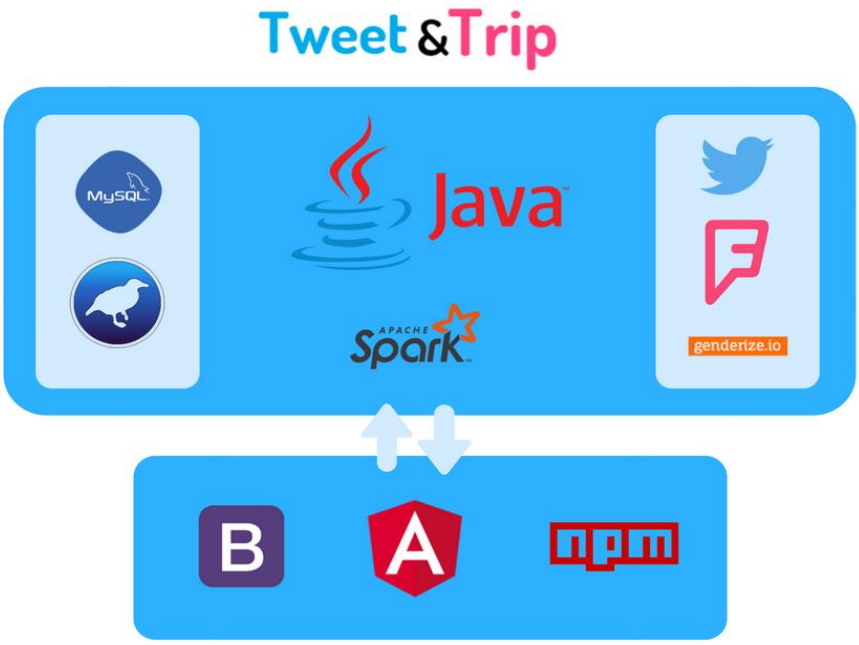


Figura 3-2: Arquitectura del proyecto

Como podemos observar, en la parte superior del diagrama de la arquitectura contamos con todas las tecnologías Back-End del proyecto. En esta parte es donde realizamos la captura de datos principales (Twitter) y auxiliares (Foursquare y Genderize), la construcción y evaluación del modelo (Weka y MySQL) y la inicialización de la API REST (librería JavaSpark). Todo esto tiene un denominador común en donde se encuentra nuestro lenguaje de programación principal (Java).

Por otro lado, contamos con las tecnologías Front-End, donde principalmente hemos utilizado Angular, Bootstrap y npm.

En conjunto, forman la aplicación TweetAndTrip.

3.2.2 Estructura de datos

La estructura de datos que se ha elegido para este proyecto es relativamente sencilla, sin embargo, la tarea para rellenarla ha sido bastante costosa, lo cual explicaremos en el apartado de desarrollo.

El diseño de esta base de datos se realizó durante la fase de análisis, una vez tuvimos clara las posibilidades que nos ofrecían las distintas tecnologías que íbamos a utilizar y nuestra intención con ellas.

Contamos con cuatro tablas que explicaremos a continuación:

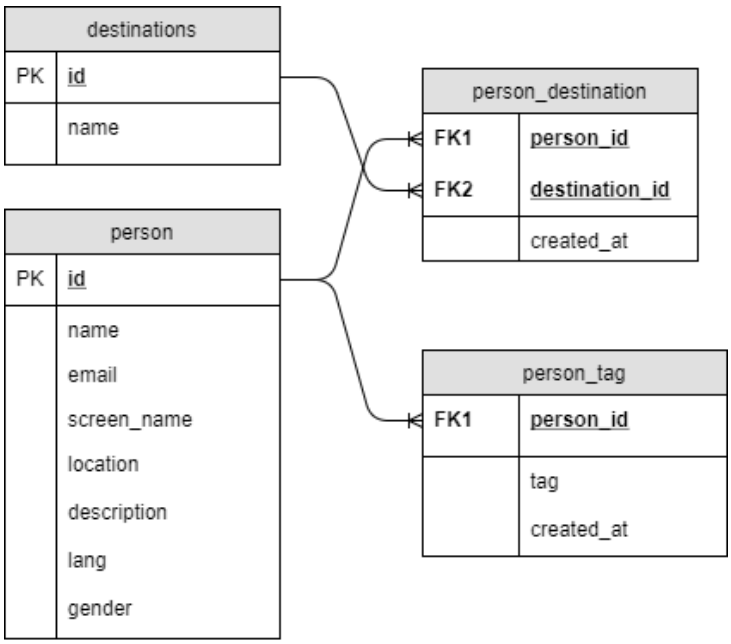


Figura 3-3: Diseño de la base de datos

- **Person**: Tabla que guarda los datos de los usuarios que sacamos de la API de Twitter y que posteriormente utilizaremos para entrenar nuestro modelo.

  Todos los atributos que tiene están sacados de la información de Twitter excepto ‘gender’ (género) y ‘age’ (edad).

  - **id** (Clave Primaria, PK): Identificador del usuario que coincide con el id del propio usuario de Twitter. De tipo BIGINT NOT NULL, el cual, al igual que el que nos retorna Twitter (int64) comprende valores entre:

    {-9223372036854775808, 9223372036854775807}

  - **name**: Nombre del usuario, no tiene por qué ser el nombre de la persona. Twitter lo limita a 20 caracteres, pero según su documentación, es posible que aumente. Por este motivo el tipo del atributo es un CHAR(50) NOT NULL.

  - **email**: Email del usuario. De tipo CHAR(50).

    - **screen_name**: Alias del usuario, limitado a 15 caracteres por Twitter, sin embargo, antiguamente el límite era mayor, por lo que lo hemos limitado a CHAR(50) NOT NULL.

    - **location**: Ubicación del usuario, definida por el mismo. No necesariamente es una ubicación real, lo que complicará en cierto modo la clasificación. De tipo CHAR(50).

    - **description:** Descripción del usuario, escrita por el mismo. La documentación de la API no especifica un límite, sin embargo, al intentar introducir un máximo de caracteres en nuestro perfil de Twitter, podemos ver que el máximo es 160. Por lo que en este caso el tipo es CHAR(160).

    - **lang**: Código del lenguaje en el que el usuario escribe los tweets. Limitado a CHAR(50).

    - **gender**: Género del usuario, obtenido a través de otras APIs que detallaremos en la parte de desarrollo. CHAR(10).

- **Destinations**: Tabla que guarda los nombres de todos los destinos posibles que nuestra aplicación puede llegar a recomendar.

    - **id** (Clave Primaria, PK): Identificador del destino. De tipo MEDIUMINT NOT NULL AUTO_INCREMENT para que se vaya incrementando en función del id del último registro.

    - **name:** Nombre del destino. Cuenta con un total de 419 filas, obtenidos mediante distintos procesos que veremos en la parte de desarrollo.

- **Person_destination**: Tabla de relación 1-n tanto con person como con destinations. En esta tabla se guardan todas las veces que una persona ha mencionado un destino.

    - **person_id** (Clave Foránea, FK): Identificador de la persona.

    - **destination_id** (Clave Foránea, FK): Identificador del destino.

    - **created_at**: Fecha de creación del tweet donde se mencionó el destino. De tipo DATE

- **Person_tag**: Tabla de relación 1-n con person. En esta tabla se guardan todos los hashtags que una persona ha escrito (limitado para no sobrecargar la tabla). En este caso, no se ha creado una tabla auxiliar, ya que no existe una tabla con un listado de tags, por lo que, pudiendo ser cualquier palabra, no es posible valorar el atributo tag como único.

    - **person_id** (Clave Foránea, FK): Identificador de la persona.

    - **tag:** Nombre del tag. CHAR(50).

    - **created_at**: Fecha de creación del tweet donde se mencionó el tag. DATE

3.2.3 Diseño de clases

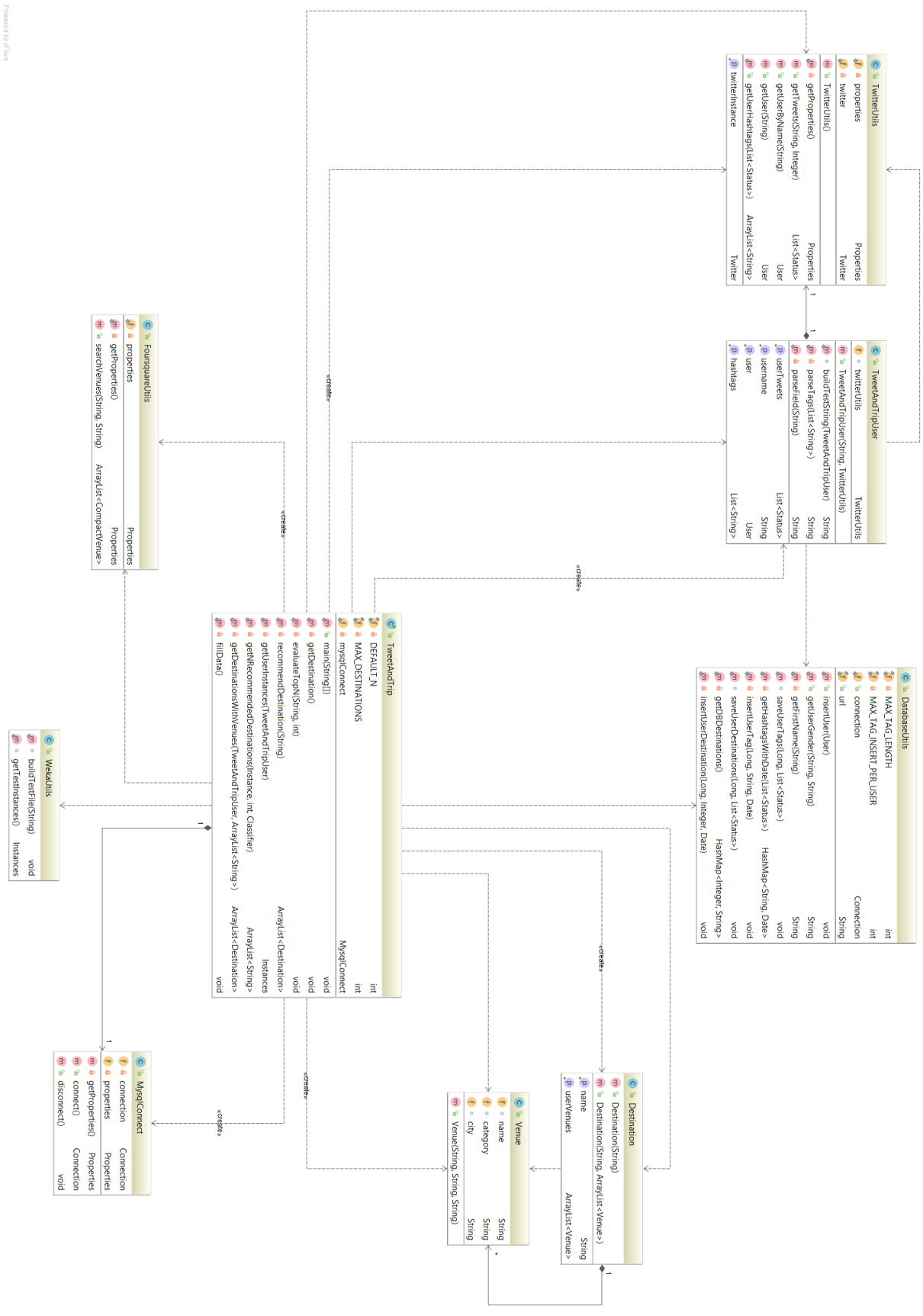


Figura 3-4: Diagrama de clases de la aplicación

El diagrama de clases presentado muestra la estructura principal de la parte Back-end del proyecto. A continuación, se va a explicar (sin entrar en lógica de implementación, la cual veremos en el apartado de desarrollo de la memoria) la funcionalidad de cada una de las clases.

Se ha intentado construir funciones atómicas, como buena práctica, con el propósito de que cada método realice únicamente una función. En relación a esto, también se ha intentado que los nombres de los métodos no sean abstractos ni genéricos, para que, al leer su nombre, indique en la medida de lo posible su finalidad. También se ha realizado un diseño de la aplicación pensando en la privacidad de los métodos y atributos, utilizando los correspondientes modificadores de acceso.

TweetAndTrip

Es la clase principal del proyecto, contiene todos los métodos necesarios para ejecutar las tres opciones que ofrece: API REST, evaluación por top N y recolección de datos para el modelo. Estas funciones son las siguientes:

- **getDestination**(): Método que levanta la API REST, escucha llamadas GET a /destination y recibe como parámetro un nombre de usuario. A su vez, utiliza el método recommendDestinations(nombre) que será el que nos devuelva un conjunto de Destinations recomendados.
    - **recommendDestinations**(nombre): Dividido en distintos pasos:
        1. Construimos nuestro usuario de tipo TweetAndTripUser y construye el fichero de test para poder usarlo contra nuestro modelo (getUserInstances(usuario)).
        2. Con esto obtenemos N destinos mediante getNRecommendedDestinations(datos, N)), donde N está fijada por la variable DEFAULT_N.
        3. Finalmente, a estos destinos le añadimos las actividades que puede realizar mediante el método getDestinationsWithVenues(usuario, destinos).

- **evaluateTopN**(datos, N): Método que permite al usuario obtener datos de clasificación (los que analizaremos en el apartado de resultados). N está limitado mediante la variable MAX_DESTINATIONS. Se realiza una partición de training y otra de test. Finalmente, reutilizamos el método getNRecommendedDestinations(datos, N) para comprobar con los datos de test el acierto del clasificador.

- **fillData**(): Método que recolecta datos para nuestra base de datos. Por lo tanto, utilizando el patrón Singleton, construimos la conexión con la clase DatabaseUtils (mysqlconnect). El resto del método podemos dividirlo en pasos:
    1. Obtener un usuario que haya escrito sobre ‘travel’.

2. Obtenemos todos sus tweets (limitado a 200 para evitar sobrecargas).
3. Insertamos el usuario en nuestra base de datos.
4. Guardamos sus tags en base de datos.
5. Guardamos los destinos que ha nombrado en sus tweets en base de datos.

Explicado el programa principal, podemos dividir el resto de clases en dos tipos:

Utilidades

- **MysqlConnect:** Utilidades para conectar con nuestra base de datos.
- **WekaUtils:** Utilidades para poder construir ficheros que posteriormente la librería de Weka pueda procesar.
- **FoursquareUtils:** Principalmente consta de un método searchVenues(destino, tag), el cual, utilizando la librería de Foursquare, nos devolverá un conjunto de actividades a realizar en un destino.
- **DatabaseUtils:** Colección de comandos y operaciones contra nuestra base de datos.
- **TwitterUtils:** Operaciones con la librería de Twitter para obtener los datos de los usuarios, así como sus hashtags.

Objetos

- **TweetAndTripUser:** Usuario extendido del usuario que nos proporciona la librería de Twitter. Contiene solo los datos que utilizamos en nuestro programa, además de funciones auxiliares para limpiar estos datos para poder insertarlos en nuestra base de datos sin problemas.
- **Destination:** Destinos con nombre y actividades recomendadas.
- **Venue:** Actividad con nombre, ciudad y categoría. Datos que son devueltos al Front-End de la aplicación y que el usuario podrá visualizar.

3.2.4 Casos de uso

Podemos diferenciar cuatro casos de uso principales para nuestra aplicación, tal y como hemos ido explicando a lo largo de la memoria:

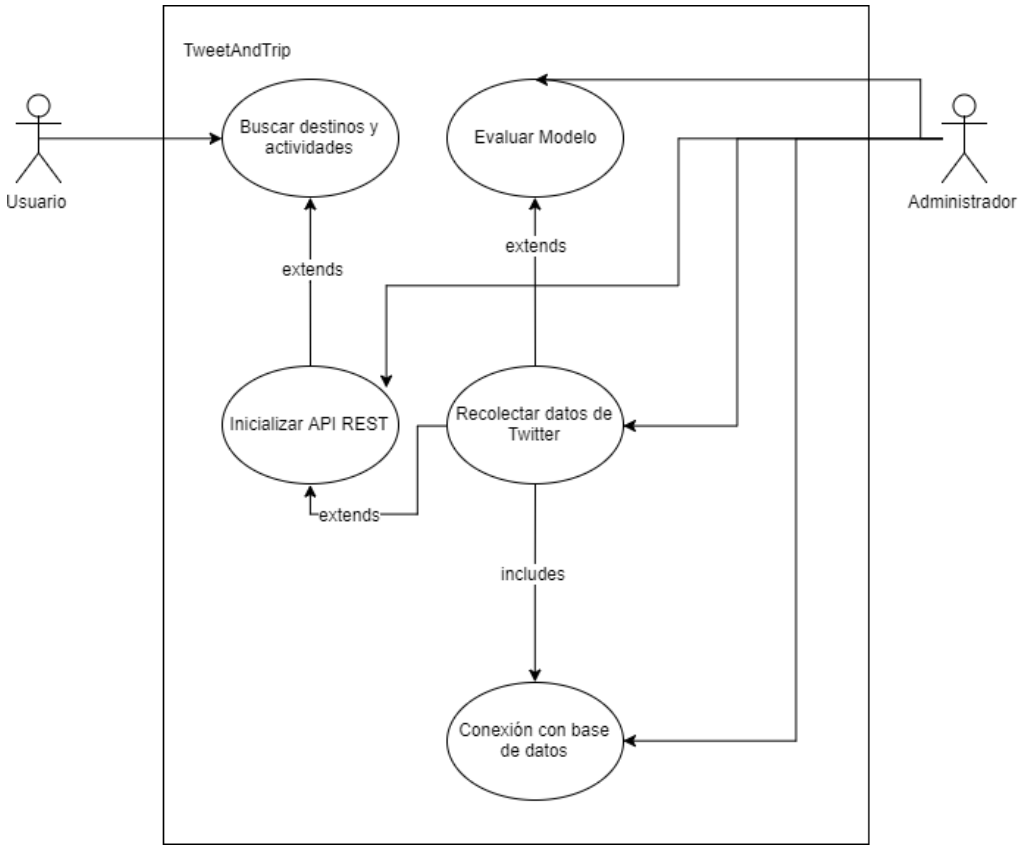


Figura 3-5: Diagrama de casos de uso de la aplicación

Como podemos observar en el diagrama, todos los casos de uso que se pueden realizar dependen de la recolección inicial de los datos de Twitter. A su vez, para poder recolectar es necesario que el Administrador levante el servidor de la base de datos.

Por otro lado, para que el Usuario pueda utilizar la aplicación web, es necesario que el Administrador haya inicializado la API REST del proyecto.

4 Desarrollo

4.1 Introducción

4.1.1 Conceptos generales

En esta sección se realizará un breve repaso de algunos conceptos que se utilizarán con frecuencia en la parte de desarrollo.

- **Cliente-Servidor:** Para la parte de la aplicación web, se ha seguido un modelo de diseño cliente-servidor, donde el cliente será nuestro front-end en Angular 5, corriendo sobre un navegador web. Este mandará peticiones HTTP a nuestro servidor JAVA levantado mediante la librería JavaSpark, y este último, realizará respuestas que el cliente recibirá.

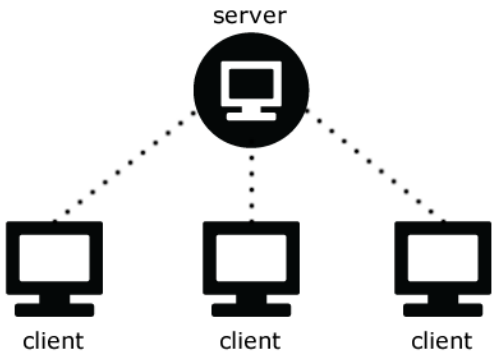


Figura 4-1: Arquitectura Cliente-Servidor

- **HTTP (Hypertext Transfer Protocol):** Protocolo de comunicación que utilizará nuestra aplicación para comunicarse. Soporta diferentes métodos entre los que encontramos GET, POST, PUT, DELETE. Cada una de estas acciones puede ser escuchada por el servidor y este último deberá responder con un código de respuesta. Dentro de los códigos de respuesta encontramos los 2XX (códigos correctos), los 4XX (error producido por el cliente) y los 5XX (error producido por el cliente).

- **API REST (Representational State Transfer):** Arquitectura de desarrollo web que se basa en HTTP, lo cual, hace que sea muy sencilla. Además, a diferencia de otras arquitecturas, no está ligada a usar un solo tipo de dato como puede ser XML, si no que permite otros que están ganando mucha importancia con el tiempo (por ejemplo, JSON). Todas las APIs que hemos utilizado y la que hemos creado se basan en este concepto.

  La idea de utilizar una API REST se basa en recursos, y no en operaciones; crear un usuario, conseguir una lista de usuarios, conseguir los tweets de un usuario, conocer las actividades para un destino, …

- **Identificación oAuth:** Tipo de identificación que utilizan algunas APIs (Twitter, FourSquare, Google Maps, …) para identificar al usuario y a la aplicación que lo utiliza. Sirve para limitar el número de llamadas que realiza una aplicación o un usuario, entre otras cosas. Normalmente es gratuito a menos que se quiera aumentar el límite de llamadas.

- **Número de llamadas de una API:** Las APIs limitan el número de llamadas que un usuario/aplicación puede realizar para evitar las sobrecargas de sus servidores.

- **Servidor de base de datos:** La base de datos requiere un servidor activo para poder atender a las consultas que se realicen desde nuestro programa Java. En el caso de MySQL, basta con abrir una consola y utilizar el comando 'mysqld'.

- **Servidor Front-End:** Para poder servir los archivos al navegador, es necesario tener levantado un servidor frontal. En nuestro caso, al utilizar Angular, nos proporciona esta opción al escribir 'ng serve' en nuestro proyecto.

4.1.2 Software utilizado

Se han utilizado una amplia variedad de programas para hacer posible este proyecto, vamos a describir brevemente los principales:

IntelliJ: IDE desarrollado por JetBrains cuya versión gratuita nos ofrece muchas posibilidades para el desarrollo en Java. Además, para los estudiantes la versión completa es gratuita, por lo que también podemos utilizarlos en el desarrollo de otros lenguajes como es el caso de nuestra aplicación web en Typescript. Su principal potencial, al menos a la hora de desarrollar en Java, es la capacidad de refactor, permitiendo prácticamente cualquier refactor que imaginemos si tenemos suficiente soltura con este programa.

Sublime Text: Editor de texto que hemos utilizado como complemento a IntelliJ. A veces no es necesario todo el potencial que te ofrece un IDE, el cual, según las circunstancias, puede llegar a ralentizar el sistema. Además, con Sublime podemos partir de una versión base a la que poco a poco podemos ir añadiendo plugins y dejarlo todo a nuestro gusto.

Git: Software de control de versiones. Consiste en crear repositorios que contengan nuestro proyecto en algún servidor de repositorios como **GitHub,** Bitbucket o GitLab**.** Una vez creado el repositorio, deberemos clonarlo en nuestro entorno personal para poder trabajar con él. En este punto normalmente se trabaja con diferentes ramas para separar entornos de producción, preproducción y desarrollo. Cuando queramos subir algún cambio, lo único que deberemos hacer es 'commit' para mantenerlo en el historial y entonces realizar 'push' hacia el repositorio remoto. Siguiendo estos sencillos pasos al final podemos obtener diferentes versiones y, en caso de desearlo, volver hacia atrás en cualquier momento.

Git Bash: Terminal principal para utilizar git, en donde, por línea de comandos, se pueden realizar todas las acciones que permite esta tecnología. También la hemos utilizado para correr servidores, programas y utilizar Angular, entre otras opciones.

HeidiSql: Software de código abierto para conexiones a MySQL, Microsoft SQL y PostgreSQL. Imprescindible programa (o similar) si se pretende realizar un proyecto que involucre una base de datos.

Weka: Software de aprendizaje automático y minería de datos. Las principales características que hemos podido encontrar en este proyecto son: la amplia variedad de clasificadores, la versatilidad que ofrece con los filtros para tratar el modelo de datos, la visualización de datos y los resultados con todas las variables que ofrece. Otro punto positivo es que permite leer tanto archivos con formato ARFF como CSV, por lo que la tarea de volcar la base de datos en un archivo parseable por Weka se hace más llevadera.

4.2 Desarrollo del proyecto

El desarrollo del proyecto se explicará en el orden cronológico en el que se implementaron las distintas fases. Desde que empezamos a investigar sobre la API de Twitter hasta que realizamos el frontal de la aplicación:

4.2.1 Recolección de datos

4.2.1.1 Elección de librería para la API de Twitter

Nuestros primeros pasos se basan en la creación del repositorio en GitHub con git. Tras crearlo, simplemente deberemos clonárnoslo mediante git clone. Gracias a git, es relativamente sencillo ver el proceso que seguimos para cada fase del proyecto.

Para la gestión de dependencias en Java hemos estado utilizando **Maven**, por tanto, con la ayuda del plugin de IntelliJ para Maven, importar una librería es relativamente sencillo. Maven se compone de un fichero pom.xml. En nuestro caso, para probar las librerías, simplemente debíamos añadirla como dependencia en este archivo y el IDE se encargaría de cargarla en nuestro repositorio local para poder utilizarla.

En este punto, nos dispusimos a probar las dos librerías que nos recomendaba Twitter para ver cuál se acercaba más a nuestro objetivo: **Twitter4J** y **HBC**. Para poder probarlas necesitamos acceder a la API mediante oAuth. Ambas librerías ofrecen distintos métodos para asignar los parámetros de autenticación.

Es aconsejable separar las contraseñas y parámetros de autenticación en otro archivo para evitar que alguien pueda robarnos las credenciales.

Entre las características de HBC, sabemos que está enfocada al streaming y recepción de eventos de Twitter. Por ejemplo, permite escuchar cuando alguien deja de seguirte o te retwittea, entre otras opciones. Por otro lado, Twitter4J está pensada para obtener datos de usuarios además de tweets con contenido específico. En conclusión, esta última cumplía todos nuestros requisitos.

4.2.1.2 Conexión a base de datos

Tras elegir la librería, fue momento de crear nuestra base de datos y de implementar una conexión Java a nuestra base de datos MySQL.

Para crear la base de datos, debemos bajarnos MySQL e instalarlo. Ejecutaremos el comando: `mysqld –initialize –console` para crear la base de datos.

Una vez hemos creado la base de datos, deberemos crear las tablas según lo hemos planificado en nuestra fase de análisis y diseño.

Es momento de realizar una conexión básica a través de Java, ya que posteriormente la utilizaremos para insertar todos los datos que vayamos recolectando. Si no hay una sesión activa, no es posible conectar con la base de datos.

Para conseguirlo, crearemos la clase **MySqlConnect** donde utilizaremos 3 métodos bastante intuitivos: getProperties (propiedades para conectar a la base de datos guardadas en un fichero seguro), connect (para iniciar la conexión con la base de datos) y disconnect (para cerrarla). Las tres clases principales que nos ayudan a que esto funcione son Properties, Connection y DriverManager.

4.2.1.3 Interacción con base de datos desde Java

Para poder mandar consultas, deberemos iniciar la conexión con la clase anteriormente creada. `MysqlConnect connection = new MysqlConnect();`

Después, utilizaremos la clase **PreparedStatement** para preparar una consulta SQL.

En caso de que la consulta nos deba devolver algo, podemos utilizar la clase **ResultSet,** en la que se almacena el retorno de la ejecución de la consulta.

4.2.1.4 Integración de los datos de Twitter en la base de datos

Establecida la base necesaria para insertar y leer de la base de datos mediante nuestro programa, es el momento de utilizar datos reales obtenidos de Twitter para rellenarla.

Mediante la clase **TwitterUtils** de nuestro proyecto, podemos realizar todas las operaciones necesarias entre la API de Twitter y nuestro programa. La primera operación necesaria es conseguir un usuario que nos interese, es decir, que haya escrito sobre ‘travel’. Tras conseguirlo, utilizaremos la clase **DatabaseUtils** para introducir este usuario en la base de datos. Sin embargo, cuando realicemos la consulta INSERT con los datos, necesitaremos hallar el género de nuestro usuario, ya que Twitter no aporta esa información.

Como alternativa a esto, hemos desarrollado una nueva llamada GET a dos APIs de género distintas: gender-api y genderize. Primero se utiliza una, y mediante una excepción, es manejada para alternar a la otra en caso de que se exceda el límite de llamadas diarias, multiplicando así la capacidad de llamadas diaria.

Una vez tenemos nuestro usuario añadido a la tabla person con todos los datos, es momento de añadir todos los hashtags de este usuario. El proceso es el siguiente: Obtenemos todos sus tweets, en cada tweet existe un atributo que nos proporciona la

librería Twitter4J, el cual es una lista de hashtags. Por tanto, obtenemos los últimos 50 hashtags que el usuario ha escrito.

Esto se ha realizado por dos motivos: En primer lugar, para no sobrecargar la base de datos y, en segundo lugar, para evitar utilizar información desactualizada del usuario; lo que escribió hace tiempo, es posible que en la actualidad no opine igual o no tenga los mismos intereses.

Por último, se realiza la inserción de los destinos que el usuario ha twitteado. Para ello, simplemente obtenemos el texto del tweet y comprobamos si alguna de las ciudades de nuestra lista se encuentra en ese texto, en caso de que así sea, se inserta un registro {personid, destinationid} en base de datos.

La lista de destinos se ha obtenido de la siguiente manera: todos los nombres en inglés de todos los países del mundo [9] a los que se les ha añadido el top 150 de ciudades más populares [6]; con esto se consigue un total de 419 destinos.

4.2.2 Construcción y evaluación del modelo

Tras la recolección de datos hay que construir el modelo. Los atributos del modelo de datos se fueron adaptando según avanzó el desarrollo, ya que surgieron problemas que explicaremos más adelante.

4.2.2.1 Construcción del modelo

Se ha seguido un proceso de volcado de la base de datos en ficheros CSV mediante consultas SQL. En este punto surgieron bastantes problemas: formar el fichero de salida correctamente, eliminar saltos de línea, tabulaciones, comillas, caracteres extraños, …

El problema principal es que Weka no informa con exactitud sobre qué está fallando en el fichero, por lo que no puede parsear el CSV y convertirlo a ARFF. Finalmente, tras una exhaustiva limpieza en los atributos, se consiguió generar un CSV que pudiera entender Weka. El modelo final, quedó de la siguiente manera:

Figura 4-2: Atributos del modelo en Weka

Como podemos observar, nuestro modelo está compuesto por los 5 datos que consideramos más significativos:

- **Localización**: Atributo de gran importancia, la residencia de la persona. No obstante, tiene un problema, y es que no es una variable discreta; el usuario puede escribir la localización que quiera.

- **Lenguaje**: Sí es un atributo fijo, en el que los usuarios escriben los tweets. Son códigos de idiomas: 'es-es', 'en-gb', 'en-eu'…

- **Género**: Obtenido a partir del nombre del usuario, por lo que, en muchos casos, el valor es NULL o indefinido. Esto se debe a que, al realizar búsquedas por la palabra 'travel', muchos de los usuarios son empresas de viaje o cuentas de países/ciudades, por lo que es normal que tengamos un alto porcentaje de 'género neutro'.

- **Tag**: En un principio también iba a ser de tipo String, como concatenación de todos los tags. Después se vio que no era una buena idea ya que no iba a mejorar el modelo. Por lo que acabó convirtiéndose en el tag que más repite el usuario en sus tweets. El razonamiento de esto es el siguiente: al final una persona que escribe mucho sobre un tema, es muy probable que le guste el mismo destino que otra persona que escribe sobre el mismo tema.

- **Destino**: Atributo de clase, ya que es el atributo que queremos predecir/sugerir al usuario.

Se llegaron a conseguir más de 180.000 registros para entrenar nuestro modelo en unas cuantas semanas de recolección de datos, incluyendo más de 5.000 usuarios únicos.

4.2.2.2 Evaluación del modelo

Para evaluar el modelo, en primera instancia, se utilizó solamente la herramienta Weka junto al clasificador Naive Bayes.

Este clasificador daba unos resultados lógicos, un acierto aparentemente bajo (cerca de un 10%) pero el cual tiene sentido si tenemos en cuenta que las posibilidades de acertar son 1/419, por lo que, si utilizásemos un clasificador aleatorio, el porcentaje de acierto debería ser cercano a 0.23%.

Sin embargo, no parecen unos resultados del todo reales, ya que lo lógico es que se tengan en cuenta las N recomendaciones más probables desde el punto de vista del clasificador, y no solamente una, al tener tal cantidad de clases.

Por tanto, para realizar este mismo proceso, pero con Top N, se vio que con la interfaz visual de Weka no era suficiente, por lo que era el momento perfecto para utilizar la librería de Java de Weka y extender el modelo de evaluación.

El proceso es muy similar, cargar nuestro modelo con los 180.000 registros, utilizar un 80% de estos registros para entrenar, un 20% como conjunto a evaluar (test) y probar con distintos valores de N. La librería de Weka es muy completa y permite aplicar cualquier tipo de clasificador de forma sencilla, así como realizar filtros y operaciones más complejas.

Cuando el clasificador termina la evaluación, nos devuelve un array sin ordenar de probabilidades asociadas a cada clase para cada instancia que se quiera clasificar. Cada índice de este array, corresponde a una clase. Por tanto, obtenemos los índices de las N clases con mayor probabilidad. Una vez tenemos estos índices, mediante Weka, podemos obtener qué clases son y ver, por fin, si alguna de ellas coincide con la que debería haber clasificado inicialmente (si aparece en el test del usuario). En caso de que exista una coincidencia, lo tomaremos como un acierto del clasificador.

Este proceso se repite para cada una de las instancias del 20% seleccionado para test.

4.2.3 Implementación de una API REST

Finalmente, con la parte principal del programa realizada, faltaba darle algún uso. Aquí es donde empieza el desarrollo de la API REST con el objetivo de que la aplicación web, posteriormente, consuma los servicios y recursos que ofrece.

Para la implementación de API REST se buscó información y actualmente encontramos dos formas rápidas y sencillas: Spring y JavaSpark.

Finalmente, la forma más sencilla y que conseguimos que funcionara antes fue JavaSpark. Spring parecía contar con un proceso más complejo y con más dependencias que JavaSpark. Cuando se intentaron importar ciertas clases de este proyecto, parecía que no las resolvía correctamente. Por lo tanto, como recomendación personal, por su sencillez de instalación, volvería a utilizar JavaSpark. La sintaxis que sigue es muy simple:

```
get("/destination", (req, res) -> {
    res.type("application/json");
    String name = req.queryParams("name");
    String jsonInString;
    try {
        ArrayList<Destination> destinations = recommendDestination(name);
        Gson gson = new Gson();
        jsonInString = gson.toJson(destinations);
    } catch (TwitterException e){
        …
    } catch (Exception e) {
        …
    }
    return jsonInString;
});
```

Para devolver un JSON como respuesta, simplemente utilizamos el objeto GSON que convierte un objeto de Java en un JSON.

Como podemos observar, podemos sacarle provecho a Java 8 con la función lambda que utilizamos para nuestra petición (request) y la respuesta (response).

La lógica es sencilla, ya que reutilizamos casi todo de la fase de evaluación; al haber dividido todo en métodos más pequeños, en nuestro nuevo método recommendDestination ha sido simplemente coger las piezas que necesitábamos para construir un array de destinos con actividades para recomendar.

4.2.4 Aplicación Web

Para la aplicación web se ha utilizado Angular 5, un framework de Typescript basado en componentes web. Se ha seguido un control de versiones con Git al igual que con la parte Java, pero con un repositorio nuevo.

4.2.4.1 Creación de una aplicación en Angular 5

El comienzo para crear una aplicación con Angular 5 es sencillo gracias a Angular CLI, el cual nos permite, mediante línea de comandos, ejecutar las sentencias necesarias para inicializar nuestra aplicación.

Antes de comenzar a crear la aplicación, deberemos tener al menos instalados npm y node con las versiones 3.x.x y 6.9.x como mínimo respectivamente.

A continuación, abriremos una línea de comandos e instalaremos globalmente (-g) en nuestro sistema Angular CLI:

```
npm install -g @angular/cli
```

El siguiente paso será crear nuestra aplicación en el directorio que queramos:

```
ng new tweet-and-trip-client
```

Por último, deberemos iniciar el servidor que se encargará de servir los archivos necesarios a nuestro navegador:

```
cd tweet-and-trip-client
```

```
ng serve -open
```

Si accedemos a http://localhost:4200/ podremos comprobar que la aplicación se ha inicializado correctamente.

4.2.4.2 Librerías de Componentes: Bootstrap y Angular Material

Para nuestra aplicación, hemos elegido los componentes que nos aportan tanto Bootstrap como Angular Material. Para ambos:

```
npm install bootstrap -save
```

```
npm install --save @angular/material @angular/cdk
```

En el caso de Bootstrap, bastará con importar en nuestros estilos el archivo en node_modules mediante el comando anterior:

```
@import '~bootstrap/dist/css/bootstrap.min.css';
```

En el caso de Angular Material, bastará con importar en nuestros archivos los módulos y componentes necesarios que vayamos a utilizar.

Además, en este caso, va acompañado de un tema de estilos, el cual podemos importar modificando simplemente nuestro archivo principal de estilos (styles.css):

```
@import '~@angular/material/prebuilt-themes/indigo-pink.css';
```

4.2.4.3 Componentes

Una vez hemos conseguido establecer una aplicación básica, con estilos y librerías extra, es hora de desarrollar nuestros propios componentes.

Angular 5 es un framework basado en Web Components. Podemos diferenciar dos tipos de elementos dentro de esta tecnología: Módulos y Componentes.

Los componentes manejan las vistas HTML, mientras que los módulos se encargan de agrupar distintos componentes, así como de sus declaraciones y dependencias, de forma que los módulos dictan qué componentes dependen de otros.

Para crear un componente, gracias a Angular CLI podemos hacerlo de una forma muy sencilla:

```
ng generate component home
```

Esto generará una carpeta home con un archivo HTML, uno CSS, uno TypeScript y uno de test. A su vez, declarará en nuestro módulo principal (app.module.ts) este nuevo componente.

De esta forma ya estamos listos para poder utilizar nuestro componente: Si vamos a nuestro archivo app.home.ts, veremos que, al declarar el componente, también se declara el selector:

```
selector: 'app-home',
```

Este selector sirve para que, en otros archivos HTML, podamos utilizarlo como etiqueta:

```
<app-home></app-home>
```

Lo cual pintará el archivo app.home.html.

A partir de aquí, todo se basa en ir creando componentes con la funcionalidad que queramos. En nuestro caso, hemos creado los siguientes componentes:

- **Home**: Página de inicio compuesta de una imagen, un input y un botón. Con las validaciones de texto necesarias para evitar que el usuario introduzca texto vacío.

- **List**: Lista en forma de tabla con los resultados. También contiene un spinner para mostrar la carga al usuario hasta que finalice la llamada HTTP.

- **Header**: Componente siempre visible en la página y por tanto añadido al index.html directamente. Fijo en la parte superior con el propósito de ser la cabecera junto con tres botones que redirigen al home, a la página de Twitter de TweetAndTrip y a los repositorios en GitHub de TweetAndTrip.

- **Errorcard**: Vista que muestra el mensaje de error en caso de que la llamada HTTP no devuelva un código 200 (OK).

4.2.4.4 Routing

Para realizar la navegación dentro de nuestra página, Angular 5 nos ofrece la posibilidad de utilizar RouterModule.

Este módulo deberá ser importado en el archivo app.module.ts de la siguiente forma:

```
imports: [RouterModule.forRoot(appRoutes)]
```

Donde appRoutes será nuestra variable que contenga la relación entre ruta y componente:

```
const appRoutes: Routes = [{
  path: '',
  component: AppHomeComponent
}, {
  path: 'results/:name',
  component: ListComponent
}];
```

Por último, allá donde queramos instanciar este router, deberemos escribir en el html lo siguiente:

```
<app-root></app-root>
```

Por tanto, si nos fijamos en el ejemplo anterior, si vamos a la ruta /, pintará el componente home, en caso de ir a results/:name, pintará el componente list.

4.2.4.5 Llamada HTTP

El último paso importante dentro del desarrollo de nuestra aplicación web ha sido el de realizar la llamada HTTP a nuestra API. Para conseguirlo, importaremos el módulo (como ya hicimos con el módulo Router) HTTPClientModule en el archivo app.module.ts.

```
imports: [HttpClientModule]
```

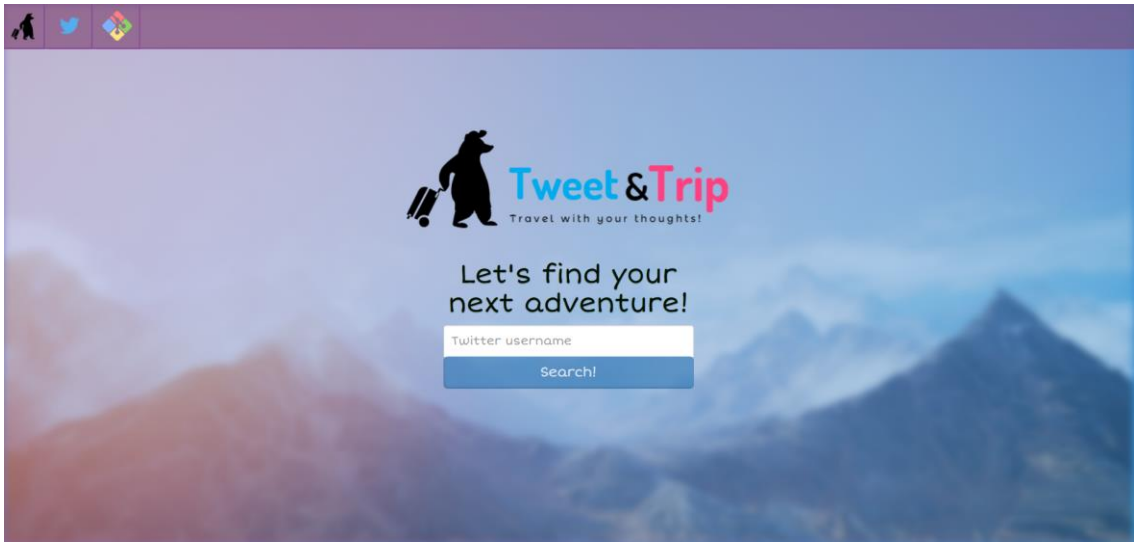
Gracias a esto, podremos utilizar en nuestro componente la funcionalidad de HTTPClient, de la siguiente manera:

```
this.http.get('http://localhost:4567/destination?name=' +
params['name']).subscribe(data => {
   //Lógica de Success
  },
  err => {
    //Lógica de Error
  }
);
```

4.2.4.6 Resultado final

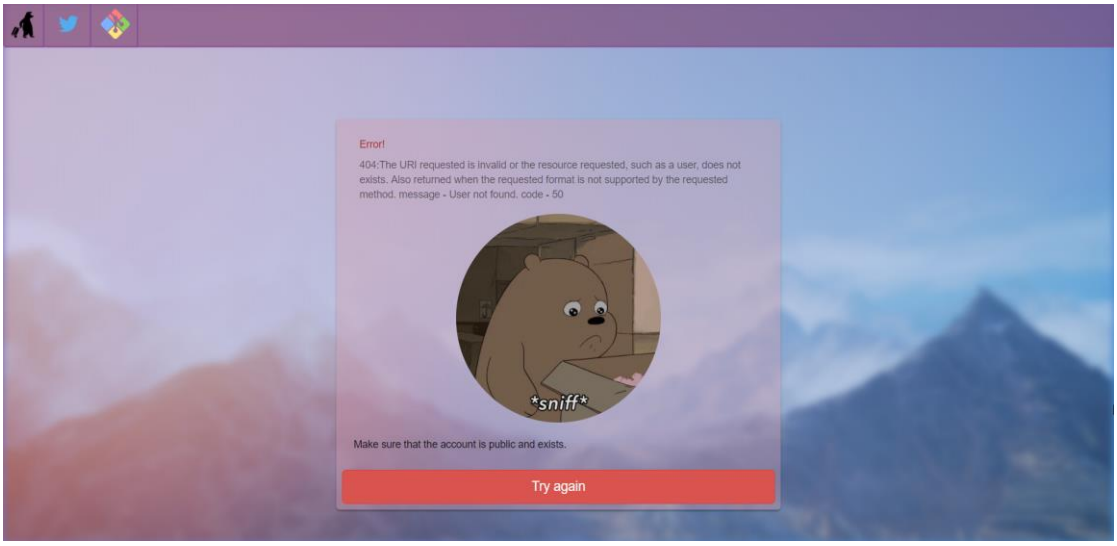
Finalmente, uniendo todos estos procesos, conseguimos crear la aplicación web en Angular 5 de TweetAndTrip:

Página Home:

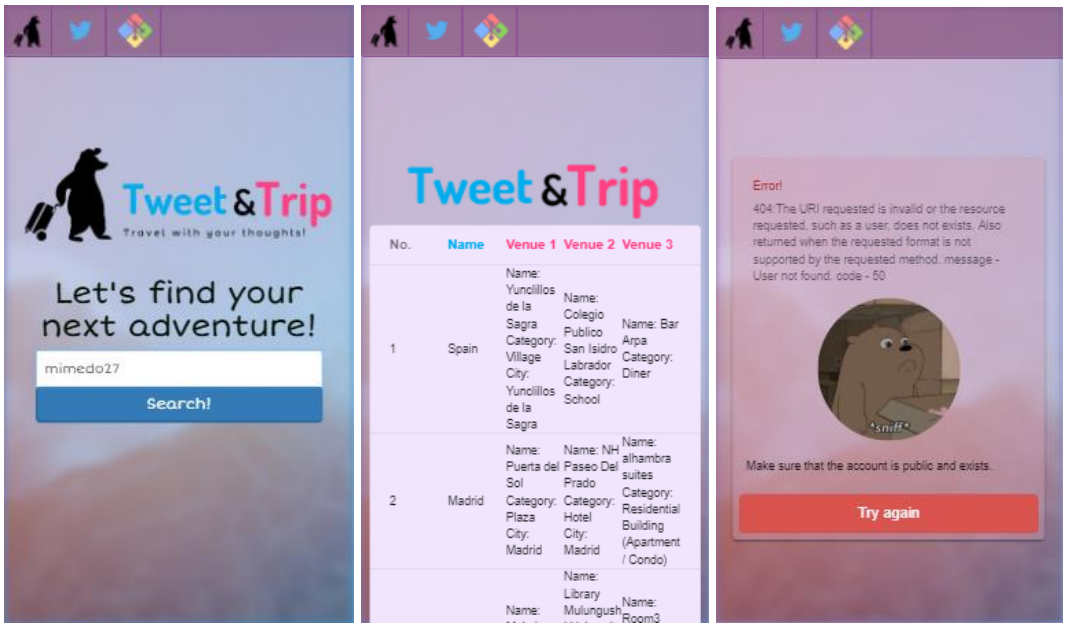


Página de resultados:

Página de error:



También se ha adaptado a versión móvil y tablet:



4.3 Problemas encontrados

En esta sección vamos a intentar hacer un pequeño resumen de los problemas que nos hemos ido encontrando a lo largo del proyecto:

4.3.1 Límite de memoria en Windows al entrenar modelos con Weka

Al principio de utilizar Weka, el modelo contenía muchos más datos, como la descripción o los tags en forma de string, por lo que el entrenamiento era mucho más costoso. De hecho, era tan costoso que el sistema acababa fallando por falta de memoria. La solución a este problema pasa por aumentar la memoria máxima a la que puede acceder el programa.

Para aumentarla podemos cambiar la variable de entorno JAVA_OPTS = -Xmx2000m, podemos cambiar el archivo RunWeka.ini añadiendo la línea 'maxheap=2000m' o ejecutar la siguiente línea: java -Xmx2000m -jar weka.jar en una terminal.

4.3.2 Eliminación de la descripción y versión single-tag del modelo

Al principio, el modelo contaba con tres atributos de tipo String (la descripción, la localización y los tags), lo cual, como hemos comentado, es demasiado costoso para entrenar, ya que habría que realizar todo tipo de combinaciones entre esos valores y el programa nunca terminaría. Se llegaron a hacer pruebas y después de más de 30 minutos Weka no daba ninguna respuesta. Por tanto, para aligerar el modelo, se decidió quitar la descripción ya que para que fuera realmente útil habría que haber sacado palabras clave mediante algún método como TFIDF.

Por otro lado, se decidió, como se ha explicado anteriormente, que, en vez de tener una lista de tags, se tuviera solamente el tag más utilizado por la persona.

Con estos dos cambios y manteniendo la localización, hemos conseguido disminuir la carga de trabajo para que Weka pudiera entrenar el modelo correctamente.

4.3.3 Volcado de base de datos en CSV, eliminar espacios, \n, \t, caracteres especiales, etc

Uno de los problemas principales ha sido intentar cargar el CSV generado de Base de Datos en Weka. Los errores informan de que el fichero está malformado, que en algunas líneas que normalmente no coinciden con las líneas que realmente causan el problema, existen un parseo de datos incorrecto.

Al final lo más sencillo es realizar un preprocesado de los datos. Una vez hemos recopilado e insertado los datos en nuestra base de datos, se les han realizado operaciones para conseguir datos legibles por Weka.

No se recomienda intentar arreglar las líneas conflictivas en ficheros tan grandes, es muy probable que, si falla una, sean más las que estén mal. En nuestro caso, la solución que se planteó fue ejecutar, sobre la base de datos los siguientes comandos, entre otros:

```
UPDATE person_tag set `tag` = REPLACE(`tag`,'\r', '');
UPDATE person_tag set `tag` = REPLACE(`tag`,'\n', '');
UPDATE person_tag set `tag` = REPLACE(`tag`,',', '');
UPDATE person_tag set `tag` = REPLACE(`tag`,'"', '');
UPDATE person_tag set `tag` = REPLACE(`tag`,'\'', '');
```

4.3.4 Género y edad en Twitter

Como ya se ha comentado brevemente, existió el problema de que no podíamos contar ni con el género ni con la edad del usuario de Twitter. Por lo que el género se extrapoló del nombre mediante dos APIs externas y, dentro de un margen de error, se consiguió seguir contando con este campo.

Por otro lado, la edad tampoco la devolvía Twitter. Investigando diferentes soluciones, no parecía que hubiera algo tan sencillo como llamar a una API con un nombre y que te devolviera la edad. Al final la edad no es un valor binario como el género, determinar la edad es mucho más complicado. Una solución que se planteaba en distintos foros es utilizar los datos que tenemos, como la descripción, los followers, los retweets, para agrupar a los usuarios en un rango de edad. Debido a que parecía una tarea complicada, se dejó el campo edad de base de datos como 0, por si en un futuro se quisiera implementar esta funcionalidad.

5 Integración, pruebas y resultados

5.1 Introducción

Durante las pruebas deberemos comprobar si nuestro glosario de requisitos diseñado en la fase de análisis se está cumpliendo. Básicamente veremos si el programa cumple su función como esperamos. Se han dividido las pruebas realizadas en los siguientes subtipos:

5.1.1 Pruebas unitarias

Es el tipo de prueba más básico que existe, se realiza a nivel de programación y consiste en comprobar que una función hace lo que debería realizar. En nuestro caso, al haber utilizado Java, deberemos comprobar que cada método responde tal y como esperamos.

Para que las pruebas sean correctas, los métodos deberán ser completamente agnósticos al resto de componentes; si existe una conexión con base de datos, esta debería poder ser sustituida para que no interfiera el estado real de la base de datos en la prueba. De este modo, sabremos que la prueba se realiza exactamente como queremos. En Java, para realizar este tipo de pruebas, se suele utilizar jUnit para realizar los tests y poder correrlos y Mockito o PowerMock para aislar las funciones de la forma que comentábamos. Por ejemplo, aplicando 'mocks' a la conexión a base de datos que comentábamos.

De esta forma, estamos probando exactamente la lógica que este método contiene, y no la lógica de métodos que llama internamente. Además, también existe la ventaja de que estas pruebas serán las más rápidas al no necesitar llamar a componentes externos.

En nuestro caso, las pruebas se han realizado durante el desarrollo del proyecto de forma manual. Un añadido sería implementar estas pruebas de forma independiente para poder correrlas cada vez que añadamos una nueva pieza a nuestro sistema.

Podemos encontrar ejemplos de este proceso en toda la aplicación, allá donde encontremos un método testeable. Por ejemplo, a la hora de añadir actividades a nuestro destino: no deberemos comunicarnos con FourSquare para obtener esas actividades, si no que las tendremos previamente. De esta forma, 'mockearemos' las actividades, serán asignadas a nuestro destino y después comprobamos si se han asignado correctamente.

5.1.2 Pruebas de integración

Con este tipo de pruebas damos un paso más y comprobamos que la integración entre dos partes es correcta. Aquí no necesitaríamos 'mockear' la base de datos, ya que lo que queremos comprobar es que la conexión se ha establecido correctamente entre nuestro programa y el servidor de base de datos. En este punto también podemos utilizar jUnit en Java, pero sin utilizar el soporte de Mockito o PowerMock.

Hemos seguido un proceso muy similar al de las pruebas unitarias, es decir, pruebas manuales durante el desarrollo del proyecto.

Un ejemplo de esto es la conexión con las distintas APIs, se probó a introducir distintos tipos de credenciales para comprobar si la conexión se estaba realizando correctamente.

Surgió la duda de si realmente era necesaria esta autenticación y, efectivamente, utilizando otras credenciales las APIs nos deniegan el acceso.

5.1.3 Pruebas de sistema

En este apartado se realizarán todas las pruebas con el comportamiento del sistema completo, para ello, dividimos las pruebas en los siguientes tipos:

- **Pruebas funcionales:** Son las pruebas que indican si nuestros requisitos funcionales se cumplen. Son pruebas de caja negra, ya que, para una entrada en el sistema, esperamos un resultado. Por ejemplo, para el RF6 (evaluación de un modelo), comprobamos que si pasamos un N = 5 a nuestro programa:

```
Please, select option:
Option   Description
1        Collect more data from Twitter (Database server must be running)
2        Evaluate top N
3        Run API
4        Exit
2
Select N (1-350)
5
Loading... Please, be patient!
```

Figura 5-1: Output del programa Java

Nos dará resultados en base a ese N (5):

```
True positive at 5: 0.2759544726227968
```

Figura 5-2: Output de los resultados para N=5

- **Pruebas de estrés:** Pruebas cuyo propósito es el de llevar al extremo ciertas funcionalidades del programa. Por ejemplo, podríamos comprobar qué sucede si intentamos introducir una N = 0 estando fuera del rango permitido:

```
Please, select option:
Option   Description
1        Collect more data from Twitter (Database server must be running)
2        Evaluate top N
3        Run API
4        Exit
2
Select N (1-350)
0
N must be between 1-350
Please, select option:
Option   Description
1        Collect more data from Twitter (Database server must be running)
2        Evaluate top N
3        Run API
4        Exit
```

Figura 5-3: Output para un N fuera de rango

- **Pruebas de carga:** Son las pruebas que se encargan de comprobar la capacidad del sistema. En nuestro programa, se sabe (gracias a realizar pruebas de carga), que existe una limitación con el número de llamadas diaria debido a las restricciones de

las APIs. Debido a esto, se implementó una API alternativa de género. También, en caso de que no pueda conectar con algunas APIs, se ha intentado que el programa informe y no pare para poder seguir con el proceso final. Por ejemplo, en caso de que FourSquare no encuentre actividades para un destino, el proceso continúa solo que al usuario final no le llegarán actividades en ese caso.

5.2 Resultados

Como resultados de la recolección de datos, podemos considerar lo siguiente:

Más de 180.000 registros, con datos de más de 5.000 usuarios únicos y 355 destinos encontrados dentro de los 419 disponibles en nuestra tabla Destinations.

Para nuestra evaluación hemos utilizado **Naive Bayes** y para construir el modelo a través de Weka está tardando una media de 17 segundos.

Entre las pruebas realizadas con Weka, hemos realizado una con validación cruzada con 10 particiones, obteniendo los siguientes resultados:

```
Time taken to build model: 17.06 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances       16444               9.0856 %
Incorrectly Classified Instances    164546              90.9144 %
Kappa statistic                          0.0857
Mean absolute error                      0.0054
Root mean squared error                  0.053
Relative absolute error                 97.9769 %
Root relative squared error            100.5202 %
Total Number of Instances           180990
```

Figura 5-4: Resultados en Weka para clasificación para 10 particiones

Con los datos de algunas de las clases:

```
=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
               0,052     0,001     0,227       0,052   0,085      0,105    0,599     0,032     Thailand
               0,062     0,000     0,313       0,062   0,103      0,138    0,695     0,046     Seoul
               0,000     0,000     0,000       0,000   0,000      -0,000   0,594     0,005     Moscow
               0,061     0,002     0,166       0,061   0,089      0,098    0,538     0,021     Las Vegas
```

Figura 5-5: Detalles de resultados para 10 particiones

Resultando en un 9% de acierto. Valor que a priori puede parecer algo bajo por diferentes razones que veremos más adelante.

Otra prueba similar, realizada con una partición por porcentaje del 80% en entrenamiento y 20% en test, obteniendo unos resultados similares:

```
Time taken to test model on test split: 23.34 seconds

=== Summary ===

Correctly Classified Instances        3124               8.6303 %
Incorrectly Classified Instances     33074              91.3697 %
Kappa statistic                          0.0811
Mean absolute error                      0.0054
Root mean squared error                  0.053
Relative absolute error                 97.9668 %
Root relative squared error            100.5395 %
Total Number of Instances            36198
```

Figura 5-6: Resultados en Weka para clasificación por porcentaje 80%

```
=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
               0,030     0,001     0,300       0,030   0,055      0,093    0,612     0,038     Thailand
               0,058     0,000     0,455       0,058   0,103      0,162    0,717     0,074     Seoul
               0,000     0,000     0,000       0,000   0,000      0,000    0,588     0,005     Moscow
               0,059     0,001     0,214       0,059   0,093      0,110    0,500     0,024     Las Vegas
```

Figura 5-7: Detalles de resultados para porcentaje 80%

Resultando en un 8.63% de acierto.

Estos resultados, a priori, pueden parecer algo bajos. Sin embargo, deberemos tener en cuenta que existen más de 350 posibles clases para recomendar. Por lo que no es un resultado demasiado justo para nuestro modelo.

Por lo tanto, la siguiente prueba que se realizó fue coger las N clases con mayor probabilidad, y ver si se encontraba entre la clase de test que debería ser, esto, como ya se ha explicado, se realizó a través de Java con la librería de Weka y no a través del cliente de Weka como en las pruebas anteriores.

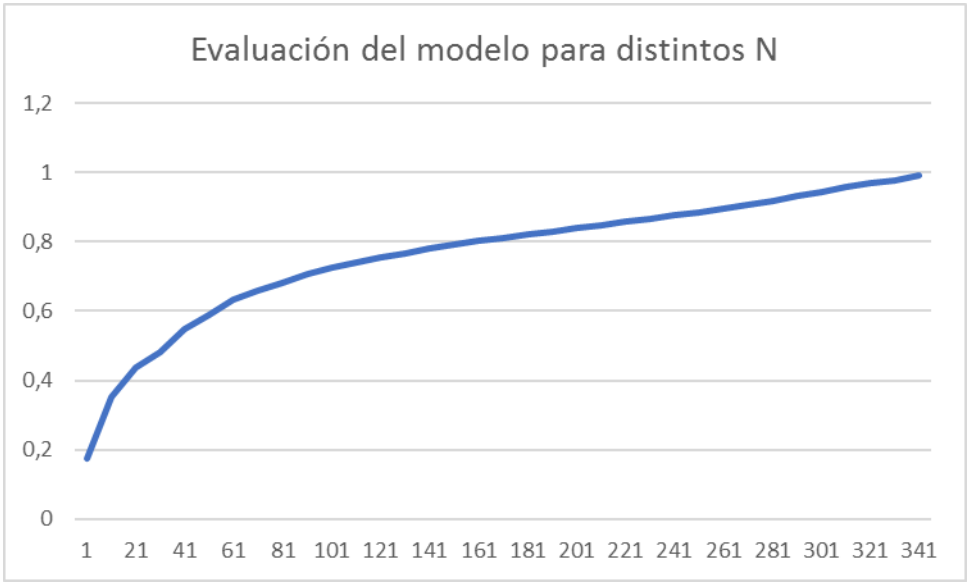


Figura 5-8: Evaluación del modelo con distintas N

En la gráfica anterior, podemos observar cómo para distintos N (1-350), vamos obteniendo un acierto mayor. Lo cual tiene sentido ya que, si usásemos una N con las 419 clases existentes, en el conjunto resultante, siempre estaría nuestra clase clasificada, dando un acierto del 100%.

Estos resultados son más satisfactorios, ya que para un N=10 obtenemos un acierto de más del 30%, lo cual le empieza a dar fiabilidad a nuestro proceso. Esto significa que, si el usuario recibiera 10 recomendaciones de nuestro sistema, habría una probabilidad del 30% de que alguna de ellas fuera relevante para él.

Esta forma de clasificación fue útil para entender que un modelo con tal variedad de clases no es realista recomendar solamente una entre las 419. Más cuando en nuestra aplicación encaja correctamente el concepto de que al usuario recomendado le puedan interesar varios destinos y no solo uno.

Por último, se han realizado pruebas con usuarios particulares con distintas características (a continuación, algunas de ellas):

Nombre	Destino 1	Destino 2	Destino 3	Destino 4	Destino 5
Neymarjr	Mexico	London	Japan	Italy	Canada
Mimedo27	Spain	Madrid	Zambia	Italy	Belize
Mohammed	Pakistan	Cambridge	Dubai	Barbados	Vietnam
Elonmusk	Japan	India	New Delhi	York	Houston
Rafaelnadal	Norfolk Island	Dalian	St. Petersburg	Zhuhai	Guinea-Bisseau
Wismichu	Japan	London	India	Russia	Mexico
Narendramodi	India	Pakistan	San Jose	Bhutan	London
Davidguetta	Norfolk Island	Dalian	St. Petersburg	Zhuhai	Guinea-Bisseau
Germangarmendia	Norfolk Island	Dalian	St. Petersburg	Zhuhai	Guinea-Bisseau
Consuelosaav	Chile	Israel	Bolivia	Glasgow	Norfolk Island

Tabla 5-1: Resultados para usuarios concretos

Dentro de estos resultados, podemos encontrar varias conclusiones:

- En verde, indicamos las recomendaciones que tienen alguna relación con el usuario (como país de origen). En rojo, encontramos resultados que se repiten sin seguir ninguna relación con el usuario. El resto de campos son resultados neutros que podrían tener relación o no con el usuario pero que no aparecen con frecuencia.

- Existe una clara desviación hacia el conjunto:

Norfolk Island	Dalian	St. Petersburg	Zhuhai	Guinea-Bisseau

Esto nos indica que existe algún error en el modelo, o al menos, que es muy mejorable. Probablemente analizando y construyendo el modelo de otra forma y limpiando algunos de los campos, podríamos evitar problemas de este tipo.

- El conjunto de Spain, Madrid, etc, se repite con frecuencia, también podría solucionarse con lo anteriormente descrito.
- Para usuarios árabes recomienda países musulmanes, lo cual, independientemente de que al usuario le interese, indica que sí está tomando en cuenta ciertos datos el modelo.
- Al usuario Wismichu se le recomienda Japón en primer lugar, lo cual probablemente esté relacionado con un viaje reciente a Japón que menciona en su Twitter.
- Al usuario indio Narendramodi se le recomienda India y su frontera, Pakistán, lo cual indica un buen acierto geográfico de los usuarios.
- A la usuaria chilena Consuelosaav se le recomienda Chile y Bolivia, lo cual encaja geográficamente, pero aporta poca diversidad y novedad, un problema que no se ha abordado en este trabajo y que es bastante importante (y difícil de resolver) en el área de los sistemas de recomendación.
- La usuaria mimedo27, de Madrid, recibe España y Madrid como recomendaciones, además de Italia, el cual, según sus palabras, es el próximo destino al que quiere ir.

Como conclusión, podemos ver que el recomendador tiene ligeras ideas sobre lo que realmente el usuario podría llegar a querer. Hay que tener en cuenta que estos resultados son con N=5, donde vimos que el acierto era de un 20-30%. Este acierto podría llegar a incrementarse con un número de usuarios superior a la hora de entrenar el modelo (5.000 es una cifra baja teniendo en cuenta que existen más de 400 clases) y con un refinamiento de los atributos del modelo, aspectos que se intentarán estudiar en el futuro.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Después de la realización de todo el proyecto, se pueden sacar conclusiones en prácticamente todas las fases del mismo.

En la parte de análisis se ha podido comprobar que es muy importante realizar un amplio estudio de todos los caminos que se pueden tomar, así como de todas las dudas y problemas que puedan aparecer. A pesar de que el análisis ha sido bastante extenso, se podrían haber evitado problemas como los comentados en la parte de desarrollo.

Aun así, gracias a esta parte, hemos podido ver el estado actual de aplicaciones similares, y es que no da la sensación de que existan tantas aplicaciones que exploten todos los recursos que ofrecen las APIs de las redes sociales, más que pequeñas aplicaciones, proyectos de estudiantes y, como suposición, a nivel privado en empresas.

También, gracias al desarrollo del proyecto se ha podido ver de forma directa la importancia que tienen determinados atributos (así como un procesamiento adecuado) en un modelo de recomendación y minería de datos.

Viendo los resultados podemos concluir que la elección del modelo es más que suficiente, pero con margen de mejora. Probablemente con unos pequeños cambios y con un dataset mayor la mejoría puede ser muy notable.

Para terminar, como opinión personal, decir que la realización de este proyecto ha sido muy satisfactoria. Realmente llena de satisfacción ver que se pueden plasmar muchos de los conocimientos adquiridos durante la carrera: diseño de software, diseño de proyectos, aprendizaje automático, bases de datos, redes, etc y terminar haciendo una aplicación end-to-end con todas las partes que eso implica. Tener la sensación de haber creado algo prácticamente desde cero es algo muy reconfortante.

6.2 Trabajo futuro

Al ser un proyecto que abarca tantos campos, es muy fácil pensar en posibles mejorar para implementar en nuestro proyecto, aquí van unas pocas:

- **TFIDF**: Este tipo de procesamiento podría haber resultado muy útil en la parte de preprocesado de la información, antes de ser insertada en la base de datos. Se podrían procesar mucho mejor algunos campos como la descripción del usuario, consiguiendo así atributos en nuestro modelo mucho más relevantes.

- **Incremento del modelo**: Como ya se ha comentado en varios puntos de la memoria, sería interesante aumentar el número de usuarios que tenemos en nuestra base de datos.

- **Aplicación móvil**: A pesar de contar con versión web, la cual es accesible desde el navegador del móvil, habría sido muy interesante implementar una aplicación móvil. Tal y como vimos en el estado del arte del proyecto, cada vez se escribe

más a través de móvil en el ámbito de redes sociales, por lo que una aplicación así encajaría mucho más en contexto móvil que en web.

- **Scrum**: A pesar de haber aplicado un modelo iterativo e incremental de entregas tal y como propone scrum, para siguientes proyectos se podría valorar la implementación de las reuniones típicas de scrum, donde en cada Sprint se realicen refinamientos de las historias de usuario que vayan entrando. De esta forma, el análisis es mucho más efectivo y el resultado acaba siendo mejor.

- **Testing**: Un punto muy importante dentro del desarrollo de código es realizar unos buenos tests. Habría sido muy positivo incluir jUnit con Mockito más sistemáticamente en todo el proyecto en vez de simplemente realizar tests manuales. Gracias a unos buenos tests se consigue una buena consistencia, lo cual implica que nuevos desarrollos no van a poder romper el código antiguo.

- **Documentación**: Se ha documentado todo el proyecto a nivel de código. Sin embargo, ya que se ha implementado una API, tener al menos una documentación de esta que su uso sea más sencillo. Aun así, contiene un fichero README en formato markdown que explica las posibilidades de nuestro programa.

Referencias

[1] "List of most followed Twitter accounts", Wikipedia, 7 Enero 2018, https://en.wikipedia.org/wiki/List_of_most-followed_Twitter_accounts

[2] "Modern alternatives to Weka", Quora, Kazem Jahanbakhsh, Junio 2016, https://www.quora.com/What-are-the-modern-alternatives-to-the-WEKA-machine-learning-library

[3] "Most famous social network sites worldwide", Statista, 2017, https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/

[4] "Number of monthly active Twitter users worldwide", Statista, 2017, https://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/

[5] "SQL vs noSQL ¿Cuál debo usar?", Alida Vergara, Facilcloud, 2016, https://www.facilcloud.com/noticias/sql-vs-nosql-which-one-should-i-use/

[6] "Top 150 Destinations: London Leads the Way", Euromonitor International, 11 Octubre 2007 https://blog.euromonitor.com/2007/10/top-150-city-destinations-london-leads-the-way.html

[7] "Top 20 Valuable Facebook Statistics", Zephoria, Enero 2018, https://zephoria.com/top-15-valuable-facebook-statistics/

[8] "Twitter by the Numbers", Omnicoreagency, Enero 2018, https://www.omnicoreagency.com/twitter-statistics/

[9] "World Cities Database", Geodatasource, Octubre 2017, http://www.geodatasource.com/world-cities-database/free

Glosario

API	Application Programming Interface
ARFF	Attribute-Relation File Format
Back End	Capa de acceso a datos de nuestra aplicación
CSV	Comma-Separated Values
End to end	Relativo a todas las capas de la aplicación
Foursquare	Red social auxiliar para conseguir actividades
Front End	Capa de presentación de nuestra aplicación
Mock	Objeto de prueba
REST	Representational State Transfer
SPA	Single Page Application
Tweet	Mensaje escrito en Twitter por un usuario
TweetAndTrip	Nombre de la aplicación del proyecto
Twitter	Red social principal utilizada en nuestra aplicación

Anexos

A Manual de uso

TweetAndTrip es una aplicación basada en recomendar viajes con la información de los usuarios de Twitter. Para utilizarla tenemos dos opciones, descargarnos el .jar generado y ejecutarlo en nuestro proyecto:

https://www.dropbox.com/s/wsa6dwtf18y94xc/TweetAndTrip-1.0-SNAPSHOT-jar-with-dependencies.jar?dl=0

O, por otro lado, tenemos la opción de clonar el proyecto mediante git que podemos encontrar en el siguiente enlace:

https://github.com/JorobomGit/TweetAndTrip

Una vez tenemos clonado el proyecto, deberemos tener un sistema con Java 1.8. En el siguiente link podemos encontrar la JDK necesaria:

http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

Tras esto, compilaremos y ejecutaremos nuestro programa, el cual nos mostrará el siguiente output:

```
1 - Collect more data from Twitter (Database server must be running)*
2 - Evaluate top N
3 - Run API**
4 – Exit
```

Para poder ejecutar la primera opción, deberemos tener iniciado nuestro servidor mysql mediante el comando ‘mysqld –console’.

En el caso de la segunda opción, iniciar la API, simplemente bastará con ejecutarla y acceder mediante un navegador a la ruta:

http://localhost:4567/destination?name=NOMBRE

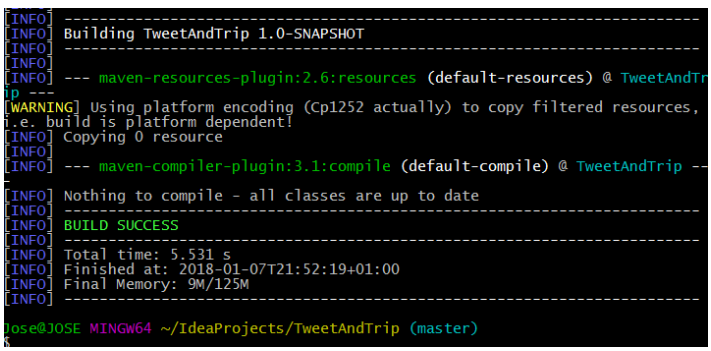
Donde nombre será el usuario al que queremos recomendarle un destino.

Por último, ya que tenemos Maven integrado en nuestro proyecto, tenemos la posibilidad de crear una nueva versión SNAPSHOT del código extrayéndolo así a un .jar ejecutable.

Para generarla, haremos lo siguiente desde una terminal en la carpeta del proyecto:

`mvn compile` y, a continuación,

`mvn package`



Lo cual construirá una nueva versión .jar en target.

Para poder utilizar la parte front del proyecto, clonaremos el repositorio de GitHub:

https://github.com/JorobomGit/TweetAndTripClient

Ahora deberemos ir a la raíz del proyecto y lanzar el servidor mediante ng serve: