Institut Puig Castellar
Santa Coloma de Gramenet

Análisis de datos con Apache Cassandra y Python

MEMORIA

CFGS DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Enrique Villarreal
Adrián Asensio

Curso **2016-2017**
1 de junio de 2017

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada 3.0 España de Creative Commons

Resumen

Hoy en día, dos de los conceptos más populares en la industria son el *big data* y la ciencia de los datos. En éste proyecto, nos mojaremos los pies un poco en ambos conceptos, apoyándonos del SGBD NoSQL Apache Cassandra y Python para realizar un breve análisis de sentimiento de twits recogidos mediante la API de Twitter.

Palabras clave: big data, data science, nosql, cassandra, python, twitter

Abstract

These days, two of the most popular buzzwords in IT are big data, and data science. In this project, we will get our feet wet in both these concepts, using the NoSQL DBMS Apache Cassandra and Python to perform sentiment analysis on tweets collected by using the Twitter API.

Keywords: big data, data science, nosql, cassandra, python, twitter

Índice

1. Introducción **1**
- 1.1. Contexto y justificación . . . 1
- 1.2. Objetivos . . . 1
- 1.3. Metodología . . . 1
- 1.4. Características técnicas . . . 1
- 1.5. Planificación del proyecto . . . 2
  - 1.5.1. Lista de tareas . . . 2
  - 1.5.2. Diagrama Gantt . . . 3
- 1.6. Descripción de los capítulos . . . 4
- 1.7. Formato de éste documento . . . 5

2. Estado del arte: bases de datos NoSQL **6**
- 2.1. Introducción . . . 6
- 2.2. Tipos de bases de datos NoSQL . . . 6
  - 2.2.1. Almacenes de columnas . . . 6
  - 2.2.2. Bases de datos documental . . . 7
  - 2.2.3. Almacenes clave-valor . . . 7
  - 2.2.4. Bases de datos orientada a grafos . . . 7
  - 2.2.5. Bases de datos orientadas a objetos . . . 8

3. Estado del arte: ciencia de los datos **9**
- 3.1. Introducción . . . 9
- 3.2. Flujo de trabajo . . . 9
  - 3.2.1. Formulación de la hipótesis . . . 9
  - 3.2.2. Obtención de los datos . . . 9
  - 3.2.3. Limpieza y preparación de los datos . . . 10
  - 3.2.4. Análisis explorativo . . . 13
  - 3.2.5. Visualización de datos y comunicación . . . 14
  - 3.2.6. Modelado . . . 14

4. Estado del arte: arquitectura del proyecto **19**
- 4.1. Introducción . . . 19
- 4.2. Características técnicas . . . 19
  - 4.2.1. Análisis de sentimiento . . . 20

5. Implementación: Apache Cassandra **21**
- 5.1. Introducción . . . 21
- 5.2. Requerimientos mínimos . . . 21
- 5.3. Instalación (nodo único) . . . 21
- 5.4. Modelo de datos de Apache Cassandra (API CQL) . . . 22
  - 5.4.1. Estructura interna de una fila . . . 23
  - 5.4.2. Claves de fila (partición) . . . 24
- 5.5. Diseño de las tablas . . . 25

6. Procesado del lenguaje natural **28**
- 6.1. Sintaxis . . . 28
  - 6.1.1. Semántica . . . 29

7. Análisis de sentimiento **30**
- 7.1. Clasificación estadística . . . 30
  - 7.1.1. Clasificador Bayesiano ingenuo . . . 30

8. Conclusiones **32**
- 8.1. Objetivos del proyecto . . . 32
- 8.2. Próximos pasos del proyecto . . . 32

9. Anexo: Estructura y instalación del proyecto **33**
9.1. Estructura del proyecto 33
9.2. Instalación del proyecto 34

10. Anexo: tipos de datos en CQL **36**

Glosario **37**

Referencias **37**

Índice de figuras

1. Base de datos de grafos 7
2. Flujo de trabajo del científico de datos 10
3. Dataframe.head() en Jupyter Notebook 11
4. Características de un dataset “limpio” 12
5. Análisis explorativo con pandas, parte 1 13
6. Temperaturas media diaria del río Fisher a su paso cerca de Dallas en el año 1990 14
7. Gráficos de ambas variables independientes contra la dependiente 16
8. Recta de regresión 18
9. Arquitectura del proyecto 19
10. Secuencia de eventos - análisis de sentimiento 20
11. Estructura de un espacio de claves 23
12. Estructura de una fila 23
13. Particiones de fila única 24
14. Particionado compuesto 25
15. Resultados del algoritmo Porter en algunas palabras 28
16. Fórmula del teorema de Bayes 30
17. Fórmula del teorema de Bayes 30

Índice de cuadros

1. Métodos de Pandas para leer datos 10
2. My caption 12
3. Dataset derretido 13
4. Tipos de datos CQL 36

1. Introducción

1.1. Contexto y justificación

Las bases de datos relacionales llevan con nosotros desde hace más de 40 años. Están basadas en el modelo relacional (Codd, 1970), en el que los datos se representan en tuplas, agrupadas en relaciones, que por lo general, representan entidades. Por tanto, almacenan **datos estructurados**, y resultan ideales en situaciones en las cuales el modelo de los datos está rígidamente definido.

De todas maneras, la mayor parte de la información que se genera a diario en Internet (pista: mucha), no se ajusta a niguna estructura, y por tanto, no puede ser almacenada tal cual en sistemas con esquemas definidos. En respuesta a ello, y al rápido aumento en la cantidad de datos a procesar, surgieron las **bases de datos NoSQL**. Al no usar el tradicional modelo relacional, éstas resultan mucho más adecuadas para manejar datos sin ningún tipo de organización o estructura.

Otro de los conceptos al que va enfocado el proyecto es el de ciencia de los datos, posiblemente el último "no va más" en el mercado. Hasta ha sido clasificado por Glassdoor como el mejor trabajo en América a inicios de 2016 [1]. Uno de los motivos por los cuales decidimos explorar éste concepto en el proyecto es el hecho de que aunque lleva ya tiempo establecido, aún sigue sin un currículo estandarizado, o una definición concreta a la que acogerse.

1.2. Objetivos

Los objetivos principales del proyecto son:

- Estudiar y conocer las bases de datos NoSQL, en concreto Apache Cassandra
  - Identificar los usos ideales para su uso.
- Entender el diseño y modelado de datos en Apache Cassandra.
- Almacenar Twits en Apache Cassandra en base a ciertos criterios de búsqueda. hagan referencia a un *hashtag*.
- Entender y familiarizarse con el flujo de trabajo del científico de datos.
  - Realizar análisis de sentimiento de los twits obtenidos mediante Python.

1.3. Metodología

El proyecto se dividirá en dos partes: la primera consistirá en un estudio de las bases de datos NoSQL, concretamente Apache Cassandra, comparando brevemente su modelo de datos con el modelo relacional. Además, exploraremos el concepto de *ciencia de los datos*, una nueva profesión que combina diferentes disciplinas, desde la programación hasta las matemáticas y la estadística, centrada en en la solución de problemas / obtención de conocimiento a partir de información.

La segunda parte consistirá en utilizar lo aprendido en la primera parte para implementar un sistema que aproveche las aptitudes de Cassandra para el *big data* y el análisis de datos. A ése fin, utilizaremos la API Streaming de Twitter para obtener twits y almacenarlos en Apache Cassandra, sobre los cuales luego utilizaremos Python para realizar análisis de sentimiento sobre los twits.

1.4. Características técnicas

Aquí se detallan las tecnologías y herramientas que se prevé utilizar durante el proyecto:

- **Documento de la memoria:** LaTeX
- **SGBD NoSQL:** Apache Cassandra
- **Lenguaje de programación:** Python 3.6.1

[1] https://www.glassdoor.com/blog/25-jobs-america-2016/

- **Librerías:**
  - Datastax Cassandra driver for Python
  - Get-Old-Tweets
  - Pandas
  - Natural Language Toolkit (NLTK)
- **Presentación:** Reveal.js
- **Aplicaciones auxiliares:**
  - Jupyter Notebook

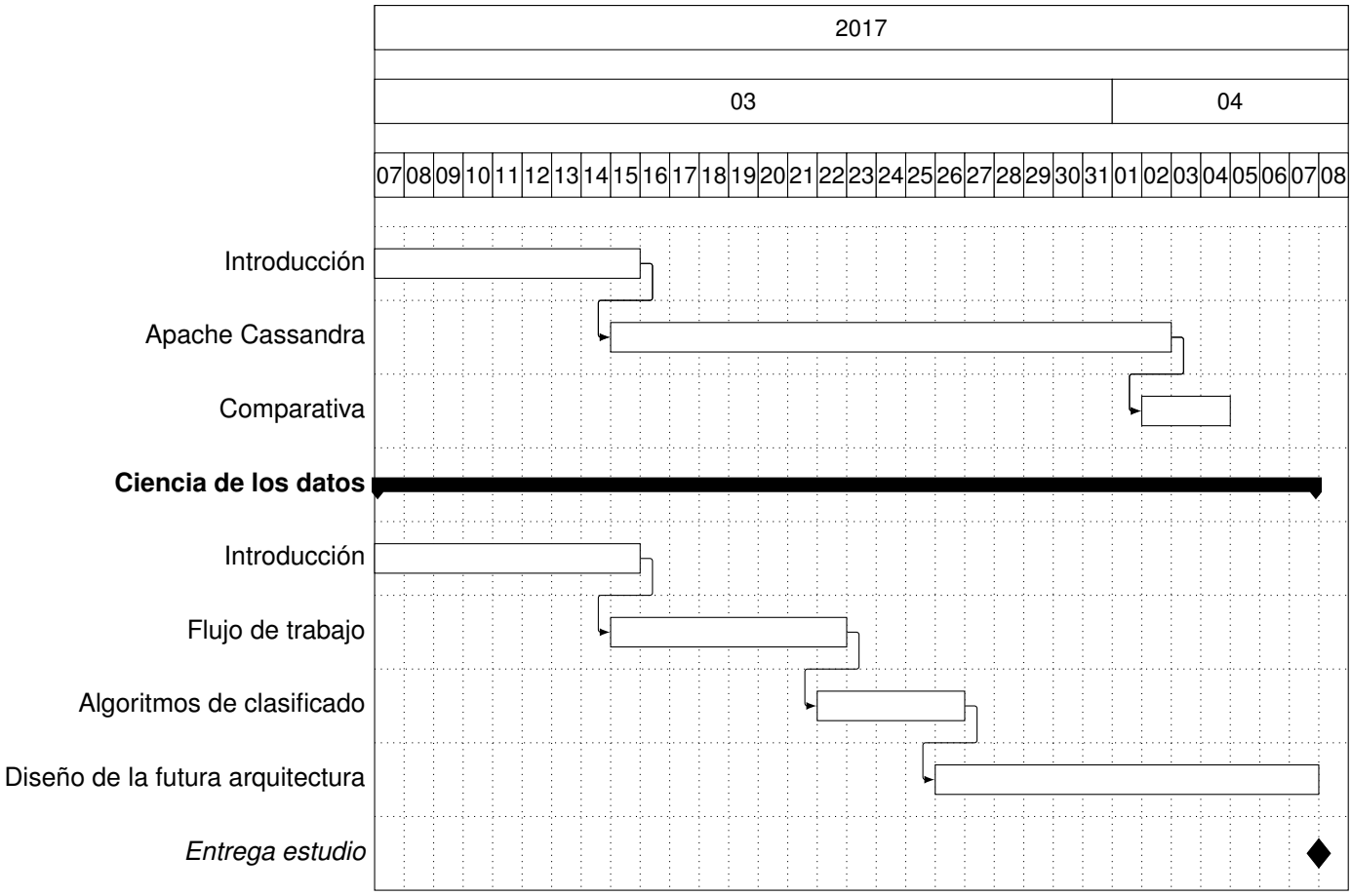
Nota: Se irá actualizando ésta lista a medida que se van descubriendo nuevas herramientas y librerías.

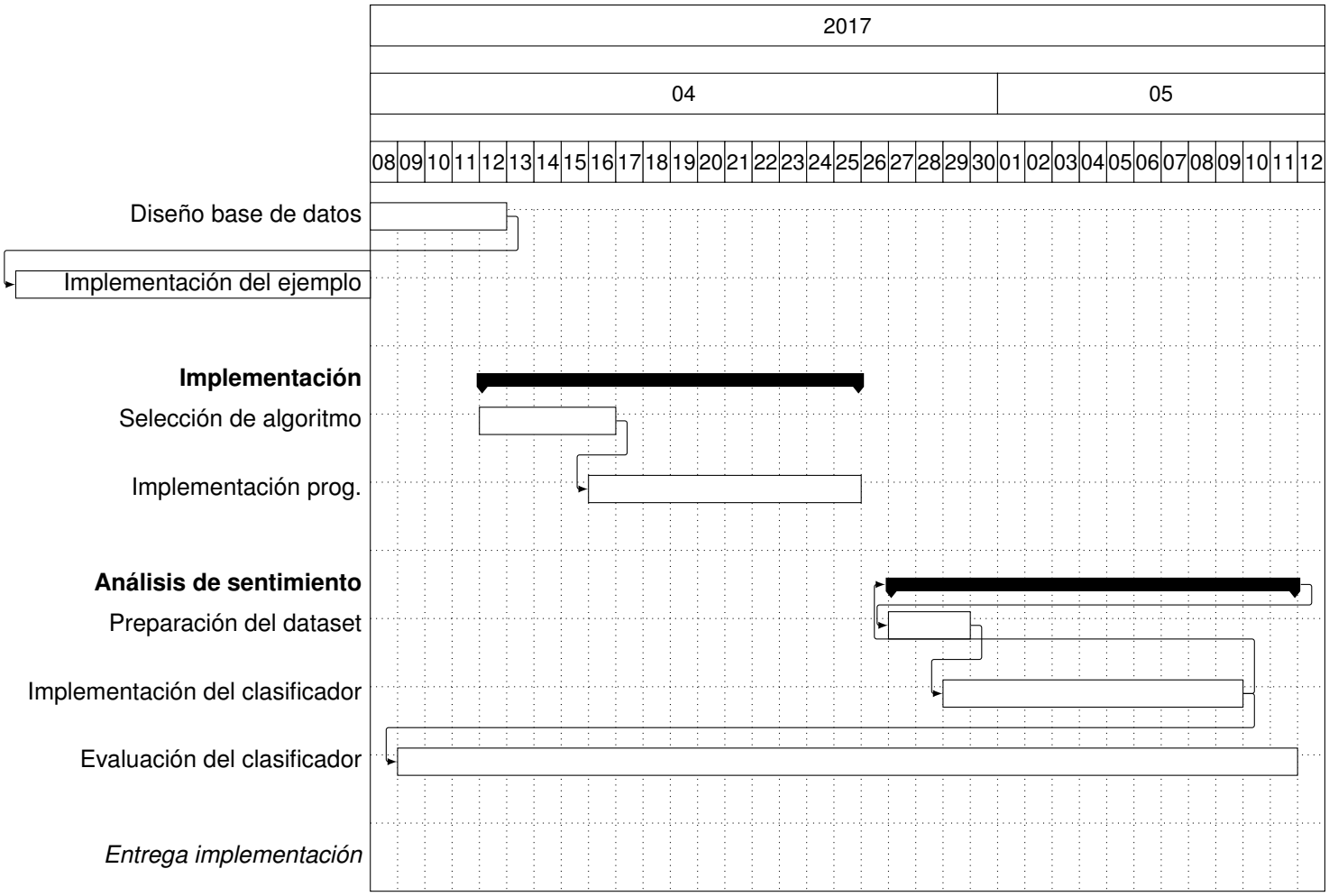
1.5. Planificación del proyecto

1.5.1. Lista de tareas

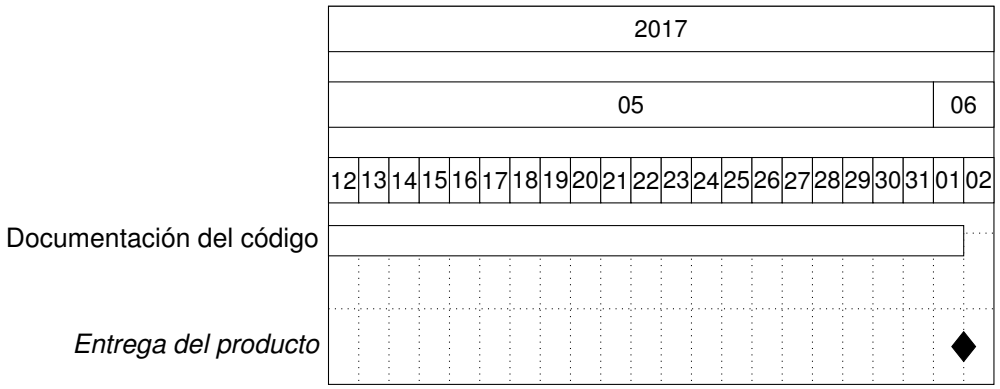
Tarea			Predecesora
1		**Redactar estudio**	
	1.1	Describir brevemente las bases de datos NoSQL	
	1.2	Estudiar Apache Cassandra: su modelo de datos, y sus casos de uso ideales	1.1
	1.3	Realizar una comparativa con las bases de datos relacionales	1.2
	1.4	Definir el concepto de ciencia de los datos	
	1.5	Entender la metodología del científico de datos	1.4
2		**Implementación del proyecto**	**1**
	2.1	Estudio de algoritmos de clasificado y selección del mismo.	
	2.2	Diseño e implementación de la base de datos	2.1
	2.3	Introducción a los algoritmos de clasificado	2.2
	2.4	Implementación del programa para popular la base de datos	2.3
	2.5	Análisis de los datos	2.4
2		**Redacción de la memoria**	

1.5.2. Diagrama Gantt





Nota: El análisis de datos es un proceso largamente iterativo, la longitud en el diagrama es orientativa.



1.6. Descripción de los capítulos

1. **Estado del arte:**

   1.1. **NoSQL y Apache Cassandra:** En esta sección estudiaremos brevemente las bases de datos NoSQL, en concreto Apache Cassandra. Trataremos de entender qué tipos hay de forma superficial, e identificar aquellas situaciones en las que es ideal utilizarlas.

1.2. **Ciencia de los datos:** Aquí introduciremos y exploraremos el concepto de ciencia de los datos. Intentaremos desentrañar a qué se refiere, con qué herramientas se trabaja, y con qué objetivos. Haremos énfasis en el flujo de trabajo del científico de los datos, es decir, la metodología estándar en un proyecto de éstas características.

2. **Implementación:** En ésta sección documentaremos la implementación de la base de datos, y el análisis de los twits, razonando las decisiones tomadas a lo largo del proceso.

3. **Conclusiones:** A ser redactada en la última entrega de la memoria, en ésta sección daremos nuestras opiniones sobre nuestro aprendizaje de la temática del proyecto, y el progreso del mismo.

4. **Anexo:** En éste capítulo se incluirán guías y descripciones varias como la preparación del entorno de desarrollo.

1.7. Formato de éste documento

En el documento, se utilizarán las siguientes convenciones de formato:

Bloque de comandos

```
comandos a ser introducidos por el usuario en un intérprete de comandos
```

```
1 for i in range(1,10):
2     print("Código fuente")
```

Aviso

Información a tener en cuenta.

¡Aviso importante!

Información importante a tener en cuenta.

¡Aviso muy importante!

Información muy importante a tener en cuenta.

2. Estado del arte: bases de datos NoSQL

2.1. Introducción

Un sistema gestor de bases de datos NoSQL es aquél que prescinde del modelo relacional a la hora de almacenar la información, y por ende del SQL tradicional como lenguaje de consultas. Éstas surgen debido a algunas carencias o problemas de las bases de datos relacionales, principalmente la **escalabilidad horizontal**, es decir, la adición de nuevos nodos para reducir el *downtime* del sistema en general.

Sobre lenguajes de consulta

Algunos SGBD NoSQL como Apache Cassandra, utilizan lenguajes con sintaxis y semántica similar al SQL.

2.2. Tipos de bases de datos NoSQL

Existen varios tipos de bases de datos NoSQL, generalmente clasificadas por el modelo de datos que emplean. Éstas clasificaciones no deberían ser tomadas literalmente, ya que muchas de las bases de datos que mencionaremos como ejemplos, implementan varios modelos de datos al mismo tiempo.

Teorema de CAP

En los sistemas distribuidos, el **teorema de CAP**, formulado por Eric Brewer, dice lo siguiente:

Teorema CAP. *Es imposible para un sistema distribuido ofrecer más de dos de las siguientes garantías al mismo tiempo: consistencia, disponibilidad y tolerancia de particiones.*

En el contexto del teorema, cada una de las garantías se define tal que:

- **Consistencia:** Cada lectura recibe como respuesta la escritura más reciente, o un error.
- **Disponibilidad:** Cada petición recibe una respuesta no-errónea, sin garantizar que contenga la escritura más reciente.
- **Tolerancia de particiones:** El sistema continúa funcionando aunque se degrade el funcionamiento de la red entre los nodos, o se pierdan mensajes entre ellos.

En realidad, la elección real es entre o consistencia, o disponibilidad, ya que debido al hecho de que una caída de parte del sistema no es común, únicamente nos vemos en la situación de escoger entre consistencia o disponibilidad, cuando ocurre un particionado en el sistema. Mientras el sistema esté funcionando con total normalidad, se puede garantizar tanto la disponibilidad como la consistencia del mismo.

Si el sistema permite la lectura antes de que todos los nodos tengan la última información escrita, el sistema será catalogado como uno consistente y tolerante a particiones (CP), mientras que si bloquea las lecturas hasta que todos los nodos se hayan actualizado, el sistema será disponible y tolerante a particiones (AP).

2.2.1. Almacenes de columnas

Los almacenes de columnas, bases de datos tabulares o orientados a columnas, son bases de datos que en lugar de serializar los datos en forma de filas como lo haría un SGBD relacional, lo hace en columnas:

Gracias a éste sistema, operaciones como consultar todos los registros que contengan un valor específico son mucho más eficientes en éste tipo de de SGBDs. Debido a ésto, resultan ideales en entornos OLAP (*Online analytical processing*, procesado analítico en línea), en los que generalmente el volumen de lecturas es mucho mayor al de inserciones / actualizaciones.

Ejemplos: Apache Cassandra, SAP Hana, Sybase IQ

2.2.2. Bases de datos documental

En una base de datos documental, la información se almacena en un formato semi-estructurado: es decir, con ficheros siguiendo un formato de documento específico, como podrían ser XML, JSON y YAML.

Éstos tipos de documentos almacenan toda la información referente al objeto que describen en el mismo documento, permitiendo que se añada información a documentos específicos sin tener que mantener una estructura consistente en todos los documentos.

En el espectro del teorema CAP, las bases de datos documentales se sitúan principalmente en el vértice de la tolerancia de particiones. Las bases de datos documentales AP (Available, partition-tolerant) como

2.2.3. Almacenes clave-valor

Las bases de datos clave-valor almacenan la información en *arrays* asociativos, o dicho de otra manera, diccionarios. Cada diccionario contiene registros o objetos, y de la misma manera que en las bases de datos documentales, cada uno puede albergar diferente información.

En éstas bases de datos, el valor de una clave es completamente opaco, por lo que es imposible filtrar por el contenido del registro. Al no disponer de lenguaje de consultas, la interacción con la base de datos se realiza mediante los comandos `get`, `put` y `delete`.

2.2.4. Bases de datos orientada a grafos

Los grafos son una estructura de datos, que modelan la información según relaciones entre objetos:

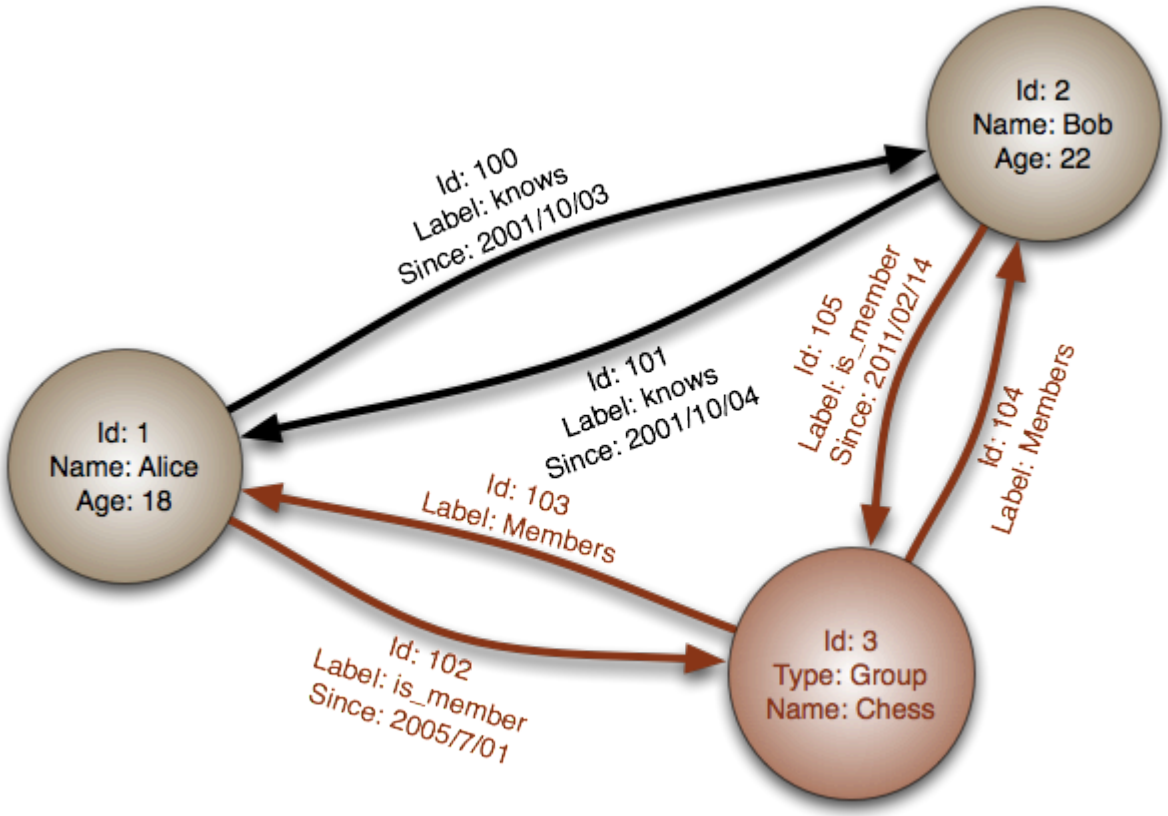


Figura 1: Base de datos de grafos

Los grafos se basan en los siguientes tres conceptos:

1. Los **nodos** representan cualquier entidad, y se pueden considerar equivalentes a otros objetos como un registro en una base de datos relacional, documento en una documental,

y así.

2. Las **aristas**, también llamadas relaciones, son las líneas que conectan los nodos, representando las relaciones entre ellos.

3. Las **propiedades** son etiquetas asociadas a los nodos nodo.

En contraste a las bases de datos relacionales, éstas almacenan directamente las relaciones entre nodos, por lo que, en teoría, las consultas que deban enlazar información son más veloces en bases de datos de grafos, ya que un SGBD relacional debe realizar una *join* en dos tablas diferentes. Ésta eficiencia se va acentuando conforme la complejidad de la consulta aumenta: una base de datos orientada a grafos sólo debe seguir las aristas a partir del nodo inicial.

Algunos ejemplos de bases de datos orientadas a grafos son AllegroGraph, Neo4j y Openlink Virtuoso.

2.2.5. Bases de datos orientadas a objetos

En éste tipo de bases de datos, la información se almacena como objetos en el paradigma homónimo. Según el “Object-Oriented Database Manifesto”, las bases de datos orientadas a objetos difieren de la serialización de los mismos en que deben proveer concurrencia, capacidad de recuperación, y algún mecanismo para consultar sobre éstos objetos.

Algunos ejemplos son: Db4o (en desuso), Realm.

3. Estado del arte: ciencia de los datos

Sobre los ejemplos de código

Los ejemplos de código en éste capítulo son meramente de ejemplo, por tanto no los incluiremos con la implementación. El código real del proyecto vendrá en forma de libretas de Jupyter y código fuente.

3.1. Introducción

Las bases de datos NoSQL no son lo único que ha traído el "Big Bang" de la información en Internet. De la misma manera que se hace evidente la necesidad de poder almacenar toda ésta nueva información de manera eficiente, también lo hace la necesidad de aprovechar y utilizar éstos datos de manera eficiente. A éstos efectos, se ha ido gestando una nueva disciplina dedicada al estudio de nuevos métodos para generar información útil y ofrecer valor a partir de datos de cualquier manera, a la que hoy se conoce como "ciencia de los datos", o en inglés, *data science*.

El primer uso del término data a 1966 cuando Peter Naur utilizó en 1966 porque no le gustaba el nombre de "ciencias de la computación", y consideraba este último más apropiado. Incluso hoy en día, aunque el término lleve usándose en su forma actual más de veinte años, no existe ninguna definición formal sobre qué entraña exactamente, sino que simplemente diferentes investigadores han aportado su opinión en qué quiere decir el término. Chikio Hayashi (Hayashi, 1998) la definió como *un concepto para unificar estadística, análisis de datos y métodos relacionados*, y Zhu et. al (Yanyong y Yun, 2011) la definieron de forma elegante como *una nueva ciencia cuyo objeto de estudio son los datos*.

3.2. Flujo de trabajo

Dentro de un proyecto de análisis de datos, existen una serie de pasos que en conjunto, describen una manera de trabajar que, aún sin existir un consenso sobre qué trata la ciencia de los datos, se ha estandarizado:

3.2.1. Formulación de la hipótesis

Al inicio del proyecto, nos encontramos ya de entrada con una decisión importante. ¿Qué es lo que queremos investigar? ¿Con qué objetivo?

Cada empresa tiene sus necesidades específicas, y opera en un mercado determinado. Por tanto, resulta importante que el científico de datos sepa formular una pregunta u hipótesis, que al ser respondida con el análisis, genere algún valor para la empresa. Para ello conviene conocer el sector y la actividad que se desarrolla en el mismo, tambien conocido como dominio de negocio (*domain knowledge* en inglés).

3.2.2. Obtención de los datos

Una vez ha definido la pregunta a responder, es hora de obtener los datos. Éstos pueden venir de una infinidad de fuentes o formatos de fichero: desde CSV, hojas de cálculo de Excel o LibreOffice y ficheros en formato JSON / XML, a bases de datos relacionales y NoSQL. Resulta imperativo que un científico de datos conozca de dónde puede sacar la información, y sepa recogerla toda en un mismo entorno.

La siguiente tabla muestra la lista de métodos de los que dispone Pandas (una librería para Python que provee estructuras de datos y herramientas para facilitar el análisis) para leer fuentes de datos, y generar un DataFrame (una estructura de datos tabular):

Obviamente, existen muchos más formatos en los que podemo encontrar los datos, a parte de los que Pandas puede leer. En ese caso, tocará recurrir a otras herramientas con las que podamos interactuar con el servicio en cuestión, para posteriormente transferir los datos a un DataFrame, o cualquier otra estructura que deseemos utilizar.

Un concepto interesante que queda a medias entre éste proceso de minería y el pre-procesado, es el uso de bases de datos intermedias. Es decir, el diseño de una base de datos adicional, dónde almacenaremos los datos después de la limpieza, pre-procesado y validación. De éste

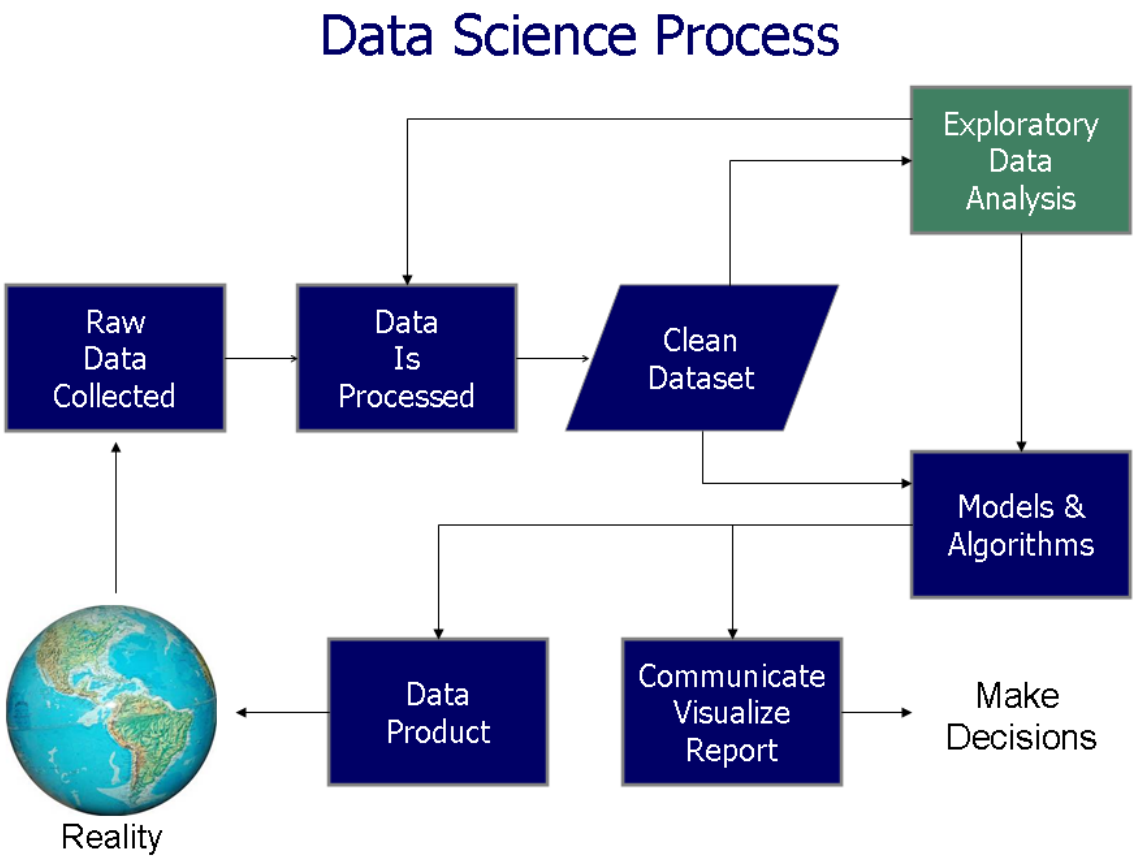


Figura 2: Flujo de trabajo del científico de datos

Método	**Fuente de datos**
read_csv	Fichero CSV
read_excel	Hoja de cálculo de Excel
read_hdf	Fichero HDF (Formato de datos jerárquico)
read_sql	Base de datos relacional
read_json	Fichero o cadena JSON
read_msgpack	Formato de serialización msgpack
read_html	Tablas dentro de ficheros HTML
read_gbq	Google BigQuery
read_stata	Ficheros DTA de Stata
read_sas	Ficheros de SAS
read_clipboard	Contenidos del portapapeles a través de read_table
read_pickle	Estructuras de datos guardadas mediante el formato pickle

Cuadro 1: Métodos de Pandas para leer datos

modo, se garantiza la calidad de los datos que utilizaremos en nuestro análisis, y la organización / mantenimiento de los mismos se hace más eficiente

3.2.3. Limpieza y preparación de los datos

Una vez ya hemos importado los datos, y los estamos manipulando con el lenguaje de nuestra elección, debemos echarle un vistazo general al dataset, y ver si observamos alguna anomalía.

En pandas, la clase DataFrame dispone de algunos métodos bastante interesantes para ver rápidamente algunos registros del dataset:

```
accidents_2015 = pd.read_csv("data/ACCIDENTS_GU_BCN_2015.csv", encoding="latin1
    ", delimiter=";")

accidents_2015            # Muestra el dataset en su totalidad
accidents_2015.head()     # Muestra los 5 primeros registros
accidents_2015.tail()     # Muestra los 5 últimos registros
accidents_2015.sample(15) # Muestra 15 registros aleatorios
```

Por ejemplo, si ejecutamos `accidents_2015.head()`, en una libreta de Jupyter veremos lo siguiente:

	Número d'expedient	Codi districte	Nom districte	Codi barri	Nom barri	Codi carrer	Nom carrer	Num postal caption	Descripc dia setmana
0	2015S005807	-1	Desconegut	-1	Desconegut	-1	Desconegut	Desconegut	Dimarts
1	2015S007685	10	Sant Martí	64	el Camp de l'Arpa del Clot	134801	Freser	0208 0208	Dimarts
2	2015S001364	10	Sant Martí	64	el Camp de l'Arpa del Clot	161407	Indústria	0336 0336	Dissabte
3	2015S004325	10	Sant Martí	64	el Camp de l'Arpa del Clot	226400	Las Navas de Tolosa	0343 0343	Divendre
4	2015S005540	10	Sant Martí	64	el Camp de l'Arpa del Clot	95506	Conca	0032 0034	Divendre

Figura 3: Dataframe.head() en Jupyter Notebook

Básicamente, las preguntas que deberíamos tratar de responder con éste análisis inicial son las siguientes:

1. Existe alguna anomalía en los datos? Falta algún valor?
2. Observamos algún patrón en los datos?

Trabajando con valores faltantes

En caso de que en nuestro dataset nos encontremos con variables para las que directamente no exista una observación, existen varias técnicas que podemos utilizar para tratar los registros.
Las dos primeras que describiremos, trabajan de forma destructiva, es decir, borrando datos:

> ¡Cuidado!
>
> Al borrar / ignorar la informacion faltante, éstos métodos pueden introducir **imparcialidad estadística**.

[2]

1. Borrar por completo cada registro en el que falte alguna observación
2. Ignorar observaciones faltantes, y realizar análisis únicamente sobre aquellas observaciones que sí se encuentren en el dataset.

[2] Se da cuando el resultado obtenido difiere del verdadero parámetro siendo estimado.

La otra manera de tratarlos recibe el nombre de **imputación**, y consiste en sustituir los valores nulos. Para ésto, también disponemos de varias técnicas:

1. Para variables contínuas, podemos sustituir valores nulos por la media aritmética de los valores no nulos.
2. Para variables categóricas, podemos sustituir valores nulos por la moda de los valores no nulos. El valor que más veces se dé, será el que utilizaremos en la imputación.

Tidy data

Aunque lo ideal sería que cada dataset que encontráramos viniera ya en un estado ideal para ponernos a analizar su contenido, esto generalmente no es así. Hadley Wickham utiliza el término *tidy data* (Wickham, 2014), para referirse a cualquier dataset que reúne las siguientes características:

1. Cada variable es una columna.
2. Cada observación es una fila.
3. Cada unidad observacional o entidad forma una tabla.

Figura 4: Características de un dataset “limpio”

Según Wickham ésta es la tercera forma normal de Codd, con las restricciones extrapoladas a lenguaje estadístico, y enfocada a un único dataset / tabla, en lugar de datasets interconectados en una base de datos relacional.

En pandas, podemos utilizar los métodos `Pandas.pivot()` y `Pandas.melt()` para realizar las tareas de “apilado” y “derretido”, que Wickham describe en el mismo artículo:

```
1 import pandas as pd
2 df = pd.DataFrame({'Id': {0: 'David', 1: 'b', 2: 'c'},
3                    'Lotería': {0: 'Juan', 1: 3, 2: 5},
4                    'Sorteo iPad': {0: 'Sergio', 1: 4, 2: 6}})
```

Visionando el DataFrame resultante mencionado el valor a secas, obtenemos lo siguiente:

Cuadro 2: My caption

	Id	Lotería	Sorteo iPad
0	David	0	2
1	Juan	3	4
2	Sergio	5	6

Observando el dataset, aunque sea de ejemplo, podemos ver que incumple uno de los tres principios del *tidy data*, que cada columna represente una variable. En éste caso, cada una describe un valor. Entonces, para transformar el dataset, en un formato más cómodo de trabajar para el análisis, utilizaremos `melt()`:

```
1 import pandas as pd
2
3 pd.melt(df, id_vars=['Id'], value_vars=['Lotería', 'Sorteo iPad'],
```

```
4         var_name='Concurso', value_name='VecesGanado')
```

Cuadro 3: Dataset derretido

	Id	Sorteo	Veces Ganado
0	David	Lotería	0
1	Juan	Lotería	3
2	Sergio	Lotería	5
3	David	Sorteo iPad	2
4	Juan	Sorteo iPad	4
5	Sergio	Sorteo iPad	6

El método `melt`, transforma un dataset en un formato "ancho", y apila sets de variables en únicamente dos columnas: Una representando el nombre de la variable, y la otra, el valor de la misma. Una vez tenemos el dataset de ésta manera, resulta mucho más sencillo realizar operaciones cómo por ejemplo ver los resultados de la lotería, o gente que haya ganado un concurso más de 5 veces:

```
1  df[df['Sorteo'] == 'Lotería']
2  df[df['VecesGanado'] > 5]
```

3.2.4. Análisis explorativo

En realidad, el preparado del dataset, y lo que se denomina análisis explorativo (cuyo propósito es familiarizarnos mejor con el mismo), van bastante unidos de la mano en el sentido de que para poder corregir anomalías en el dataset, debemos verlo bien de cerca. De todas maneras, en Pandas disponemos de muchísimos más métodos de los que hemos visto antes para conocer más de cerca el dataset:

```
1
2     DataFrame.shape   # Devuelve las dimensiones (Filas x Columnas)
3     DataFrame.columns # Devuelve las columnas (features)
4     DataFrame.info    # Devuelve la cantidad de valores no-nulos en cada
          columna
5     DataFrame.describe() # Agregado de estadísticas como la media, min / max,
          etc...
6
7     DataFrame['VariableCategórica'].unique() # Devuelve la lista de valores que
           toma la variable.
```

Figura 5: Análisis explorativo con pandas, parte 1

Llegados a éste punto, en el que hemos identificado la variable en cuestión como categórica, es necesario cambiarle el tipo a la misma de cara a la representación que Pandas hace de la misma:

```
1  df['VariableCategórica'] = df['VariableCategórica'].astype('category')
```

Una vez hecho ésto, podemos ver la distribución de valores de la variable:

```
df['VariableCategórica'].value_counts()
```

3.2.5. Visualización de datos y comunicación

Otra parte muy importante que va de la mano con el análisis explorativo, es generar gráficos que nos ayuden a entender de manera visual, diferentes aspectos sobre el dataset. Aspectos muy importantes como cualquier patrón, variación estacional en series de tiempo y correlación, suelen estar ocultos en los números, y los gráficos son una buena manera de hacerlo.

Para mostrar ejemplos de éstos gráficos, utilizaremos un dataset de la temperatura media diaria del río Fisher, cerca de Dallas, de enero de 1998 a diciembre de 1991:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
%matplotlib inline # Necesario para que Jupyter muestre las figuras en la misma página
matplotlib.style.use('ggplot')
```

```
df = pd.read_csv('fisher.csv', delimiter=',', parse_dates=[0], index_col = 0)
```

Antes de proseguir, es importante destacar los dos últimos parámetros que le hemos pasado a (read_csv): (parse_dates) y (index_col). En conjunto, éstos harán que el primer campo del dataset, que es la fecha, actúe como índice, permitiéndonos hacer cosas como la siguiente:

```
df['1990'].plot(figsize=(15, 5)) # Muestra únicamente el año 1990
```

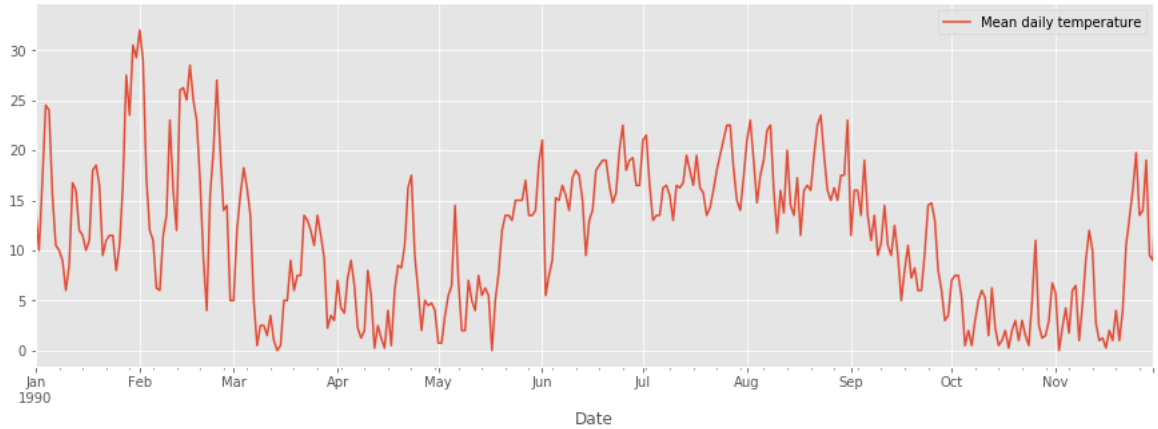


Figura 6: Temperaturas media diaria del río Fisher a su paso cerca de Dallas en el año 1990

3.2.6. Modelado

Un modelo estadístico es una clase específica de modelo matemático, una manera formal de describir sistemas mediante conceptos matemáticos.

Generalmente, un modelo estadístico implica dos tipos de variables: dependientes y explicativas:

- **Variable dependiente:** se dice de aquella que queremos estudiar o predecir. Suele ser aquella que se representa en el eje de las Y.
- **Variable explicativa:** También llamadas independientes, son aquellas en las que nos apoyamos para describir las variables dependientes.

Existen infinidad de modelos estadísticos que tienen en cuenta tanto el tipo de las variables, como el número de éstas. Uno de los más simples de entender y implementar es la regresión lineal, empleada para estudiar la relación de dependencia entre una variable dependiente Y, una o más variables independientes X_i, y un término aleatorio ϵ. Si se utiliza una única variable independiente X, hablamos de una regresión linear simple.

Teoría de la regresión linear simple

La ecuación que describe la recta de regresión es:

$$Y_i = \beta_0 + \beta_i X_i + \epsilon_i \tag{1}$$

Dónde:

- $\hat{Y}$ es el valor a predecir
- β es la pendiente de la recta
- A es el punto de intersección en el eje de las Y.

La ecuación para la pendiente es:

$$\beta = rs\frac{\overline{x} * \overline{y} - \overline{xy}}{(\overline{x})^2 - \overline{x^2}} \tag{2}$$

Mientras que la del punto de la intersección es:

$$A = M_y - bM_x \tag{3}$$

Implementando la regresión lineal simple

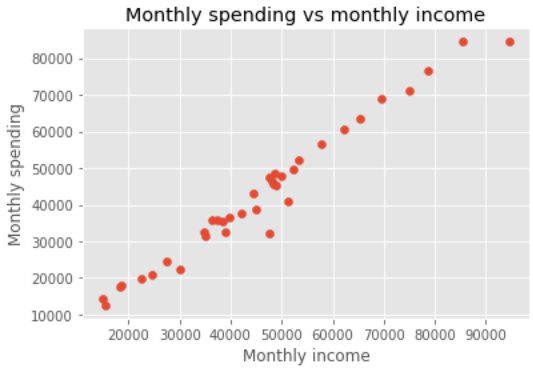
En los siguientes bloques de código se demuestra la implementación de una regresión lineal simple en Python. Para éste ejemplo, estamos utilizando otro dataset nuevo, que simula el gasto mensual de una persona teniendo en cuenta sus ingresos y la cantidad de viajes que realiza:

```
1  import pandas as pd
2  import numpy as np
3
4  import matplotlib
5  import matplotlib.pyplot as plt
6
7  from sklearn import linear_model, model_selection
8  from sklearn.metrics import mean_squared_error
9  %matplotlib inline
10 matplotlib.style.use('ggplot')
```

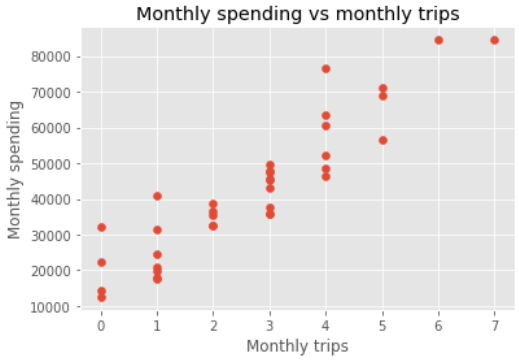
```
1  df = pd.read_csv("data/monthlySpending.csv")
2  df.info()
```

```
# Gráfico izquierdo
plt.scatter(df['M_INC'], y)
plt.title("Gasto mensual vs ingresos mensuales")
plt.xlabel("Ingresos mensuales")
plt.ylabel("Gasto mensual")

# Gráfico derecho
plt.scatter(df['M_TRP'], y)
plt.title("Gasto mensual vs viajes mensuales")
plt.xlabel("Viajes realizados")
plt.ylabel("Gasto mensual")
```



(a) Gasto mensual vs ingresos mensuales



(b) Gasto mensual vs viajes mensuales

Figura 7: Gráficos de ambas variables independientes contra la dependiente

En el siguiente bloque definimos nuestras variables, tanto dependientes (el gasto mensual), como independientes (el número de viajes, o los ingresos).

```
y = df['M_SPEND']
X = df['M_TRP']
```

```
linearRegression = linear_model.LinearRegression()

# Validación cruzada
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y,
test_size = 0.33)

linearRegression.fit(X_train, y_train)
```

Validación cruzada

Merece la pena explicar la segunda instrucción, `train_test_split()`. Éste método se utiliza para dividir nuestras variables en dos datasets cada una, uno que se utilizará para entrenar el algoritmo, y el otro para comprobar el mismo una vez entrenado. Ésto se hace debido a que la regresión daría una precisión del 100 % si lo probamos con los mismos datos con los que lo entrenamos. A ésto se le llama **validación cruzada**(*cross-validation*). Se utiliza para estimar cómo

los resultados de un análisis estadístico se ajustarán a datos independientes. Generalmente se aplica cuando se desea estimar la fiabilidad de un modelo predictivo en la práctica.

Validación cruzada en K iteraciones

La validación cruzada en k iteraciones (en inglés, *K-fold cross-validation*), es otro método de validación cruzada en el que la muestra original se reparte en k bloques. De éstos bloques, se guarda uno de ellos como bloque de validación / prueba, y el resto de los bloques (k-1) se utilizan como bloques de entrenamiento.

El proceso se repite k veces, utilizando cada vez un bloque distinto para la validación. Una vez completadas todas las iteraciones, se utiliza la media de los resultados para ofrecer una única estimación.

Una vez ya entrenado el modelo, podemos obtener las métricas de regresión:

```
1 print('Root mean squared error: %.2f ' % math.sqrt(mean_squared_error(reg.
      predict(X_test), y_test)))
2 print('Variancia explicada: %.2f' % explained_variance_score(reg.predict(X_test
      ), y_test))
3 print('Coeficiente de determinación: %.2f' % r2_score(reg.predict(X_test),
      y_test))
```

Interpretadas tal que:

- Cada punto se aleja de media en 2077 unidades de la línea de mejor ajuste (*Root mean squared error*)
- Variancia
- El coeficiente de determinación, también escrito como R^2, indica la proporción de la variancia en la variable dependiente que es predecible de la variable independiente. En otras palabras, estima cuán bien la recta de regresión se ajusta a los datos reales. El valor va de 0 a 1, donde un 1 indica un ajuste perfecto.

Una vez verificado el modelo, podemos visualizar la recta de regresión junto a los *datapoints* con el siguiente bloque de código:

```
1   plt.scatter(X, y)
2
3   # Línea de regresión
4   plt.plot(X, np.poly1d(np.polyfit(X, y, 1))(X), color='blue')
5   plt.xlabel('Monthly trips')
6   plt.xlabel('Monthly spending)
```

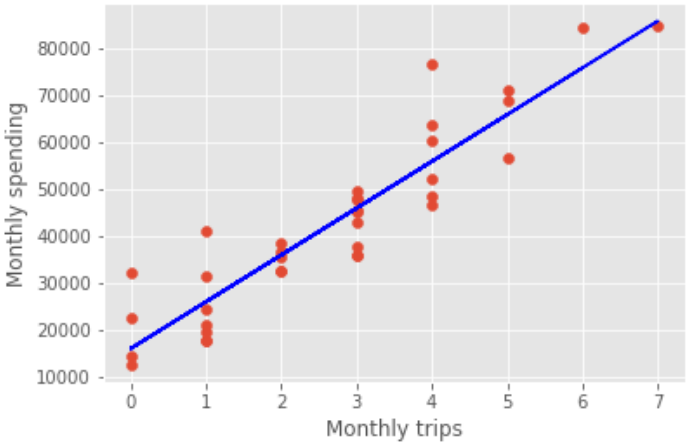


Figura 8: Recta de regresión

4. Estado del arte: arquitectura del proyecto

4.1. Introducción

El siguiente diagrama muestra un esquema simple de la arquitectura del proyecto:

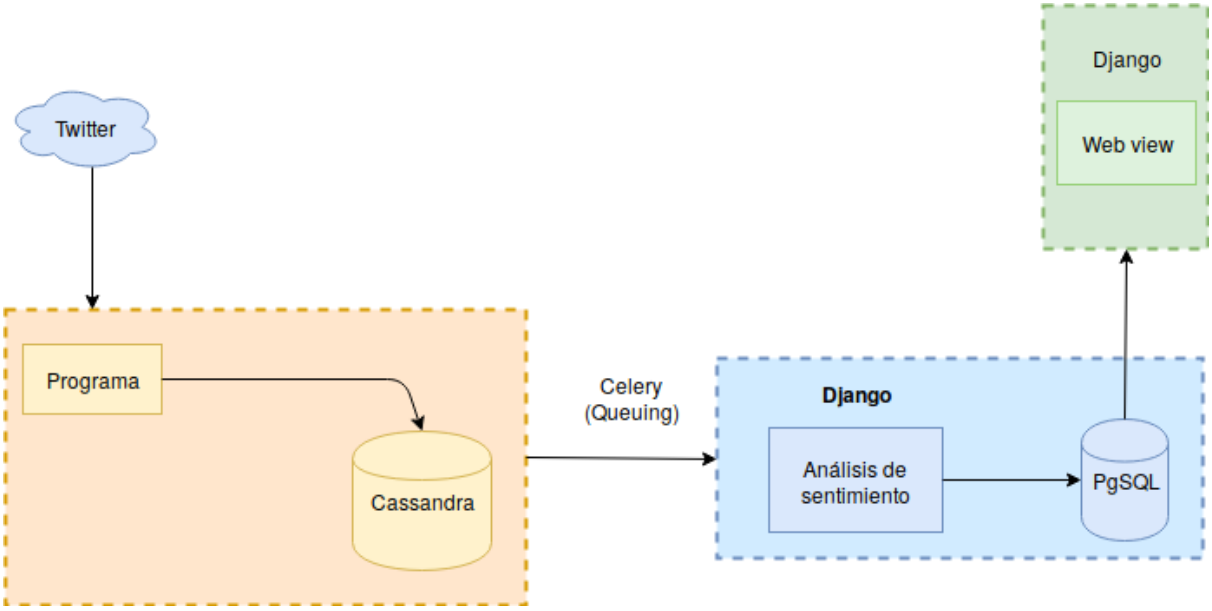


Figura 9: Arquitectura del proyecto

La aplicación o proyecto, constará de una *pipeline* con los siguientes componentes:

1. En primer lugar, un programa en Python recogerá los twits acorde a diferentes criterios, y los depositará en Apache Cassandra. Cada tabla estará modelada correspondiendo al criterio por el que se recogieron los twits: por ejemplo, podemos tener una tabla llamada `tweets_by_region`, en los que irán twits que se obtuvieron consultando por la región geográfica en concreto.
2. El segundo bloque de la pipeline interactuará con Apache Cassandra, procesando los twits, y obteniendo un resultado en base a las especificaciones del usuario: evolución del sentimiento a lo largo de una franja de tiempo, por zona geográfica, demografía de los usuarios de Twitter, etc... Una vez procesados los twits, el resultado se almacenará en una base de datos PostgreSQL, para un posterior visionado más sencillo.
    - La selección de criterios, se realizará mediante una aplicación web en Django.
3. El tercero y final bloque de la pipeline consistirá en una aplicación web desarrollada en Django, en la cual se observarán los resultados del análisis, obtenidos directamente de la base de datos en PostgreSQL.

4.2. Características técnicas

Obtención de twits

Originalmente, queríamos realizar ésta tarea con la librería tweepy, que interactúa directamente con las API de Twitter para obtener los datos. Nos hemos visto obligados a optar por otro proyecto, por limitaciones con las mismas:

- La API Streaming va recogiendo y mostrando twits a la que van sucediendo. Ésto es ideal para conexiones persistentes, y podría ser utilizado en caso de que decidamos ofrecer análisis de sentimiento en tiempo real.

- La API REST expone varios *endpoints* para recibir twits en base a varios criterios. El único problema con ésta API es que por lo general, no devuelve twits más viejos de una semana, lo cual nos deja un *dataset* bastante pobre con el que trabajar.

Ésto nos ha llevado a utilizar el siguiente proyecto como mecanismo para obtener los twits: GetOldTweets-python. Funciona aprovechando el mecanismo real mediante el que Twitter devuelve los resultados de la búsqueda en el navegador: mediante un cursor y respuestas JSON. Ya que está utilizando una licencia MIT, tenemos libertad absoluta para coger el código, y ajustarlo a las necesidades del proyecto.

4.2.1. Análisis de sentimiento

Para ésta parte, las librerías a utilizar son principalmente Pandas y el Natural Language Toolkit para manipular el texto, a parte del driver de Cassandra para acceder a la base de datos. De momento, no se preveé implementar código desde cero para sustituir partes de éstos frameworks excepto en aquellos casos realmente necesarios.

El siguiente es un breve diagrama que muestra de cualquier manera la secuencia de eventos de éste programa:

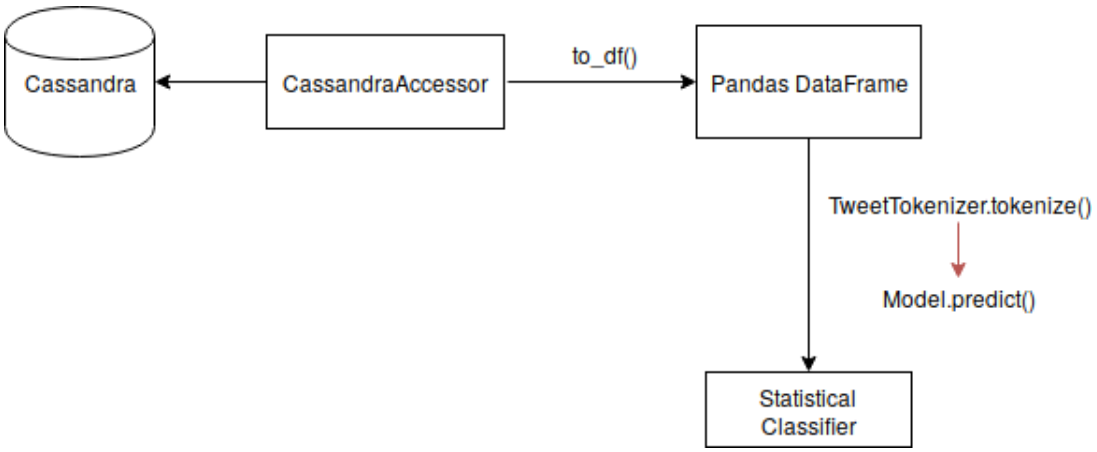


Figura 10: Secuencia de eventos - análisis de sentimiento

Realmente, ya que la única "clase" que acabemos escribiendo nosotros sea el tokenizador, debido a pruebas no-satisfactorias con algunos existentes, no podemos crear un diagrama de clases al uso.

Éste programa primero se conectará a Apache Cassandra, y devolverá un DataFrame de pandas con los resultados de la consulta. Una vez hecho ésto, se tokenizará, y con las *features* resultantes, se utilizará un modelo estadístico ya entrenado para predecir si el twit en cuestión es negativo o positivo.

El modelo a utilizar en el proyecto, será un **clasificador bayesiano ingenuo**. Son una familia de clasificadores probabilísticos [3] basados en la aplicación del teorema de Bayes con una fuerte asunción de independencia (de ahí el apelativo de ingenuo) entre las características.

[3] Capaces de devolver una distribución de probabilidades, no sólo la predicción de clase.

5. Implementación: Apache Cassandra

5.1. Introducción

En el caso de Apache Cassandra, nos encontramos con un híbrido de base de datos clave-valor y orientado a columnas. Su modelo de datos se describe como un almacén de filas particionado, con consistencia configurable. Ésta es una extensión del concepto de consistencia eventual (que garantiza que si no se actualiza un valor, eventualmente todos los accesos al mismo devolverán el valor más reciente), permitiendo que el cliente decida cuán consistentes deben ser los datos de cada operación de lectura y escritura.

5.2. Requerimientos mínimos

Apache Cassandra necesita para funcionar:

- Java 8 o superior
- Para usar cqlsh, Python 2.7 o superior
- En términos de hardware [4], se recomienda:
  - 4 GB de RAM mínimo en entornos virtualizados (EC2 Large)
  - Cpu de 8 núcleos en servidores físicos, para entornos virtualizados se recomienda cualquiera que soporte *CPU bursting*.
  - DISCO

5.3. Instalación (nodo único)

1. Primeramente, se deben añadir el repositorio y la clave:

   Añadiendo el repositorio

   ```
   echo "deb http://debian.datastax.com/community stable main" | sudo
       tee -a /etc/apt/sources.list.d/cassandra.sources.list
   ```

   Añadiendo la clave

   ```
   curl -L http://debian.datastax.com/debian/repo_key | sudo apt-key
       add
   ```

2. Una vez añadidos, ya podemos proceder con la instalación:

   Instalando Apache Cassandra

   ```
   sudo apt-get update && sudo apt-get install cassandra
   ```

3. Cuando ya esté instalado, sería buena idea arrancarlo y habilitarlo en cada reinicio del sistema:

   Habilitando servicio

   ```
   systemctl start cassandra
   ```

[4] https://wiki.apache.org/cassandra/CassandraHardware

```
systemctl enable cassandra
```

4. Para comprobar el estado del servidor, podemos utilizar el comando `nodetool`:

Comprobando el estado de Cassandra

```
nodetool status

Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
-- Address Load Tokens Owns (effective) Host ID Rack
UN 127.0.0.1 222.14 KB 256 100.0% 2a0b7fa9-23c6-40d3-83a4-
    e6c06e2f5736 rack1
```

5.4. Modelo de datos de Apache Cassandra (API CQL)

Debido a que es una base de datos distribuida, no existen joins de ningún tipo, y por tanto, no hay manera de enlazar tablas. Lo que se hace entonces, es diseñar las tablas de forma autocontenida: todo lo que se desee conocer de un registro, estará en la misma tabla.

La unidad principal de un clúster de Apache Cassandra, análoga a la base de datos (o esquema de la misma), es el *keyspace*. Dentro encontramos tres atributos principales:

Sobre el factor de replicación

Generalmente, no debería exceder el nombre de nodos en el clúster, aunque siempre puede dejarse alto y añadir los nodos a posteriori.

1. El **factor de replicación** especifica el número de nodos que recibirán copias de los mismos datos.
2. La **estrategia de localización de réplicas** describe de qué manera se distribuyen las réplicas de datos. Existen dos a utilizar, dependiendo de cuántos datacenters consista nuestro clúster:
   - La `SimpleStrategy`, utilizada para un datacenter único, coloca la primera réplica en el nodo especificado por el particionador, colocando las siguientes réplicas en los nodos contiguos en el sentido de las agujas del reloj, sin prestar atención a la topología de red.
   - La `NetworkTopologyStrategy`, es ideal para situaciones en las que disponemos de más de un datacenter, o planeamos expandirnos en el futuro. Ésta estrategia especifica cuántas copias queremos tener en cada datacenter. `NetworkTopologyStrategy` funciona igual que la estrategia simple, excepto que deja de colocar réplicas en el mismo datacenter cuando detecta que ha pasado a otro.

     Generalmente, la configuración de una `NetworkTopologyStrategy` se realiza de una de dos maneras, en base a resistir fallos del clúster, y evitar lecturas *cross-datacenter* (de un datacenter a otro):
     - Dos réplicas por datacenter permiten el fallo de un nodo por datacenter, y lecturas locales a un nivel de consistencia `ONE`.
     - Tres réplicas por datacenter permiten el fallo de un nodo por datacenter a un nivel de consistencia más fuerte (`LOCAL_QUORUM`), o múltiples fallos a nivel de consistencia `ONE`.

3. Las **familias de columnas** son contenedores de colecciones de filas, cada una contenienendo columnas ordenadas. Las familias de columnas son análogas a las tablas de una base de datos relacional, en cierto modo.

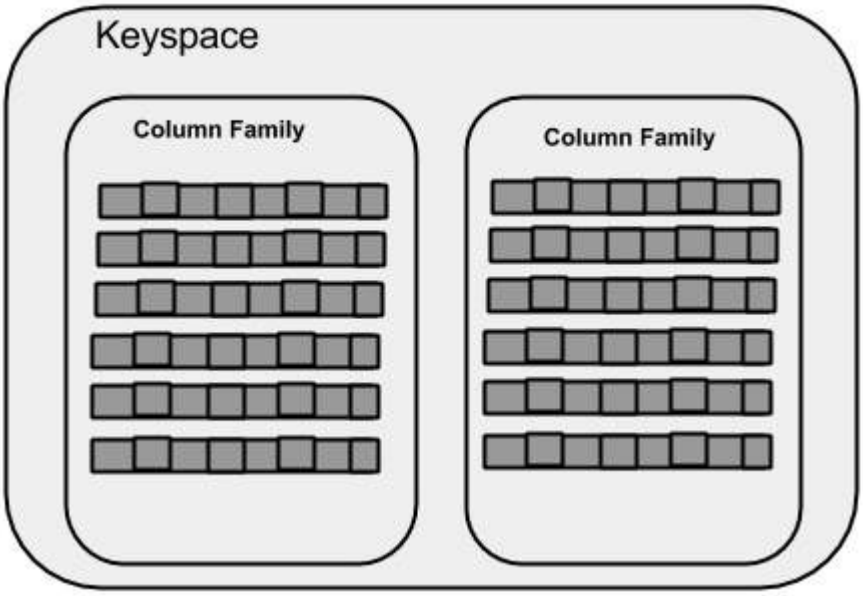


Figura 11: Estructura de un espacio de claves

5.4.1. Estructura interna de una fila

La fila es la unidad de almacenamiento de datos más pequeña dentro de Apache Cassandra. Dentro de ellas encontraremos pares de clave-valor. Éstos pueden contener valores atómicos, o colecciones (como pueden ser listas, mapas, o conjuntos).

> **Sobre los tipos de datos en CQL**
>
> En el anexo, tendremos una sección dedicada a los tipos de datos admitidos en CQL, con ejemplos de los mismos.

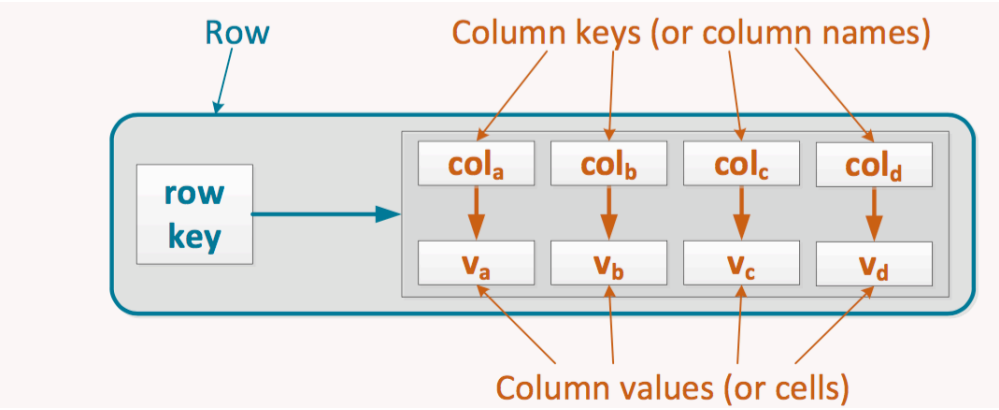


Figura 12: Estructura de una fila

- La **clave de fila** identifica inequívocamente la fila en cuestión dentro de la familia de columnas.

- Las **claves de columna** identifican inequívocamente un **valor** dentro de la fila. En base al número de claves de columna, clasificamos las filas en dos tipos:
  - Las filas “delgadas” (del inglés *skinny*), tienen un número fijo y relativamente pequeño de claves de columna.
  - Por el otro lado, las filas “anchas” (del inglés *wide*), tienen un número muy grande de columnas, que puede ir creciendo.

5.4.2. Claves de fila (partición)

Las claves de fila, que a partir de ahora llamaremos de partición, cumplen un doble cometido:

- En primer lugar, como hemos dicho, identifican inequívocamente una fila.
- El otro propósito que cumplen, es controlar la distribución de los datos a través del clúster. Eso quiere decir, que todas las filas con un mismo valor en la clave de partición, irán al mismo clúster.

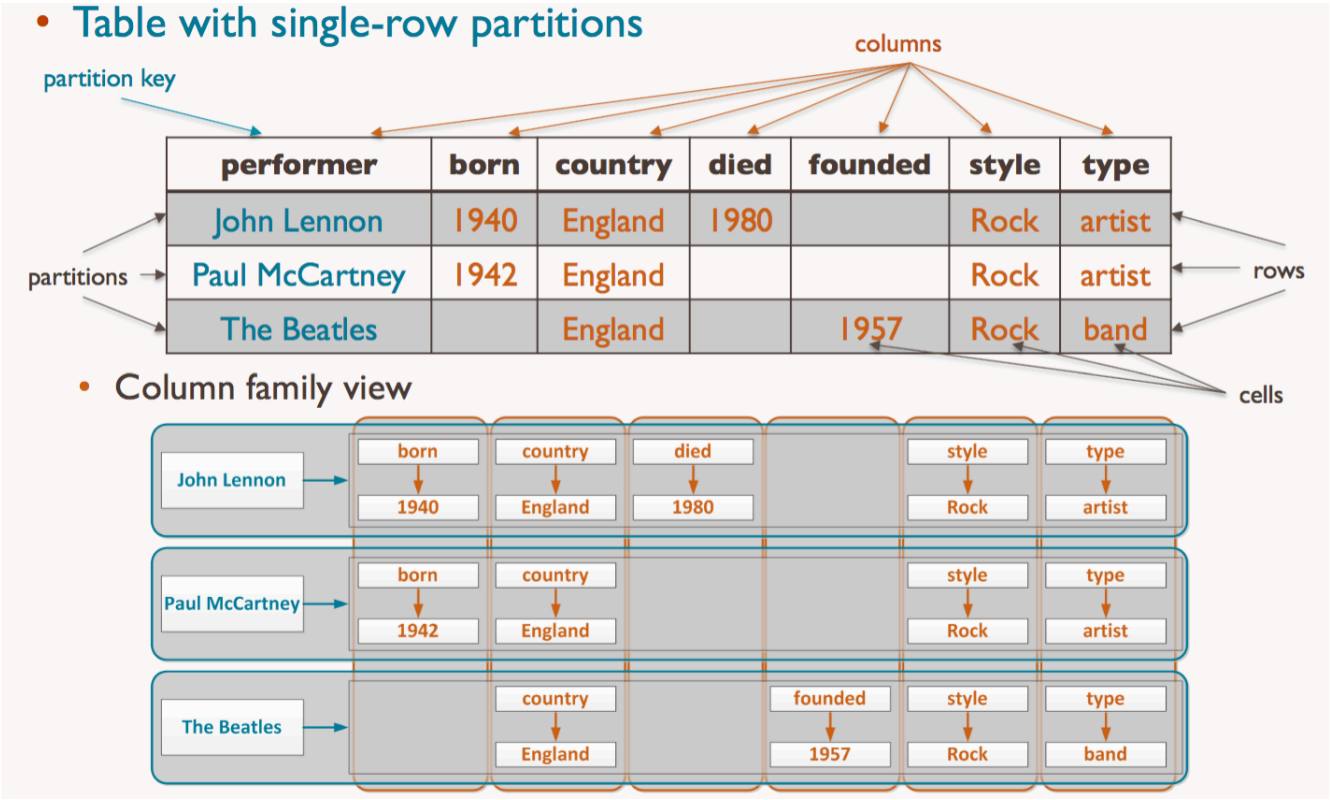


Figura 13: Particiones de fila única

Claves compuestas y de agrupación

Las claves de fila, no tienen por qué ser únicas. De la misma manera que una clave primaria compuesta en un RDBMS identifica un registro por un par de valores, una clave de partición compuesta funciona de la misma manera:

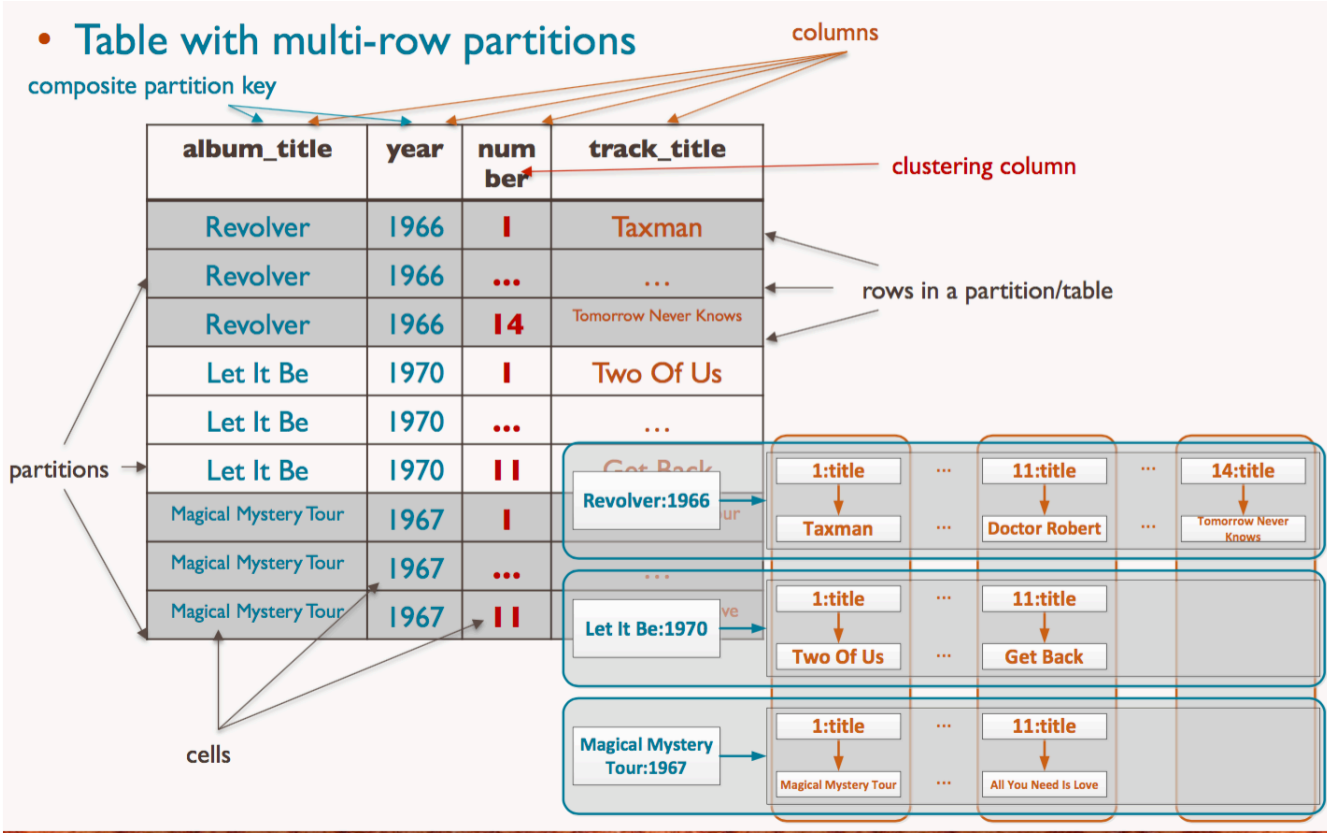


Figura 14: Particionado compuesto

La figura también introduce otro concepto, el de clave de agrupación (en inglés, *clustering key*). Ésta determina la ordenación de los datos dentro de la partición. En el ejemplo, la clave de particionado compuesta es (`album_title`, `year`), mientras que la de agrupación es `num_bef`.

5.5. Diseño de las tablas

En el modelo relacional, un concepto importante a la hora de diseñar la base de datos es la **normalización**, el proceso de organización de las relaciones y sus atributos para aumentar la integridad de los datos y reducir la renundancia. En Apache Cassandra (y cualquier otro sistema distribuido), no tiene sentido perseguir la normalización. En el caso de Cassandra, es necesaria la duplicación de datos para tanto obtener mayor disponibilidad, y eficiencia en las lecturas.

Objetivos del modelado

Mientras que en un SGBD relacional modelamos la base de datos para que sea normalizada y minimizar información duplicada, en Apache Cassandra, existen dos reglas / principios a tener en cuenta a la hora de diseñar las tablas:

1. Repartir la información equitativamente por todo el clúster: el caso ideal es que cada nodo tuviera aproximadamente la misma cantidad de datos. Recordando lo explicado previamente, ésto lo conseguimos mediante una buena **clave de particionado**.
2. Minimizar el número de particiones leídas: al hacer una consulta, interesa leer del mínimo de particiones posibles. La razón es que se realizan comandos separados a cada nodo, por cada partición que leemos, lo que incurre en mayor *overhead* y latencia.

El problema, es que estos dos principios entran en conflicto. Al repartir la informació por el clúster, aumentamos el número de particiones en las que los datos recaen. La respuesta a éste dilema es un tanto cliché: hay que balancear ambos requerimentos.

La forma de optimizar las lecturas, es **modelar a partir de las consultas a realizar**, y no a partir de las entidades a almacenar. Dicho de otra manera, por cada tipo de consulta que queramos hacer, habrá una tabla que refleje la consulta.

Aplicando las reglas

Antes de ponernos a crear las tablas, debemos crear el *keyspace*. Ya que sólo tendremos un nodo en el clúster, emplearemos la `SimpleStrategy`, y un factor de replicación de 1:

```
CREATE KEYSPACE twitter WITH REPLICATION = { 'class' : 'SimpleStrategy', '
    replication_factor' : 1 };
```

En nuestro proyecto, se nos pueden ocurrir diferentes criterios por los que nos interesaría realizar una consulta:

- Todos los twits de una misma cuenta
- Todos los twits que contengan un hashtag concreto
- Todos los twits que mencionen una cuenta

Teniendo en cuenta el principio de optimización de consultas, podemos crear las tablas tal que:

```
CREATE TABLE twitter.tweets_by_username(
    author_username text,
    tweet_text text,
    hashtags list<text>
    retweets int,
    favorites int,

    PRIMARY KEY (author_username, tweet_text)
);
```

En la consulta de hashtags, existe un problema. Aunque Cassandra soporta tipos de colecciones, éstos solo son soportados como clave primaria si son *frozen*. Si se congelan, la única manera de consultar por ellos será por la lista entera. Es decir, si tenemos un tweet con los hashtags #fútbol y #barça, no podremos consultar sólo por uno de ellos, deberemos realizar la consulta por la lista ['#fútbol', '#barça'] entera, por lo que la consulta será muy rígida en los twits que obtendrá.

```
CREATE TABLE twitter.tweets_by_hashtags_frozen(
    author_username text,
    tweet_text text,
    hashtags frozen <list<text>>
    retweets int,
    favorites int,

    # Los datos se repartirán en base a la lista congelada, y ordenados por el
        texto
    PRIMARY KEY (hashtags, text)
);
```

La *workaround* que hemos pensado, viola el segundo principio ya que es posible que lea muchas particiones, pero conserva la capacidad de consultar por parte de la lista:

```
CREATE TABLE twitter.tweets_by_hashtags(
    author_username text,
    tweet_text text,
    search_hashtag text,
    hashtags list<text>,
    retweets int,
    favorites int,

    # Los datos se repartirán en base al hashtag por el que se buscó, y
    ordenados por el texto
    PRIMARY KEY (search_hashtag, text)
);
```

6. Procesado del lenguaje natural

En el campo del procesado del lenguaje natural, existen varias técnicas, que pueden ser clasificadas de manera rápida en cuatro grupos: aquellas referentes a la sintaxis, semántica, discurso y habla. Ya que nuestro proyecto trata exclusivamente texto escrito, únicamente se detallarán algunas técnicas referentes a las dos primeras categorías:

6.1. Sintaxis

La sintaxis se define como la parte de la gramática que estudia la generación de frases, y las reglas que se aplican. En éste contexto, algunas de las tareas que podemos aplicar son:

Stemming y lematización

La lematización y el *stemming* son ambos procesos en los que se reducen formas derivadas de una palabra, a una raíz que puede no ser una raíz real en el caso del stemming, y un lema real en el caso de la lematización.

En el contexto de la clasificación de texto, ambos son un proceso clave, considerando que las características en nuestro caso, son las palabras. Dependiendo del tamaño de nuestro conjunto de entrenamiento, puede ser conveniente normalizar éstas palabras para pulir un poco los resultadso

El algoritmo de *stemming* mejor conocido, es el algoritmo Porter. Funciona mediante detección de los sufijos de las palabras, y eliminándolos:

$$killing \rightarrow kill$$
$$killed \rightarrow kill$$
$$disappointment \rightarrow disappoint$$
$$disappointing \rightarrow disappoint$$
$$loved \rightarrow love$$
$$loving \rightarrow love$$

Figura 15: Resultados del algoritmo Porter en algunas palabras

La lematización funciona similar, pero con una diferencia: obtendremos siempre palabras (o lemas, mejor dicho) reales, mientras que el stemming puede devolver términos inexistentes.

El proceso para lematizar unas palabras con NLTK es el siguiente:

```
1  from nltk.stem import WordNetLemmatizer
2
3  lem = WordNetLemmatizer()
4
5  lem.lemmatize('dogs') --> dog
6  lem.lemmatize('feet') --> foot
7  lem.lemmatize('stemmed') --> stemmed
8  lem.lemmatize('stemmed', pos='v') --> stem
9  lem.lemmatize('better') --> better
10 lem.lemmatize('better', pos='a') --> good
```

Fijémonos en una parte interesante: cuando hemos lematizado *stemmed* y *good*, por defecto nos han devuelto la misma palabra: Eso se debe a que algunas palabras pueden funcionar como más de una categoría gramatical. Es aquí donde entra en juego el etiquetado gramatical, otro proceso en el cuál se trata de determinar la función sintáctica de cada palabra.

6.1.1. Semántica

Reconocimiento de entidades nombradas

Mejor conocido por sus siglas en inglés (NER, *named entity recognition*), tiene como objetivo detectar y clasificar entidades con nombre propio, dentro de categorías predefinidas como personas, organizaciones, monedas, etc...

Por ejemplo, en el el texto "Javier se gastó 25 euros en Zara", el resultado sería `Javier _[PERSONA] se gastó 25 euros _[MONEDA] en Zara_[ORGANIZACIÓN].`

En NLTK, el proceso de reconocimiento de entidades nombradas es bastante sencillo:

```
1  # pos_tag es la parte que se encarga de realizar el etiquetado gramatical
2
3  from nltk import word_tokenize, pos_tag, ne_chunk
4
5  print ne_chunk(pos_tag(word_tokenize('My friend Dave works for Microsoft
6  in Canada.')))
7
8  (S
9  My/PRP
10 friend/NN
11 (PERSON Dave/NNP)
12 works/VBZ
13 for/IN
14 (ORGANIZATION Microsoft/NNP)
15 in/IN
16 (GPE Canada/NNP))
```

7. Análisis de sentimiento

El análisis de sentimiento se refiere al procesado de lenguaje natural en el ámbito de identificar información subjetiva de un texto. En su versión más elemental, y la que llevaremos a cabo en este proyecto, determinaremos la polaridad de un twit, dicho en otras palabras, si tiene significado / connotación negativa o positiva.

Obviamente, el análisis de sentimiento no para ahí, si no que un nivel por encima podríamos detectar el estado emocional del autor.

7.1. Clasificación estadística

En el aprendizaje máquina y estadística [5], la clasificación es uno de los dos problemas quintesenciales, junto a la regresión. Consiste en determinar a qué categoría pertenece una nueva observación, apoyándose sobre datos existentes ya clasificados. Este hecho la convierte en una tarea de aprendizaje supervisado[6]. La tarea equivalente en el aprendizaje no-supervisado es la agrupación, que consiste en agrupar los datos existentes en base a las similitudes entre los puntos.

Dentro de la clasificación estadística, existen multitud de métodos, aunque para la tarea concreta que nos ocupa, el análisis de sentimiento (que en realidad, no es más que una simple tarea de clasificación de textos), destacan dos métodos:

- Clasificadores Bayesiano ingenuo (*Naive Bayes classifier*)
- Máquinas de vectores de soporte (*Support Vector Machines*)

Por su simplicitud, y popularidad como método base para la categorización de texto, emplearemos el clasificador bayesiano ingenuo para éste proyecto.

7.1.1. Clasificador Bayesiano ingenuo

Los clasificadores bayesianos ingenuos son una familia de algoritmos probabilísticos (que devuelven una distribución de probabilidad), basados en la aplicación del teorema de Bayes, asumiendo independencia entre las características (de ahí el apelativo de ingenuo.)

La fórmula del clasificador bayesiano ingenuo es la siguiente:

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)},$$

Figura 16: Fórmula del teorema de Bayes

Dónde:

1. A y B son eventos, y $P(B) \neq 0$
2. $P(A)$ y $P(B)$ son las probabilidades de observar A y B de forma independiente.
3. $P(A|B)$ es la probabilidad condicional de que se dé A (en nuestro caso, que el texto pertenezca a una clase), en caso de que B sea verdad.

La aplicación de ésta fórmula a un clasificador, resulta en lo siguiente:

$$p(C_k \mid x_1, \ldots, x_n) = \frac{1}{Z} p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)$$

Figura 17: Fórmula del teorema de Bayes

Dónde:

[5] El hecho de que un ordenador aprenda sin ser programado explícitamente
[6] Ver glosario

1. $p(C_k|x_1, ..., x_n)$ es la probabilidad de pertenecer a la clase C_k dadas las características $x_1, ..., x_n$
2. $Z = p(x)$ es un factor de escalado dependiente de las de las características.
3. $\prod_{i=1}^{n} p(x_i|C_k)$ es el producto de la secuencia de probabilidades condicionales de que se dé la característica x_i cuando C_k sea verdad (la clase).

Entrenamiento del modelo

El siguiente bloque de código muestra el proceso de entrenamiento del clasificador. También se facilita la Jupyter notebook debidamente comentada.

```
1
2  import TwitterSentimentAnalysis.TwitterTokenizer
3  import TwitterSentimentAnalysis.data_processing as dp
4  import nltk
5  import pickle
6  from nltk.tokenize import word_tokenize
7  from nltk.corpus import stopwords
8
9  df_train, df_test = dp.load_data()
10
11 df_train = df_train.replace('&quot;', "'", regex=True)
12 df_train = df_train.replace('&lt-+', "", regex=True)
13
14 df_test = df_test.replace('&quot;', "'", regex=True)
15 df_test = df_test.replace('&lt-+', "", regex=True)
16
17 df_train['SentimentText'] = df_train['SentimentText'].map(lambda x: x.strip())
18 df_test['SentimentText'] = df_test['SentimentText'].map(lambda x: x.strip())
19
20 all_words = dp.get_tweets_words(df_train)
21
22 stop_words = set(stopwords.words('english'))
23 features = dp.get_word_features(all_words)
24 filtered_features = [w for w in all_words if not w in stop_words]
25
26 train_tweets = dp.get_tweet_and_sentiment(df_train)
27 training_set = nltk.classify.apply_features(extract_features, train_tweets)
28
29 classifier = nltk.NaiveBayesClassifier.train(training_set)
30
31 print(classifier.show_most_informative_features(100))
```

8. Conclusiones

8.1. Objetivos del proyecto

Al principio del proyecto, nos marcamos los siguientes objetivos:

1. Estudiar y conocer las bases de datos NoSQL, en concreto Apache Cassandra
   1.1. Identificar los casos ideales para su uso.
2. Entender el diseño y modelado de datos en Apache Cassandra.
3. Almacenar Twits en Apache Cassandra en base a ciertos criterios de búsqueda. hagan referencia a un *hashtag*.
4. Entender y familiarizarse con el flujo de trabajo del científico de datos.
   4.1. Realizar análisis de sentimiento de los twits obtenidos mediante Python.

Creemos que podemos considerar explorados, a mayor o menor medida, la mayoría de los objetivos marcados. Si bien, por razones de empleo, el tiempo que le hemos podido emplear al proyecto ha sido bastante bajo. Por ésta razón, creemos que el proyecto no ha cumplido los objetivos a un nivel totalmente satisfactorio, y que se puede profundizar en muchos de las tareas, si bien consideramos que los conocimientos adquiridos y herramientas trabajadas durante el proyecto, nos serán de gran utilidad en el futuro.

Personalmente (Enrique) el análisis de datos y *data warehousing* son dos disciplinas que recientemente han despertado mi interés. En mi nuevo trabajo, muchos de los conceptos que hemos tratado aquí aparecen de vez en cuando, por lo que considero que lo aprendido en éste proyecto me ayudará a crecer profesionalmente, y a introducirme en conceptos que tarde o temprano acabaré encontrándome en mi carrera.

8.2. Próximos pasos del proyecto

En éste proyecto, hemos explorado a muy alto nivel tanto el uso de Apache Cassandra, como el flujo de trabajo de un científico de datos, todo en el marco del análisis de sentimiento de texto (clasificación.)

Teniendo en cuenta el punto hasta que el proyecto ha progresado, las áreas en las que pensamos seguir trabajando y estudiando son:

1. Complicar la arquitectura de Apache Cassandra: montar un clúster real, observar el funcionamiento de la consistencia configurable y *failover*.
2. Implementar un sistema de colas (Celery).
3. Implementar la infraestructura completa descrita en el documento.
   - En una de las reuniones que mantuvimos con profesionales en el mundo del *big data* y ciencia de los datos, se concibió la compleja arquitectura que se describe en el proyecto. Una de las partes en las que se hizo énfasis, fue en la interacción entre las partes del pipeline, concretamente entre la parte que obtenía datos de Twitter, las depositaba en Cassandra, y la otra que los cogía y realizaba el análisis de sentimiento. Se llegó a la conclusión de que sería ideal implementar un sistema de colas, que pusiera orden en la interacción entre las dos partes. No se ha llegado a explorar en el proyecto, así que es una de las
4. Estudiar y conocer más procesos y herramientas aplicables al científico de datos: desde lenguajes de programación (Julia, R, Scala), a herramientas y frameworks populares en la industria.
   - Procesos dentro del procesado del lenguaje natural: traducción máquina, análisis sintáctico y semántico...
   - Aprendizaje máquina: redes neurales, árboles de decisiones, deep learning...
   - Implementación y evaluación de más métodos de clasificación: máquinas de vectores de soporte lineales, y más.

9. Anexo: Estructura y instalación del proyecto

9.1. Estructura del proyecto

El árbol de directorios del proyecto está conformado por una estructura fácilmente adaptable a cualquier proyecto de análisis de datos o de Python, en general.

Aunque no hagamos uso de la totalidad de los directorios del mismo, creemos conveniente describir la estructura igualmente:

```
Projecte
├── data
│   └── twitter-sentiment-dataset.csv
├── environment.yml
├── figures
├── notebooks
│   ├── ProjecteFinal-Python.ipynb
│   └── ProjecteFinal-Cassandra.ipynb
├── output
│   └── naive_bayes_classifier.pickle
├── README.md
├── setup.py
└── TwitterSentimentAnalysis
    ├── manager
    ├── models
    ├── data_processing.py
    ├── __init__.py
    ├── regexps.py
    └── TwitterTokenizer.py
```

- En el directorio **data**, incluiremos todos los ficheros de datos a procesar. En nuestro caso, se trata de un CSV que contiene twits ya marcados con el sentimiento.
- El fichero **environment.yml** se utiliza para definir las dependencias del entorno virtual (venv) de Python. En él podremos definir tanto dependencias que incluya Anaconda de por sí, como indicarle que instale nuevas desde pip.
- El directorio **figures** incluiría gráficos y plots, si generásemos alguno.
- En **notebooks** almacenaremos las libretas de Jupyter correspondientes al proyecto. Las libretas de Jupyter son ficheros basados en JSON, que combinan tanto celdas de texto (escritas en Markdown, que luego se renderiza), como de código ejecutable. Éste se ejecuta celda por celda, cuando el usuario decide ejecutarlas. El hecho de poder controlar la ejecución de las celdas, junto a la posibilidad de combinarlas con Markdown, las hace especialmente ideales para describir código en una presentación o seminario.

  En el caso de nuestro proyecto, incluimos dos libretas:

1. ProjecteFinal-Python: En ésta libreta describimos el proceso de preparado de los datos y el posterior análisis de sentimiento.
2. Cassandra: Aquí se describe el proceso de interacción con Apache Cassandra desde Python.

- El directorio **output** contiene ficheros resultantes del procesado. Por ejemplo, el clasificador ya entrenado en formato Pickle, y un fichero CSV que contiene los resultados del análisis de sentimiento de twits previamente desconocidos.
- El fichero setup.py nos permitirá instalar las dependencias del virtualenv una vez instalado, mediante la instrucción `python setup.py install`
- Dentro de TwitterSentimentAnalysis tenemos nuestro código fuente, comprendido en los siguientes ficheros:
  - La declaración del módulo en __init__.py. Ésto nos permitirá importar el código en las libretas de Jupyter.
  - texto durante el proceso de preparado de los datos.
  - data_processing.py incluye los métodos mediante los que trataremos el texto durante el proceso de preparado de los datos.
  - TwitterTokenizer.py es una clase personalizada mediante la cual separaremos los twits en Tokens. Incluye expresiones regulares para tratar correctamente elementos comunes en twits como las menciones (@usuario), hashtags (#hashtag) y emoticonos.
  - Manager y Models contienen el código de GetOldTweets, un proyecto en Github que utilizamos para extraer los twits.

9.2. Instalación del proyecto

Las siguientes instrucciones detallan cómo instalar el proyecto, para el propósito de ejecutar correctamente las libretas de Jupyter.

1. Instalar la plataforma de análisis de datos basada en Python Anaconda, desde `https://www.continuum.io/downloads`. Para el proyecto se ha utilizado la versión que emplea Python 3.6. Durante la instalación, Anaconda preguntará si queremos añadir el directorio de sus binarios al principio del PATH. Si no se hace, recomendamos que al menos esté al final para acceder con facilidad al programa principal de Anaconda (conda).
2. Descargar el proyecto / clonar el repositorio de Github
3. Dentro del directorio raíz del proyecto, crear el venv, y activarlo cada vez que se quiera trabajar con él:

Creando el entorno

```
conda env create -f environment.yml
```

Activando el entorno

```
# El nombre será el definido dentro del fichero environment.yml

source activate nombre-del-venv
```

4. Una vez activado el venv, si deseamos instalar nuevos paquetes mediante pip, podemos hacerlo añadiéndolos al environment.yml y actualizando el entorno de la siguiente manera:

Actualizando el entorno

```
conda env update environment.yml
```

5. Si se hacen cambios en el código fuente de nuestro módulo, será necesario "volverlo a instalar", para que el venv reconozca los cambios. En la raíz del proyecto:

Actualizando cambios en el código.

```
python3 setup.py install
```

6. Cuando tengamos ya el entorno virtual activo, todo lo que ejecutemos en relación a Python, estará aislado de la instalación nativa del sistema: el intérprete incluso. Ésta es la característica principal de los entornos virtuales: la posibilidad de definir entornos aislados entre sí, cada uno con versiones específicas de los paquetes.

Antes de entrar a la libreta, deberemos cerciorarnos de que el entorno virtual que acabamos de configurar está disponible para su uso en la libreta de Jupyter. Para ello, utilizaremos el paquete iPykernel que hemos especificado en las dependencias:

Configurando el kernel en Jupyter.

```
# Nombre especificado en environment.yml
python -m ipykernel install --user --name nombre-venv --display-name
"Python (mi entorno)"
```

Con el entorno activo, sólo hará falta ejecutar `jupyter notebook` para que se abra en el navegador la ventana principal del programa, quedando únicamente navegar al directorio dónde tengamos las libretas en cuestión, abrirla, y seleccionar el kernel que acabamos de configurar en el menú superior, en Kernel >Change Kernel.

10. Anexo: tipos de datos en CQL

La siguiente tabla muestra los tipos de datos admitidos en CQL:

Tipo CQL	Valor	Descripción
ascii	Cadena	Cadena de carácteres en ASCII-US
bigint	Enteros	Long de 64 bits con signo
blob	Blob binario	Bytes arbitrarios, expresados en hexadecimal
boolean	Booleano	True / False
counter	Entero	Contador distribuido de 64 bits
decimal	Entero / flotante	Decimal de precisión variable
double	Entero / flotante	Flotante de 64 bits (IEE-754)
float	Entero / flotante	Flotante de 32 bits (IEE-754)
frozen	Tuplas, colecciones, tipos definidos	Serialización de múltiples valores en un único valor (blob)
inet	Cadena	Dirección IPv4 o IPv6.
int	Entero	Entero de 32 bits con signo.
list	Lista	Colección ordenada de elementos.
map	Mapa	Array estilo JSON de claves-valores.
set	Conjunto	Conjunto de elementos.
text	Cadenas	Cadenas UTF-8
timestamp	Enteros / cadenas	Fecha y hora, encodeada como 8 bits desde la época Unix.
timeuuid	uuid	UUID de tipo 1
tple	Tupla	Grupo de dos a tres campos.
UUID	UUID	Valor UUID estándar
varchar	Cadena	Cadena UTF-8
varint	entero	Entero de precisión arbitraria

Cuadro 4: Tipos de datos CQL

Glosario

aprendizaje no supervisado se dice de aquellas tareas de aprendizaje máquina destinadas a describir estructuras en aquellos datasets sin clasificar..

aprendizaje supervisado se dice de aquellas tareas de aprendizaje máquina en las que un dataset con observaciones previamente clasificadas está disponible..

escalado horizontal se dice del mecanismo de escalado en el que el rendimiento de un sistema se aumenta añadiendo máquinas adicionales..

escalado vertical se dice del mecanismo de escalado en el que el rendimiento de un sistema se aumenta añadiéndole más recursos de hardware a una máquina existente..

hashtag texto precedido por una almohadilla, representando una etiqueta que ayuda a encontrar mensajes que hablen del mismo tema.

variable categórica se dice de una variable que puede tomar un valor de un juego limitado de valores.

variable contínua se dice de una variable numérica o de fecha, que tiene infinitos valores entre dos valores determinados.

variable discreta se dice de una variable numérica que tiene un número contable de valores entre dos valores determinadods.

Referencias

Codd, Edgar F. (1970). “A Relational Model of Data for Large Shared Data Banks”. En: *Commun. ACM* 13.6, págs. 377-387. ISSN: 0001-0782. DOI: 10.1145/362384.362685. URL: http://doi.acm.org/10.1145/362384.362685.

Hayashi, Chikio (1998). “What is Data Science ? Fundamental Concepts and a Heuristic Example”. En: *Data Science, Classification, and Related Methods: Proceedings of the Fifth Conference of the International Federation of Classification Societies (IFCS-96), Kobe, Japan, March 27–30, 1996*. Ed. por Chikio Hayashi y col. Tokyo: Springer Japan, págs. 40-51. ISBN: 978-4-431-65950-1. DOI: 10.1007/978-4-431-65950-1_3. URL: http://dx.doi.org/10.1007/978-4-431-65950-1_3.

Yanyong, Zhu y Xiong Yun (2011). *Datalogy and Data Science: Up to Now.* En la web, visitado el 21 de marzo de 2017. URL: www.paper.edu.cn/en_releasepaper/content/4432156.

Wickham, Hadley (2014). “Tidy data”. En: *The Journal of Statistical Software* 59 (10). URL: http://www.jstatsoft.org/v59/i10/.

Carpenter, Jeff y Eben Hewitt (2016). *Cassandra: The Definitive Guide*. 2.ª ed. O’Reilly Media. ISBN: 3257227892.

Smith, F. Jack (2006). “Data science as an academic discipline”. En: *Data Science Journal* 5, págs. 163-164. DOI: 10.2481/dsj.5.163.

Atkinson, Malcolm y col. (1992). “Building an Object-oriented Database System”. En: ed. por François Bancilhon, Claude Delobel y Paris Kanellakis. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. Cap. The Object-oriented Database System Manifesto, págs. 1-20. ISBN: 1-55860-169-4. URL: http://dl.acm.org/citation.cfm?id=140592.140595.