Programming with ElasticSearch

23-9-2022

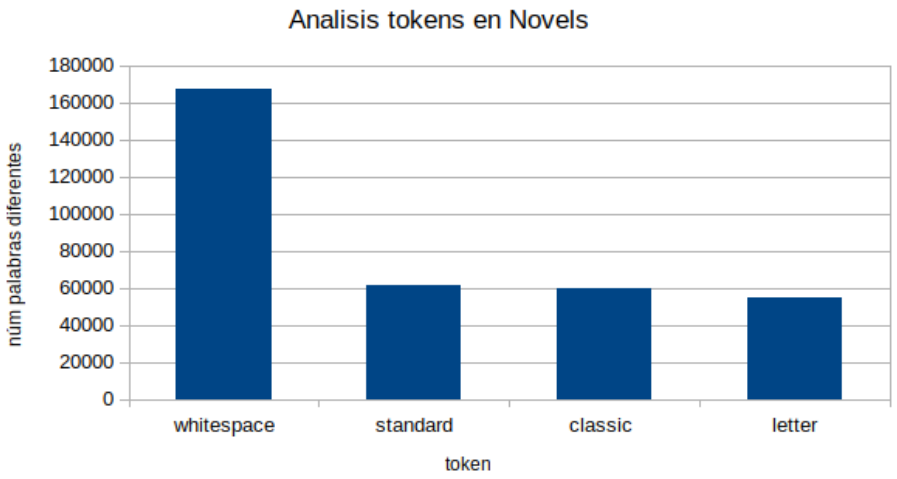
He Chen

Daniel Muñoz

Grupo 12

1. The index reloaded

Para ver la diferencia entre los diferentes tokens, hemos utilizado el CountWords.py junto con el token a analizar para ver el número de palabras diferentes y hemos obtenido los siguientes resultados:



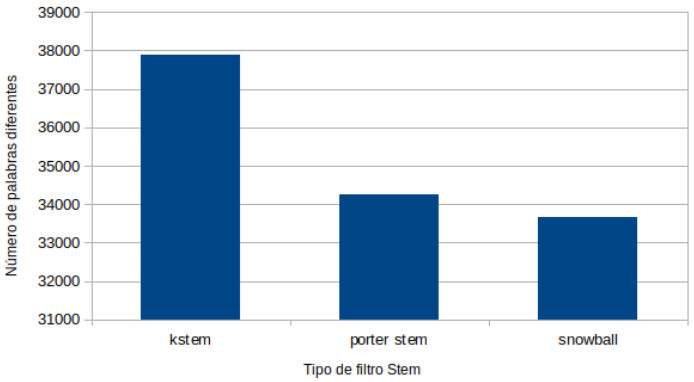
Como podemos ver el token más agresivo es el letter. Este token divide el texto en términos cada vez que encuentra un carácter que no es una letra.
Mirando las palabras, hemos visto que hay muchas que son una combinación de números, signos de puntuación y caracteres que no son letras como '_' o '-' entre otros, así que tiene sentido que sea este el que menos palabras tenga ya que es el más restrictivo.

Para los filtros, probaremos con *lowercase*, *asciifolding* y *stop* en este orden. Es importante que estén en este orden ya que a la hora de aplicar los filtros, si se cambian puede influir en el proceso, ya que no es lo mismo, quitar las stop words y después pasar todo a minúsculas que pasarlo todo a minusculas y después quitar las stopwords, ya que la lista de las stop words están todas en minúsculas y si hay alguna en el texto con letras en mayúsculas, estas no se quitan y despues se pasarían a minúsculas y no se filtrarán, en cambio si se pasa a minúsculas y después se eliminan, se eliminarán todas las stop words.

Creemos que estos tres filtros deberían estar, ya que para comparar dos temas, deberían estar con las palabras que tengan significado (con el token *letter* y los filtros *lowercase* y *asciifolding*) y que no sean comunes (quitando stopwords, ya que estas suelen estar en todos los textos).

Para escoger uno de los tres stems, probaremos con los ficheros de *novels*, el token y los filtros anteriores y con un filtro extra que será el stem que queramos utilizar. Despues utilizaremos el *Countingwords.py* para contar el número de palabras,

El filtro stem lo que pretende hacer es buscar la raíz de las palabras y se supone que mientras mejor lo haga habrá más palabras con las misma raíz y por lo tanto menos palabras diferentes.



Como podemos observar, el que menos palabras diferentes tiene es el snowball, así que como hemos comentado, escogeremos este, ya que consideramos que es el mejor de los tres.

Analizando el código de IndexFilesPreprocess.py, las configuraciones del analizador de texto se especifican con el comando parser.add_argument en la función main del script.

```
parser.add_argument('--token', default='standard', choices=['standard',
'whitespace', 'classic', 'letter'], help='Text tokenizer')
```

En default se le puede cambiar el tokenizer por defecto y se le pueden añadir más opciones en el vector que aparece en choices. Las opciones disponibles del elastic search estan explicadas en esta página:
https://www.elastic.co/guide/en/elasticsearch/reference/master/analysis-tokenizers.html
En esta salen todos los tokens que se pueden utilizar y que hace cada uno de ellos.

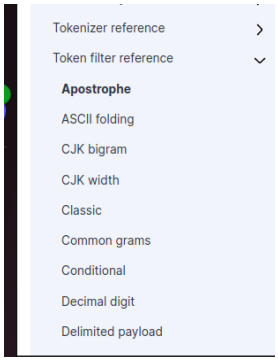
```
parser.add_argument('--filter', default=['lowercase'],
nargs=argparse.REMAINDER, help='Text filter: lowercase, '
'asciifolding, stop, porter_stem, kstem, snowball')
```

En el caso de filtros, para anadir una opción de filtrado hay que modificar la lista que verifica que los filtros sean válidos y instalar el plugin correspondiente al filtro, si no esta, lanza un error, así que para ello se han de incluir en la lista y como parámetros al final del comando con la opción --filter con los filtros que queramos utilizar.

```
# check if the filters are valid
for f in args.filter:
     if f not in ['lowercase', 'asciifolding', 'stop', 'stemmer',
'porter_stem', 'kstem', 'snowball', 'phonetic']:
          raise NameError( 'Invalid filter must be a subset of:
lowercase, asciifolding, stop, porter_stem, kstem, snowball, phonetic')
```

Por ejemplo, aquí hemos permitido la opción *phonetic*. Para poder usar esta opcion, primero debes instalar el plugin, puedes encontrar las instrucciones de instalacion en:

https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-phonetic-tokenfilter.html



Los diferentes filtros aparecen en la columna de la izquierda de la página anterior, “desplegando token filter reference”, se pueden ver todas las opciones disponibles de los diferentes filtros que hay.

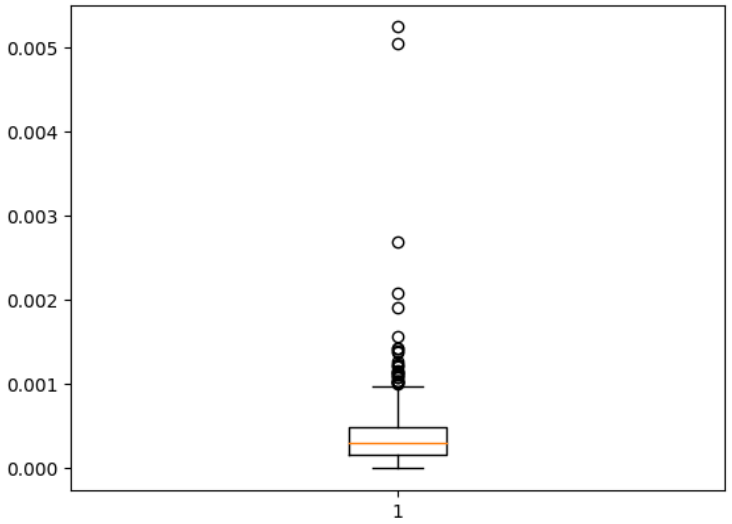
2. Computing tf-idf ’s and cosine similarity

En esta sección completamos el escript que computa el cosine similarity que hay entre dos documentos.
En la función principal del script, podemos ver el proceso de cómputo. Después de encontrar los identificadores de los dos ficheros, calculamos sus respectivos vectores TF-IDF para al final calcular el cosine sililarity.
Para hacer el cómputo tenemos que completar las funciones *toTFIDF, normalize*, *print_term_weigth_vector*, *cosine similarity* esmentado en el enunciado de la práctica.
Para implementar las funciones *toTFIDF* y *normalize* solo hay que aplicar las fórmulas. La funcion *print_term_weigth_vector* simplemente es imprimir un vector. Finalmente la función *cosine_similarity* es la más difícil de implementar. Aprovechando vectores están ordenados alfabéticamente, hemos inicializado dos variables índices que apuntan a los elementos de cada vector, para cada iteración se decide si hay que acumular el valor de multiplicar dos elementos del vector (si apuntan la misma palabra) y avanzar los dos índices, o avanzar un índice (el menor de las dos palabras en orden alfabético). De esta manera conseguimos implementar la función con un coste linear.

3. Experimentacion

Hemos acabado el código del script *TDFIDFViewer.py* como indicaba el pdf y para comprobar si estaba bien, hemos probado la similaridad entre un fichero con él mismo mediante la similaridad del coseno y nos ha salido con similaridad = 1. Esto tiene sentido, ya que al hacer la similaridad del coseno, el ángulo entre un fichero y él mismo es 0 y el cos(0) = 1. Así que parece que está bien implementado.

Para el experimento, lo que haremos será comparar dos temas que aparentemente no tienen nada que ver (beisbol y mac hardward) para ver si la similaridad del coseno, en media, es prácticamente 0.
Para compararlos, teniendo en cuenta que son textos muy cortos, lo que hemos hecho es escoger uno a uno los ficheros de béisbol (r*ec.sport.baseball)* y compararlos con cada uno de los ficheros de hardware (*comp.sys.mac.hardware*) mediante el script de python *TDFIDFViewer.py* que calcula la similitud del coseno. Para cada fichero de béisbol, haremos una media de los resultados de la similaridad del coseno al comparar un fichero de baseball con todos los de mac hardware. En la carpeta *rec.sport.baseball* y en la de comp.sys.mac.hardware hay unos mil ficheros de datos en cada una, como el coste temporal de computar una media era bastante elevado, al final computamos 676 medias para el análisis. Para hacer este cómputo, modificamos el main del script *TDFIDFViewer.py*, puedes ver que las modificaciones están comentadas.



Hemos creado un script *Make_Boxplot.py* para hacer el boxplot con los resultados obtenidos después de ejecutar *TDFIDFViewer.py.* Guardamos el resultado en un txt, para que *Make_Boxplot.py* lo lea y genere el boxplot.
Como se puede observar en el boxplot la media de similaridad es de unos 0,0004, incluso los dos outliers más grandes, el máximo valor que dan es de unos 0,0055.
Tanto la media como los dos outliers más grandes, tienen un valor de similaridad muy bajos, así que si que es cierto que los foros, los que hablan del hardware del mac no tiene nada que ver con los que hablan del béisbol. Esto puede ser debido a que al ser dos temas muy diferentes, tienen palabras específicas diferentes y por eso no se parecen en nada.

Como explica el enunciado, si tokenizamos el path, la búsqueda de los ficheros fallaria. Se puede acceder al path porque este se trata como un tipo “keyword”, de esta manera es un tipo diferente que el resto y no se tokeniza.