Máster en Data Science. URJC
Sistemas Distribuidos de Procesamiento de Datos

Análisis de sentimientos con Hadoop

Javier Llorente Mañas
Carlos Sánchez Vega

2017

Índice

0.1 Introducción

El objetivo de este documento es mostrar cada una de las partes de la que está compuesto el proyecto (código desarrollado, pasos para su correcta ejecución y las conclusiones). Más concretamente, nos hemos centrado en el *análisis de sentimientos* relativos al partido de *El Clásico*, es decir, del Real Madrid contra el Barcelona. Para realizar ese análisis se ha utilizado el fichero *AFINN-111.txt*, que contiene palabras en inglés valoradas de menos cinco (negativo) a más cinco (positivo). Las palabras de dicho fichero fueron introducidas manualmente por Finn Årup Nielsen in 2009-2011.

0.2 Secciones de la memoria

- Apartados resueltos de la práctica
- Obtención de datos
- Análisis de datos
- Representación de los datos
- Limitaciones
- Comentarios sobre la práctica

0.3 Apartados resueltos de la práctica

Hemos resueltos **todos** los apartados de la práctica, incluidos los opcionales.

0.4 Obtención de datos

Nos hemos basado en el script de python para obtener los tweets. No obstante, hemos "mejorado" la version previa para automatizar la toma de tweets (fichero twitter.py, función (*streamingapi*).

```
twitter = twitter(api_key,api_secret,api_token,api_token_secret)
#we are going to retrieve data about #ElClasico.
twitter.streamingapi(['Elclasico','elclásico','ElClasico','elclasico'], 'elclasico21102017'
,'twitterjavierllorente',0.8)
```

En dicha función, se toman tweets durante el periodo indicado por el parámetro *number_of_days* y se almacenan, en la cuenta de S3, en el bucket indicado por *bucket*. De esta manera, el comportamiento del programa es mucho más agil y nos podemos despreocupar del momento en el cual queremos parar su ejecución. En cuanto a la búsqueda de los datos, se ha usado el script de búsqueda *search.py*, que recoge los tweets de *'Elclasico'*,*'elclásico'*,*'ElClasico'* y *'elclasico*':

La región en la cual se tomarán los tweets es EEUU en inglés descartando, por tanto, los de cualquier otra región o idioma. Además, como cada tweet contiene mucha información que para nosotros es irrelevante, hemos almacenado sólo aquellas partes que tienen interés de cara a las problemáticas que se plantean en cada uno de los apartados.

0.4.1 Análisis de los resultados

Para realizar el análisis de los resultados, hemos usado *mrjobplaces.py*. Concreamente, hemos decidido escoger el framework de MRJob para la realización de la tarea. En un principio se planteó la dificultad de la codificación del lenguaje en los tweets que recogíamos, puesto que habían signos de puntuación que dificultaban el análisis de los términos. Por ello, la lectura del fichero json de tweets se codificó con un encoding *utf-32*, tal y como se muestra a continuación:

```
@staticmethod
def filter_decode_tweets(line):
    try:
        tweet_object = json.loads(line, encoding='utf-32')
        return tweet_object
    except ValueError:
        return None
```

En cuanto al planteamiento del problema, hemos tenido que usar *steps* o pasos, en los que hemos resuelto cada uno de los apartados:

```
def steps(self):
    return [
        MRStep(mapper=self.mapper_english,
               combiner=self.combiner,
               reducer=self.reducer_per_type_of_object),
        MRStep(reducer=self.reducer_top_state_and_Hastags)
    ]
```

- Mapper (self.mapper_english): la explicación del mapper se podría descomponer en las siguientes partes:

Al inicio del mapper, se creará un diccionario con las palabras del fichero *AFINN-111.txt* junto con su valoración (desarrollo incluido en el fichero *diccionaries.py*)

Posteriormente, por cada tweet recibido se procesarán solo quellos tweets generados en EEUU en inglés:

```
if 'lang' in tweet_object and tweet_object['lang'] == 'en':
```

Además, el tweet se descompondrá en palabras: En caso de que la palabra sea un hashtag, se generará un par *yield (word, 1)*. En otro caso, se consultará la valoración de la palabra en el diccionario anteriormente mencionado. Para la localización de los tweets hemos distinguidos dos situaciones:

1) La localización viene de la forma *[country_code][estado]*
2) La localización viene con el tag *['geo']*, caso en el cual se calculará la región en la que se han generado mediante el **cálculo de polígonos** y las coordenadas del tweet. Una bez que se ha procesado la localización, se generará un par *yield (state, tweet_value)* con el Estado y el valor asociado a dicha palabra según *AFINN-111.txt*.

- Combiner (self.combiner): Se hará la suma de los valores asociados a cada clave:

```
def combiner(self, key, value):
    yield (key, sum(value))
```

- Reducer (self.reducer_per_type_of_object): Se calcula la suma de los valores asociados a los siguientes dos casos:

```
def reducer_per_type_of_object(self, key, value):
    hastag_sentiment = (sum(value), key)
    if '#' in key:
        yield ('#', hastag_sentiment)
    else:
        yield ('location', hastag_sentiment)
```

Caso 1 [hashtag][suma de veces que aparece el hashtag]

Caso 2 [hashtag][suma de las valoraciones de las palabras generadas en ese listado].

- Reducer (self.reducer_top_state_and_Hastags): se ordenan los estados, del que tiene mejores valoraciones al que tiene peor valoración. Posteriormente, se muestra el Estado con valoración más positiva y se muestran por pantalla los 10 primeros hashtags con mayor número de ocurrencias.

0.4.2 Representación de los datos

Para la representación de los datos se ha decidido usar un mapa, en el que se puede ver de forma clara cuáles serán los estados que cuentan con una mayor representación de tweets positos relativos al "clásico" del Real Madrid.

TODO

0.4.3 Resultados de la ejecución en los diferentes entornos

1. Local: `python mrjobplaces.py tweets.json111`
2. EMR : `python mrjobplaces.py -r emr s3://urjc.datascience.data/tweets/tweets.json111 --ec2-instance-type c1.medium --num-core-instances 3 --output-dir=s3://urjc.datascience.jon/output14D`

0.5 Limitaciones

A pesar de la rapidez de ejecución gracias al procesamiento distribuido, hay algunas valoraciones a tomar en cuenta: el cálculo semántico se realiza conforme a las palabras y no existe el concepto de *evento*. Por tanto, una posible mejora sería normalizar todas las referencias a este partido ya que, ahora mismo, para el cálculo de los scores no sería lo mismo *elClasico* que *elPartidodelSiglo*, que *RealMadridVSBarcelona*, que *RealMadridVSBarça*...Además, no se distinguirían aquellas referencias a la ciudad en los tweets recogidos. Es decir, puede que en el tweet de *elClasico* se hagan valoraciones relativas a la ciudad de Barcelona o Madrid que no tienen por qué guardar relación con los equipos. Por otro lado, otros conceptos más complejos del lenguaje natural no han sido contemplados, como la doble negación.Por ejemplo, algunas expresiones como *not too bad* serían categorizados como negativos por contener la palabra *bad*, siendo su significado totalmente contrario. Otro ejemplo podría ser *very good* ya que, tal y como se realizan las valoraciones actualmente, tendría el mismo score que *good* siendo el primero de los conceptos superior.

0.6 Comentarios sobre la práctica

Connsideramos que se logrado varias mejoras en la ejecución de los scripts (se ha automatizado la toma de tweets durante el periodo de tiempo deseado, almacenando en un bucket los resultado) y hemos logrado implementar **todos** los apartados con éxito. Sin embargo, nos hemos encontrado con varias dificultades: Por un lado, la codificación de los ficheros, pues cuando el *tweet* contenía emoticonos se representaban de forma errónea si no eran tratados con la codificación correcta. Además, tuvimos que tener en cuenta los signos de puntuación y abreviaturas, debido a los apóstrofes. Por otro lado, nos resultó difícil realizar todos los puntos en el mismo programa, es decir, combinarlo de tal manera que la ejecución de todas las partes fuera compatible.