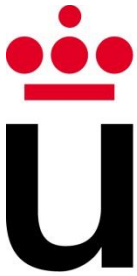

Universidad Rey Juan Carlos

MÁSTER EN DATA SCIENTE

SISTEMAS DISTRIBUIDOS DE PROCESAMIENTO DE DATOS I

Curso académico 17/18

TWITTER SENTIMENTAL ANALYSIS

Autor: Ester Cortés García

Índice

1. Descripción del código

A continuación, se procede a la descripción del código del fichero MRJobTwitter.py, que realiza la analítica de los tweets descargados mediante twitterstream.py.

1.1. Imports

Para la realización de la práctica ha sido necesario descargar diversas librerías:

- mrjob.job.MRJob, para poder realizar los trabajos de Map Reduce mediante MRJob.
- mrjob.step.MRStep, para poder definir varios pasos en el proceso.
- mrjob.protocol.JSONProtocol, para poder indicar que el protocolo interno se corresponde al formato JSon.
- json, para poder cargar los tweets en este formato
- sys, para poder coger los ficheros necesarios de la ruta actual
- operator.itemgetter, para poder ordenar tuplas en función de una clave numérica
- re, para el uso de expresiones regulares mediante patrones

1.2. Main

En la función main simplemente se hace un llamamiento a la clase MRTwitterSentimentalAnalysis para poder ejecutarla.

1.3. Class MRTwitterSentimentalAnalysis

En esta parte del código es donde se realiza todo el proceso de analítica de los tweets descargados.

En primer lugar se define el protocolo interno que emplearán los mappers y reducers.

En segundo lugar, se define el mapper inicial, donde se crean distintas variables globales necesarias para el proceso.

- english_dictionary, donde se llama a un método privado para cargar el diccionario dado por el profesor.
- pattern, para definir el patrón que se aplicará a la hora de leer el texto de los tweets para facilitar la detección de palabras y eliminar caracteres no válidos
- states, diccionario formado por los estados de Estados Unidos. Se empleará para detectar si es válida la localización de un tweet.

A continuación, figuran varios métodos privados que usan internamente algunos de los mapper y reducers.

- load_dictionary, método empleado para leer el fichero de texto 'AFINN-111.txt', que corresponde al diccionario de palabras en inglés con sus valores asociados, y cargarlo en un diccionario para usarlo en el programa.
- tweet_state, método que recibe la localización del tweet mediante el campo "full_name" del tweet. Esta localización puede estar expresada de dos formas distintas:
    - (Ciudad,Abreviatura del estado). En este caso, se comprobará si la abreviatura del estado se encuentra entre los valores del diccionario de estados y, si es así, se devolverá.
    - (Nombre del estado,USA). En este caso, se buscará el nombre del estado en el diccionario de estados y, si la búsqueda de esta clave proporciona un valor, se devolverá dicho valor.

  En ambos casos, nos quedamos con la abreviatura del estado para trabajar posteriormente y pasarla como clave en algunos de los mappers.

- tweet_word_punctuation, recibe el texto del tweet y el diccionario en inglés, pasa el texto recibido a minúsculas y se queda, mediante la aplicación del patrón definido anteriormente, con los caracteres válidos. Con esto, tendremos una lista con las palabras detectadas en el texto. A continuación, se van comparando con las palabras del diccionario. Si alguna coincide, se va sumando o restando su valor a una variable de puntuación, que será la que se retorne una vez procesado todo el texto.

Tras todo esto, tenemos los mappers y reducers necesarios para la analítica.

- mapper_tweet_filtering, se encarga de ir leyendo los tweets del fichero json y los va filtrando en función de distintos aspectos como son la existencia de la palabra 'text', para eliminar tweets borrados, el idioma ('en') y el país ('us'). Si los tweets cumplen estos tres requisitos, se procede a comprobar la localización exacta y a dar una puntuación al texto mediante los métodos privados explicados anteriormente. Finalmente, de cada tweet nos quedamos con la abreviatura del estado, la puntuación y los hashtags (necesarios para realizar la segunda parte opcional), que se pasarán al reducer usando como clave el estado (para poder posteriormente conseguir la puntuación total de mismo) y como valor un diccionario con los aspectos comentados anteriormente.
- reducer_state_punctuation, recibe todos los tweets con la misma clave, en nuestro caso, pertenecientes al mismo estado. Lo que hace es, por cada tweet que recibe, acumular la puntuación y los hashtags. Una vez procesados todos los tweets, divide la puntuación total entre el número de tweets para obtener una media. Se devuelve como clave el estado y como valor su información

- mapper_happiest_state, simplemente sirve para cambiar el formato de entrada de los datos. Recibe un estado como clave y una información como valor y devolverá una clave nula y una tupla como valor, formada por el estado y la información. El objetivo de que la clave sea nula es que todos los resultados de este mapper vayan a un mismo reducer para poder tener a nuestra disposición todos los datos. Este mapper podría eliminarse si se cambian los datos retornados por el anterior reducer al formato que se adopta aquí. También tiene una utilidad personal para localizar en el código dónde empieza la primera parte optativa.
- reducer_happiest_state, recibe tuplas con todos los estados y su información. Se encarga de ir comparando la puntuación que le llega con la que tiene almacenada; si la supera, se guarda la nueva puntuación junto con el nombre del estado al que le corresponde. Aparte, acumula todos los hashtags de todos los estados con el fin de realizar la segunda parte optativa. A continuación, recorre la lista de los hashtags para poder pasarlos como claves al próximo mapper. Como valor se pasa la información del estado más feliz y un contador inicializado a 1, para poder sumar posteriormente el número de ocurrencias de cada hashtag.
- mapper_hashtag_treatment, recibe un hashtag y su información asociada y devuelve exactamente lo mismo. Tiene simplemente una utilidad personal para saber dónde comienza la segunda parte opcional.
- reducer_hashtag_treatment, recibe todos los hashtags iguales y su información. Tiene un contador en el que se va sumando el número de hastags. Una vez finalizado el recuento, se realiza un cast a entero del contador para poder realizar una ordenación en el siguiente reducer. Devuelve una clave None (para que vayan todos los datos al mismo reducer) y un valor con la información necesaria.
- reducer_trending_topic, recorre todos los hashtags que recibe para poder quedarse con los 10 que tengan un mayor recuento, para ello, según va recibiendo los almacena en un array de longitud 10 y los ordena en función del recuento. Una vez que la ocupación del array esté completa, como el array estará siempre ordenado, se va comprobando con la última posición los nuevos hashtags, si el recuento es mayor, se elimina el elemento de la última posición y se añade el nuevo hashtag. Finalmente, devolverá un diccionario con distintos campos (la lista de los 10 hashtags más importantes junto con su recuento, la abreviatura del estado más feliz y su puntuación).

Por último, se encuentra la definición de los pasos, que son los siguientes:

- mapper_initial – mapper_tweet_filtering – reducer_state_punctuation
- mapper_happiest_state – reducer_happiest_state
- mapper_hashtag_treatment – reducer_hashtag_treatment
- reducer_trending_topic

2. Formas de ejecución

En principio, la ejecución del código se llevó a cabo en la máquina virtual de Cloudera pero, a la hora de realizar la ejecución mediante hadoop, me surgieron varios problemas en cuanto a permisos. No me dejaba acceder a los archivos que subí a hdfs, así que por recomendación de algunos compañeros que habían realizado la práctica con la imagen de Cloudera de Docker, cambié mi entorno de ejecución.

Para ejecutar en la imagen de Cloudera, lo primero es tener Docker instalado, en mi caso en Ubuntu. Después hay que bajarse la imagen de Cloudera y copiar los ficheros necesarios en ella (MRJobTwitter, AFINN-111.txt y los diversos .json con los distintos volúmenes de tweets). A continuación, ejecutaremos la imagen, obteniendo como resultado acceso a la consola de Cloudera. Una vez ahí, podremos ejecutar como lo haríamos desde la terminal de la máquina virtual de Cloudera.

2.1. Ejecución inline

Método usado para hacer debug y testeo. Con él he ido probando el código mientras lo realizaba hasta que todo funcionó perfectamente. La manera de lanzarlo es la siguiente:

python MRJobTwitter.py -r inline tweets.json –file AFINN-111.txt

2.2. Ejecución local

Este método es similar al anterior pero si que genera distintos subprocesos por cada tarea. Se lanza de la siguiente forma:

python MRJobTwitter.py -r local tweets.json –file AFINN-111.txt

2.3. Ejecución en Hadoop

Como ya he comentado anteriormente, a la hora de ejecutar en hadoop ocurrieron numerosos problemas relacionados con los permisos de hdfs: no me permitía acceder a las rutas donde tenía mis archivos guardados, imposibilitando la llamada de dichos archivos para lanzar este método.

La elección de este tipo de ejecución sobre EMR simplemente está basada en una mayor simplicidad del proceso, ya que con hadoop la única diferencia respecto a los métodos anteriores era el cambio de una palabra en el comando y subir un fichero a hdfs. Tras los problemas de permisos me planteé el cambio de método pero, tras la recomendación de diversos compañeros que les había pasado lo mismo, me decanté por Docker, ya que lo tenía instalado en mi máquina Ubuntu junto la imagen de Cloudera.

Para ejecutar con hadoop, lo primero que hay que hacer es subir el archivo de entrada a hdfs mediante el mandato hdfs dfs -put tweets.json /practica (habiendo creado previamente el directorio practica en hdfs). A continuación, se lanza este mandato:

python MRJobTwitter.py -r hadoop hdfs:///usr/root/practica/tweets.json –file AFINN-111.txt

2.4. Resultados

Los resultados obtenidos tras la ejecución de tweets6.json (8,4GB) son los siguientes:

“data”{‘trending_topic’:[[‘job’,421],[‘CareerArc’,352],[‘Hiring’,313],[‘hiring’,184],[‘Job’,130],[‘Jobs’,130],[‘Retail’,60],[‘Nursing’,32],[‘Hospitality’,26],[Transportation’,24]],’happiest_state’:’ID’,’punctuation’:1.3846153846}

3. Evaluación de escalabilidad

Para el volumen de datos conseguidos mediante twitterstream.py (máximo 8GB en el espacio de 12 horas) y según la solución planteada para el cálculo del estado más feliz y los 10 hashtags más usados, no es necesario ejecutar el proceso en un entorno distribuido. La ejecución es muchísimo más lenta y no aporta ninguna mejora ni resulta más eficiente.

Un entorno distribuido será mucho más útil cuando tenemos un gran volumen de datos, datos que se estén actualizando constantemente, o ambos. En nuestro caso, no se cumple ninguna de estas condiciones.

4. Comentarios personales

Esta práctica me ha resultado de gran utilidad para afianzar los conceptos explicados en clase. La dificultado no ha sido elevada, ya que los ejemplos proporcionados por el profesor se podían aplicar en este caso y, el único problema grave que he tenido estaba relacionado con un tema de permisos al que ha sido fácil encontrar una alternativa.

También considero esta práctica especialmente interesante porque se trata de una aplicación real del conocimiento adquirido y porque estaba relacionada con otras dos asignaturas del máster, ‘Obtención de Datos’, por el uso de la API de Twitter, tema que nos explicaron en dicha asignatura, y ‘Arquitecturas en la nube’, ya que gracias a los conocimientos que adquirí en dicha asignatura he podido emplear Docker sin problemas para resolver mi problema.