Assignment 2

MET CS 777 - Big Data Analytics kNN classifier
(20 points)

GitHub Classroom Invitation Link

https://classroom.github.com/a/AoSgtpVr

1 Description

In this assignment, you will be working on implementing a kNN (k-Nearest Neighbors) classifier. kNN is a non-parametric machine learning algorithm that classifies new data points based on their similarity to the k nearest labeled data points in a training set.

The objective of this assignment is to use kNN to classify text documents. For instance, if you search for "How many goals did Vancouver score last year?" the algorithm can search through the document corpus and find the top K documents that are similar to your search text. To accomplish this, we will generate a TF-IDF (Term Frequency - Inverse Document Frequency) Matrix from the top 20k words of the text corpus. This matrix will be used to compute the similarity distances between the query text and each document in the corpus.

2 Wikipedia Data Set

You will use the Wikipedia data set. We have downloaded a copy of the whole Wikipedia data set from (https://dumps.wikimedia.org) and stored it in a large file.

Each Wikipedia Page is a document with a unique document ID and a specific URL. For example

- **docID** 418348
- **URL** https://en.wikipedia.org/wiki?curid=418348

Data Description:

Each line is a single document in a pseudo XML format.

```
<doc id="434042" url="https://en.wikipedia.org/wiki?curid=434042"
title="Speed Metal Symphony">Speed Metal SymphonySpeed Metal Symphony
...
all-time top ten shred albums.</doc>
```

3 Obtaining the Dataset

3.1 Small Data Set - Wikipedia Pages

You can find a small data set (Only 1000 Wikipedia pages):

Web access:

- https://storage.googleapis.com/met-cs-777-data/WikipediaPagesOneDocPerLine1000LinesSmall.txt.bz2
- https://storage.googleapis.com/met-cs-777-data/wiki-categorylinks-small.csv.bz2

Google Cloud Storage

- gs://met-cs-777-data/WikipediaPagesOneDocPerLine1000LinesSmall.txt.bz2
- gs://met-cs-777-data/wiki-categorylinks-small.csv.bz2

3.2 Large Data Set (1M pages)

Web access:

- https://storage.googleapis.com/met-cs-777-data/WikipediaPagesOneDocPerLine1m.txt.bz2
- https://storage.googleapis.com/met-cs-777-data/wiki-categorylinks.csv.bz2

Google Cloud Storage

- gs://met-cs-777-data/WikipediaPagesOneDocPerLine1m.txt.bz2
- gs://met-cs-777-data/wiki-categorylinks.csv.bz2

4 Assignment Tasks

In the pyspark code template, you need to complete the missing implementation parts. The missing parts are marked using ”???” (3 question marks).

NOTE: You are free to create an implementation that will be a solution to the assignment tasks if you find it more convenient for you.

Task 1 - Generate the Top 20K dictionary and Create the TF-IDF Array (4 Points)

First we need to generate the top 20,000 words in a local array and sort them based on word frequency, we follow these steps. The top 20K most frequent words in the corpus will constitute our dictionary. Then, we will examine each document and determine if its words appear in the top 20K words.

At the end, we will produce an RDD that includes the document ID as the key and a NumPy array indicating the position of each word in the top 20K dictionary.

The resulting RDD will have the following structure:

```
(docID, [dictionaryPos1, dictionaryPos2, dictionaryPos3, ...])
```

Once we have obtained the top 20K words, our next objective is to create a large array where the columns represent the word list in order, and the rows correspond to the documents. We will begin by creating the term frequency (TF) vector for each document.

Now, let's delve into the specific steps involved in achieving these objectives.

Get the top 20,000 words in a local array and sort them based on the frequency of words. The top 20K most frequent words in the corpus are our dictionary, and we want to go over each document and check if its words appear in the Top 20K words.

In the end, produce an RDD that includes the docID as key and a NumPy array for the position of each word in the top 20K dictionary.

(docID, [dictionaryPos1, dictionaryPos2, dictionaryPos3...])

After having the top 20K words, we want to create a large array that its columns are the word list in order and rows are documents.

The first step is to create the term frequency *TF* (*x*, *w*) vector for each document.

$$TF(x,w) = \frac{x[w]}{\sum_{w'} x[w']}$$

Inverse Document Frequency is:

$$IDF(w) = log\left(\frac{size\ of\ corpuse}{\sum_{"x\ in\ corpus"}\ 1\ if\ (x[w]\geq 1), 0\ otherwise}\right)$$

$$TF-IDF(w) = TF(x,w) \times IDF(w)$$

For more description about TF-IDF see the Wikipedia page: https://en.wikipedia.org/wiki/Tf-idf

- In your code print out print(allDocsAsNumpyArrays.take(3))
- In your code print out print(allDocsAsNumpyArraysTFidf.take(2))

Task 2 - Implement the getPrediction function (8 Points)

Finally, you should implement the function getPrediction (textInput, k). This function will predict the membership of this text string in one of the ten different categories.

You can use the provided template code to implement your kNN.

Note: You may need to optimize the template code to run it on the large dataset

Print out the results for the following queries:

```
print(getPrediction('Sport Basketball Volleyball Soccer', 10))
print(getPrediction('What is the capital city of Australia?', 10))
print(getPrediction('How many goals Vancouver score last year?', 10))
```

Print the results on the output console, or store them on the cloud storage.

Task 3 - Using Dataframes (6 points)

Task 3.1 (3 points)

Use Spark Dataframe to provide summary statistics (max, average, median, std) about the number of Wikipedia categories used for Wikipedia pages. Print the results on the output console, or store them on the cloud storage.

Task 3.2. (3 points)

Use Spark Dataframe to find the top 10 most used Wikipedia categories. Print the results on the output console, or store them on the cloud storage.

Task 4 - Removing Stop Words, do Stemming, and redo task 2 (2 points)

Task 4.1 - Remove Stop Words (1 point)

Describe if removing the English Stop words (most common words like ”a”, “the”, “is”, “are”, “i”, “you”, ...”) would change the final kNN results here.

Would your result be changed heavily after removing the stop words? Why? Provide reasons.

You do not need to implement this task, only discuss your expected outcome results.

Task 4.2 - Do English word stemming (1 point)

We can stem the words [”game”,”gaming”,”gamed”,”games”] to their root word ”game”. Read more about stemming here https://en.wikipedia.org/wiki/Stemming.

Would stemming change your result heavily? Why? Provide reasons and describe them.

You do not need to implement this task, only discuss your expected outcome results.

5 Important Considerations

- **Machines to Use**

One thing to be aware of is that you can choose virtually any configuration for your Cloud Cluster - you can choose different numbers of machines and different configurations of those machines. And each is going to cost you differently! Since this is real money, it makes sense to develop your code and run your jobs locally, on your laptop, using the small data set. Once things are working, you’ll then move to Cloud.

As a proposal for this assignment, you can use the e2-standard-4 machines on the Google Cloud, one for the Master node and two for worker nodes. You will have three machines with a total of 12 vCPU and 48GB RAM. 100 GB of disk space will be enough.

Remember to delete your cluster after the calculation is finished!!!

More information regarding Google Cloud Pricing can be found here https://cloud.google.com/products/calculator. As you can see average server costs around 50 cents per hour. That is not much, but **IT WILL ADD UP QUICKLY IF YOU FORGET TO SHUT OFF YOUR MACHINES**. Be very careful, and stop your machine as soon as you are done working. You can always come back and start your machine or create a new one easily when you begin your work again. Another thing to be aware of is that Google and Amazon charge you when you move data around. To avoid such charges, do everything in the ”Iowa (us-cental1)” region. That’s where data is, and that’s where you should put your data and machines.

- You should document your code very well and as much as possible.
- Your code should be compilable on a Unix-based operating system like Linux or macOS.

• Academic Misconduct Regarding Programming

In a programming class like ours, there is sometimes a very fine line between ”cheating” and acceptable and beneficial interaction between peers. Thus, it is essential that you fully understand what is and what is not allowed in collaboration with your classmates. We want to be 100% precise, so there can be no confusion.

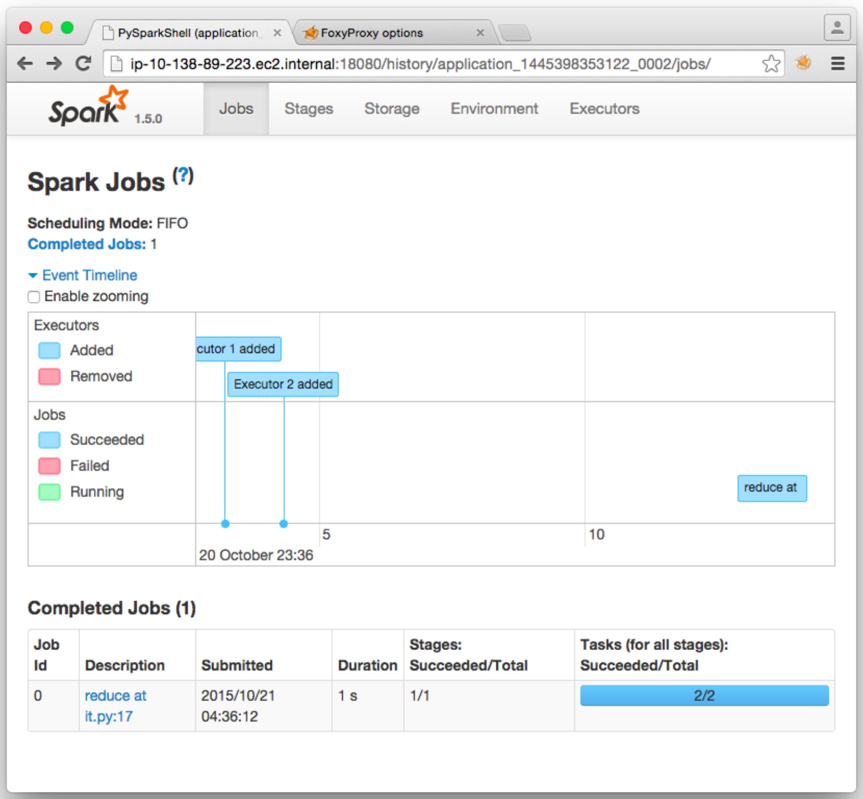
The rule on collaboration and communication with your classmates is very simple: you cannot transmit or receive code from or to anyone in the class in any way—visually (by showing someone your code), electronically (by emailing, posting, or otherwise sending someone your code), verbally (by reading code to someone) or in any other way we have not yet imagined. Any other collaboration is acceptable.

The rule on collaboration and communication with people who are not your classmates (or your TAs or instructor) is also very simple: it is not allowed in any way, period. This disallows (for example) posting any questions of any nature to programming forums such as **StackOverflow**. As far as going to the web and using Google, we will apply the **”two-line rule”**. Go to any web page you like and do any search that you like. But you cannot take more than two lines of code from an external resource and actually include it in your assignment in any form. Note that changing variable names or otherwise transforming or obfuscating code you found on the web does not render the ”two-line rule” inapplicable. It is still a violation to obtain more than two lines of code from an external resource and turn it in, whatever you do to those two lines after you first obtain them.

Furthermore, you should cite your sources. Add a comment to your code that includes the URL(s) that you consulted when constructing your solution. This turns out to be very helpful when you’re looking at something you wrote a while ago and you need to remind yourself what you were thinking.

• Turnin

1. Fill in the results in the provided template.
2. For each task, and for each Spark job you ran, include a screenshot of the Spark History. **To demonstrate that you did execute your code on the cloud, it is important to include URLs in the screenshots. Otherwise, there is no way for us to verify if the code was executed in your cloud account.**



ip-10-138-89-223.ec2.internal:18080/history/application_1445398353122_0002/jobs/

Spark Jobs (?)

Scheduling Mode: FIFO
Completed Jobs: 1

Completed Jobs (1)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	reduce at it.py:17	2015/10/21 04:36:12	1 s	1/1	2/2

Figure 1: Screenshot of Spark History

3. Please zip up all of your code and your document (use .zip only, please!), or attach each piece of code and your document to your submission individually.
4. Please have the latest version of your code on GitHub. Zip the GitHub files and submit your latest version of assignment work to Blackboard. We will consider the latest version on the Blackboard, but it should exactly match your code on GitHub.
5. **Remember to delete your cluster after the calculation is finished!!!**