CS777

Big Data Analytics

Spring 2021

Professor Kia Teymourian

Assignment 1

Kunfei Chen

U15575304

Introduction

This project applies k-nearest neighbors classifier (KNN) in multiple steps to classify text documents via Python and pyspark. The Term Frequency-Inverse Document Frequency (TF-IDF) matrix is a core algorithm to compare similarity distances between different string vectors. There are two data sets for this project: small data set and large data set, which are available at Google Storage and Amazon Web Services. The small dataset is used on local machine to evaluate the project code, whereas the big data set is run at Google Could Platform by cluster. Each data set consists of two files: one is “Wikipedia Pages”, another is “Wikipedia Page Categories”.

Task 1 - Generate the Top 20K dictionary and Create the TF-IDF Array

As shown in Figure 1 and Figure 2, the structure of categories file follows the format with two columns: “doc id” and “category”, the structure of page document is a long string that contains “doc id”, “url”, “title” and “description”.

Figure 1

```
wikiCats.take(2)
```

```
[('434042', '1987_debut_albums'), ('434042', 'Albums_produced_by_Mike_Varney')]
```

Figure 2

```
<doc id="431953" url="https://en.wikipedia.org/wiki?curid=431953"
title="Doug Harvey (ice hockey)">Doug Harvey (ice hockey)Douglas Norman Harvey
    ...
</doc>
```

The big data set’s page document has 1000000 rows of text. After a series of filter, map transformations as well as strings splitting by Regular Expression, the “keyAndListOfWords” Resilient Distributed Datasets (RDD) is generated (See Figure 3).

Figure3

```
keyAndListOfWords.take(1)

Out[11]: [('431949',
  ['black',
   'people',
   'and',
   'mormonismwhile',
   'at',
   'least',
   'two',
   'black',
   'men',
   'held',
   'the',
   'priesthood',
   'in',
   'the',
   'early',
   'church',
   'from',
   'the',
   'mid-1800s'
```

Then all of the words are counted and sorted based on each word's frequency. The top 20000 words are selected to generate a words dict. Each word is represented by an index, the bigger the index is, the less frequent this word occurred (See Figure 4).

Figure 4

```
dictionary = topWordsK.map (lambda x : (topWords[x][0], x))
print("Word Postions in our Feature Matrix. Last 20 words in 20k positions: ", dictionary.top(20, key = lambda x : x[1]))

Word Postions in our Feature Matrix. Last 20 words in 20k positions:  [('underpasses', 19999), ('jammed', 19998), ('underpass', 19997),
('exiting', 19996), ('53rd', 19995), ('haysville', 19994), ('clearwater', 19993), ('retreating', 19992), ('penney', 19991), ('tenacity',
19990), ('142', 19989), ('hermann', 19988), ('mid-september', 19987), ('loos', 19986), ('aisne', 19985), ('prodigious', 19984), ('col',
19983), ('colville', 19982), ('mobilised', 19981), ('balaklava', 19980)]
```

After a series of "map", "join", and "group" transformations, all of the documents are converted into word vectors with "doc id", this RDD is the Term-Frequency (TF) matrix that will be used for TF-IDF calculation (See Figure 5).

Figure 5

```
print(allDocsAsNumpyArrays.take(3))

[('431971', array([0.08153846, 0.02461538, 0.05076923, ..., 0.        , 0.        ,
       0.        ])), ('431999', array([0.06451613, 0.03225806, 0.03870968, ..., 0.        , 0.        ,
       0.        ])), ('432000', array([0.07231245, 0.04524362, 0.02397525, ..., 0.        , 0.        ,
       0.        ]))]
```

Then the IDF array is calculated via a series of count, division, and logarithm operation (See Figure 6).

Figure 6

```
print(idfArray)
```

```
[0.08446916 0.08338161 0.14041215 ... 5.80914299 5.80914299 6.90775528]
```

The TF-IDF matrix can then be calculated by multiple TF matrix and IDF array (See Figure 7), and the featuresRDD is generated by joining categories file with the TF-IDF matrix (See Figure 8).

Figure 7

```
allDocsAsNumpyArraysTFidf = allDocsAsNumpyArrays.map(lambda x: (x[0], np.multiply(x[1], idfArray)))
print(allDocsAsNumpyArraysTFidf.take(2))
```

```
[('431971', array([0.00688749, 0.00205247, 0.00712862, ..., 0.        , 0.        ,
       0.        ])), ('431999', array([0.00544962, 0.00268973, 0.00543531, ..., 0.        , 0.        ,
       0.        ]))]
```

Figure 8

```
featuresRDD = wikiCats.join(allDocsAsNumpyArraysTFidf).map(lambda x: (x[1][0], x[1][1]))
featuresRDD.take(10)
```

```
[('Asteroid_spectral_classes',
  array([0.00724021, 0.00374366, 0.00401178, ..., 0.        , 0.        ,
         0.        ])),
 ('S-type_asteroids',
  array([0.00724021, 0.00374366, 0.00401178, ..., 0.        , 0.        ,
         0.        ])),
 ('All_stub_articles',
  array([0.00815339, 0.0035413 , 0.00813198, ..., 0.        , 0.        ,
         0.        ])),
 ('Military_communications_of_the_United_States',
  array([0.00815339, 0.0035413 , 0.00813198, ..., 0.        , 0.        ,
         0.        ])),
 ('United_States_Department_of_Defense_agencies',
  array([0.00815339, 0.0035413 , 0.00813198, ..., 0.        , 0.        ,
         0.        ])),
 ('United_States_military_stubs',
  array([0.00815339, 0.0035413 , 0.00813198, ..., 0.        , 0.        ,
         0.        ])),
 ('2004_European_Parliament_election',
  array([0.00669928, 0.00258771, 0.00387344, ..., 0.        , 0.        ,
         0.        ])),
```

Task 2 – Implement the Get Prediction function

In this function, the input text will go through the same operations as what has been done above by each text row in the page document. Then the distance between two strings can be

represented by multiplying their TF-IDF value with norm, the bigger the distance value is, the more similar the two strings are. Initially, I calculated the top 10 distances by multiplying fearuresRDD with the input text's TF-IDF array. It performed well in small data set but failed in large data set since featuresRDD has many duplicated TF-IDF arrays with 20000 size and such requires huge RAM. Therefore, I firstly calculated distances by multiplying allDocsASNumpyArraysIFidf with input text's TF-IDF array. Then the distances RDD is joined with wikiCats RDD which also has many duplicated distance values but only with 1 size.

Finally, after the 10 top items generated, the "reduce" function is applied to get the number of times each document category appeared, which is the final result. This operation follows the thoughts of KNN algorithm.

The three prediction results for the big data are shown in Figure 9, Figure 10, Figure 11.

Figure 9

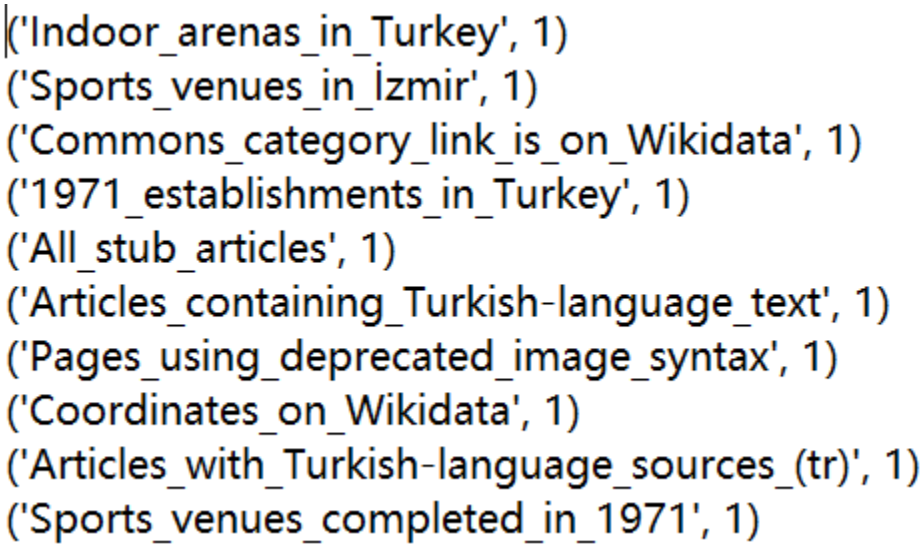

Figure 10

```
('All_disambiguation_pages', 3)
('All_article_disambiguation_pages', 3)
('Disambiguation_pages_with_short_description', 2)
('Disambiguation_pages', 1)
('Airport_disambiguation', 1)
```

Figure 11

```
('Commons_category_link_is_on_Wikidata', 1)
('Use_dmy_dates_from_March_2015', 1)
('Pages_using_infobox_settlement_with_possible_area_code_list', 1)
('Articles_with_short_description', 1)
('All_Wikipedia_articles_written_in_Canadian_English', 1)
('CS1_maint:_location', 1)
('Populated_places_on_the_British_Columbia_Coast', 1)
('Coordinates_on_Wikidata', 1)
('North_Vancouver_(city)', 1)
('Use_Canadian_English_from_March_2015', 1)
```

Task3 - Using Data Frames

- **Task 3.1**

To get the summary statistics about the number of wikipedia categories that are used for wikipedia pages, the two data frames have to be inner joined firstly. Then, the joined dataframe need to be grouped by “Category” column and each group need to be counted its number. As shown in Figure 10, there are total 750859 categories that are used for wikipedia pages. Each category’s mean value of count is 13.07, with standard variance is 690.4, min number is 1, and max number is 376342.

Figure 10

```
Row(summary='count', cat_count='750859')
Row(summary='mean', cat_count='13.068335066903373')
Row(summary='stddev', cat_count='690.4067874030914')
Row(summary='min', cat_count='1')
Row(summary='max', cat_count='376342')
```

- **Task 3.2**

  After sorted by each category's count value, the top 10 categories data frame is generated (See Figure 11)

Figure 11

```
Row(Category='All_stub_articles', cat_count=376342)
Row(Category='Articles_with_short_description', cat_count=235148)
Row(Category='Coordinates_on_Wikidata', cat_count=177113)
Row(Category='Living_people', cat_count=139203)
Row(Category='Wikipedia_articles_with_VIAF_identifiers', cat_count=114446)
Row(Category='Wikipedia_articles_with_WorldCat-VIAF_identifiers', cat_count=114442)
Row(Category="Articles_with_'species'_microformats", cat_count=87462)
Row(Category='Wikipedia_articles_with_LCCN_identifiers', cat_count=87059)
Row(Category='Wikipedia_articles_with_ISNI_identifiers', cat_count=76145)
Row(Category='Webarchive_template_wayback_links', cat_count=70791)
```

- **Task 3.3**

  The joined data frame is then filtered by setting its categories belongs to the top 10 categories. After which, it is grouped by "ID" column and counted. This final top 10 pages that have top 10 largest number of categories is shown in Figure 12.

Figure 12

```
Row(ID='22107', page_count=251)
Row(ID='16775', page_count=243)
Row(ID='19856', page_count=240)
Row(ID='26584312', page_count=233)
Row(ID='40245346', page_count=231)
Row(ID='568942', page_count=230)
Row(ID='2418099', page_count=219)
Row(ID='26103207', page_count=208)
Row(ID='143988', page_count=201)
Row(ID='26208186', page_count=200)
```

Task 4 – Removing Stop Words, do Stemming and redo the task 3

- **Task 4.1**

  In my local machine, I have tried to apply the NLTK package to achieve words splitting. I imported its stop words and use it to filter some words that usually always have high frequency

in text (See Figure 13), which means its IDF value is low. If these stop words can be removed, the comparison for TF-IDF will be more precise, but the prediction result will not be heavily changed because the frequency of these words is generally similar in different input texts. In addition, it also reduces the calculation and reduce time costs.

Figure 13

```
import re
from nltk.corpus import stopwords
print(set(stopwords.words("english")))
```

```
{'ain', 'no', 'and', 'should', "mustn't", 'those', 'just', 's', 'before',
"hadn't", 'most', "wouldn't", 'very', "needn't", 'our', 'haven', 'be', 'c
an', 'mightn', "that'll", "you'd", 'again', "shouldn't", 'she', "she's",
'was', 'above', 'to', 'there', 'ourselves', 'same', 'on', 'themselves',
'until', 'are', 'all', 'hadn', 'isn', 'shouldn', "doesn't", 'weren', 'd',
'during', 'a', 'they', 'himself', 'in', 'that', 'against', 'both', 've',
'after', 'don', 'about', 'below', "mightn't", 'her', 'few', "aren't", 'of
f', 'my', 'is', 'does', 'with', 'whom', 'o', 'too', 'yourself', 'will',
"should've", 'me', "wasn't", 're', 'into', "you'll", 'because', "shan't",
'now', 'myself', 'who', 'him', 'll', 'had', 'other', 'nor', 'ours', 'thes
e', 'than', 'doing', 'each', 'couldn', "won't", 'such', 'more', 'didn',
'here', 'while', 'how', 'of', 'when', 'your', "haven't", 'has', 'have',
'i', "didn't", 'you', 'we', 'do', "hasn't", 'at', 'once', 'wouldn', 'm',
'what', 'mustn', 'he', 'for', 'if', 'by', 'needn', 'up', 'won', "don't",
'shan', 'as', 'aren', 'from', 'their', "it's", 'out', 't', 'down', 'betwe
en', 'only', 'its', 'yours', 'an', 'y', 'this', 'yourselves', "isn't", 'h
asn', "you're", 'them', 'or', 'through', 'under', 'then', 'so', 'some',
"couldn't", 'hers', 'did', 'theirs', 'but', "you've", 'it', 'not', 'ma',
'itself', 'which', 'herself', 'over', 'why', 'were', 'own', 'been', 'hi
s', 'being', 'am', 'any', 'wasn', 'further', 'where', 'doesn', 'the', 'ha
ving', "weren't"}
```

- **Task 4.2 – Do English word stemming**

The word stemming will make the result more precise but will not heavily change the result, because although this will significantly increase some words' TF value, their IDF value will also significantly decrease.