CS777 – Week 2 Homework Submission Template

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

Get the top 20,000 words in a local array and sort them based on the frequency of 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...])

* In your code print out print(allDocsAsNumpyArrays.take(3)).

|  |
| --- |
| **[('1266808', array([0.08139003, 0.02286237, 0.02331962, ..., 0. , 0. , 0. ])), ('45023105', array([0.06428571, 0.03809524, 0.05 , ..., 0. , 0. , 0. ])), ('146605', array([0.1056231 , 0.05357143, 0.03381459, ..., 0. , 0. , 0. ]))]** |

* In your code print out print(allDocsAsNumpyArraysTFidf.take(2)):

|  |
| --- |
| **[('13971', array([0.007576 , 0.00863177, 0.00416492, ..., 0. , 0. , 0. ])), ('23158463', array([0.0072345 , 0.00555417, 0.00455503, ..., 0. , 0. , 0. ]))]** |

* Include the relevant code excerpt that you used for creating an RDD that satisfies the above conditions

|  |
| --- |
| **allWordsWithDocID = keyAndListOfWords.flatMap(lambda x: ((j, x[0]) for j in x[1]))**  **allDictionaryWords = dictionary.join(allWordsWithDocID)**  **justDocAndPos = allDictionaryWords.map(lambda x: (x[1][1], x[1][0]))**  **allDictionaryWordsInEachDoc = justDocAndPos.groupByKey()**  **allDocsAsNumpyArrays = allDictionaryWordsInEachDoc.map(lambda x: (x[0],**  **buildArray(x[1])))**  **print(allDocsAsNumpyArrays.take(3))**  **######################################################################**  **zeroOrOne = allDocsAsNumpyArrays.map(lambda x: (x[0], zero\_or\_one(x[1])))**  **dfArray = zeroOrOne.reduce(lambda x1, x2: ("", np.add(x1[1], x2[1])))[1]**  **multiplier = np.full(numTopWords, numberOfDocs)**  **idfArray = np.log(np.divide(multiplier, dfArray))**  **allDocsAsNumpyArraysTFidf = allDocsAsNumpyArrays.map(lambda x: (x[0],**  **np.multiply(x[1], idfArray)))**  **print(allDocsAsNumpyArraysTFidf.take(2))** |

**Task 2 – Implement the getPrediction function (8 Points)**

Print out the results for the following queries:

* print(getPrediction('Sport Basketball Volleyball Soccer', 10))

|  |
| --- |
| **('All\_stub\_articles', 4)**  **('All\_disambiguation\_pages', 3)**  **('Disambiguation\_pages', 3)**  **('Disambiguation\_pages\_with\_short\_description', 3)**  **('All\_article\_disambiguation\_pages', 3)**  **('Articles\_containing\_Turkish-language\_text', 2)**  **('Articles\_with\_Turkish-language\_sources\_(tr)', 2)**  **('Living\_people', 2)**  **('1958\_establishments\_in\_Turkey', 1)**  **('Sports\_venues\_in\_İzmir', 1)** |

* print(getPrediction('What is the capital city of Australia?', 10))

|  |
| --- |
| **('All\_disambiguation\_pages', 4)**  **('Disambiguation\_pages\_with\_short\_description', 4)**  **('All\_article\_disambiguation\_pages', 4)**  **('Disambiguation\_pages', 3)**  **('All\_stub\_articles', 2)**  **("Articles\_with\_'species'\_microformats", 1)**  **('Commons\_category\_link\_from\_Wikidata', 1)**  **('Moths\_described\_in\_1852', 1)**  **('Regional\_capitals\_in\_Tanzania', 1)**  **('Lists\_of\_countries', 1)** |

* print(getPrediction('How many goals Vancouver score last year?', 10))

|  |
| --- |
| **('Webarchive\_template\_wayback\_links', 3)**  **('Coordinates\_on\_Wikidata', 3)**  **('Articles\_with\_short\_description', 2)**  **('All\_disambiguation\_pages', 2)**  **('All\_articles\_with\_unsourced\_statements', 2)**  **('Disambiguation\_pages', 2)**  **('Disambiguation\_pages\_with\_short\_description', 2)**  **('All\_article\_disambiguation\_pages', 2)**  **('North\_Vancouver\_(city)', 1)**  **('Populated\_places\_on\_the\_British\_Columbia\_Coast', 1)** |

* Include the relevant code excerpt that shows how you implemented the getPrediction() function

|  |
| --- |
| **def getPrediction(textInput, k):**  **myDoc = sc.parallelize(('', textInput))**  **wordsInThatDoc = myDoc.flatMap(lambda x: ((j, 1) for j in regex.sub(' ',**  **x).lower().split()))**  **allDictionaryWordsInThatDoc = dictionary.join(wordsInThatDoc).map(lambda x:**  **(x[1][1], x[1][0])).groupByKey()**  **myArray = buildArray(allDictionaryWordsInThatDoc.top(1)[0][1])**  **myArray = np.multiply(myArray, idfArray)**  **distances = allDocsAsNumpyArraysTFidf.map(lambda x: (x[0], cousinSim(x[1],**  **myArray)))**  **topK = distances.top(k, lambda x: x[1])**  **docIDRepresented = sc.parallelize(topK).map(lambda x: (x[0], 1))**  **docIDwithWikiCat = wikiCats.join(docIDRepresented).map(lambda x: (x[1][0],**  **x[1][1]))**  **numTimes = docIDwithWikiCat.reduceByKey(lambda x, y: x + y)**  **return numTimes.top(k, lambda x: x[1])** |

**Task 3 – Using Dataframes (6 points)**

**Task 3.1**

Use Spark Dataframe to provide summary statistics (max, average, median, std) about the

number of Wikipedia categories that are used for Wikipedia pages. Print the results on the output

console, or store them on the cloud storage.

|  |
| --- |
| **[Row(max(count)=587)]**  **[Row(avg(count)=5.566655905819486)]**  **[Row(stddev\_pop(count)=5.567163357116429)]**  **[3.0]** |

**Task 3.2**

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.

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

|  |
| --- |
| **Yes,remove these stop words will decrease the low level and highly repeated information**  **of our text then our KNN classifier could focus on comparing the important information**  **of texts, then optimize the KNN result.** |

**Task 4.2 – Do English word stemming (1 point)**

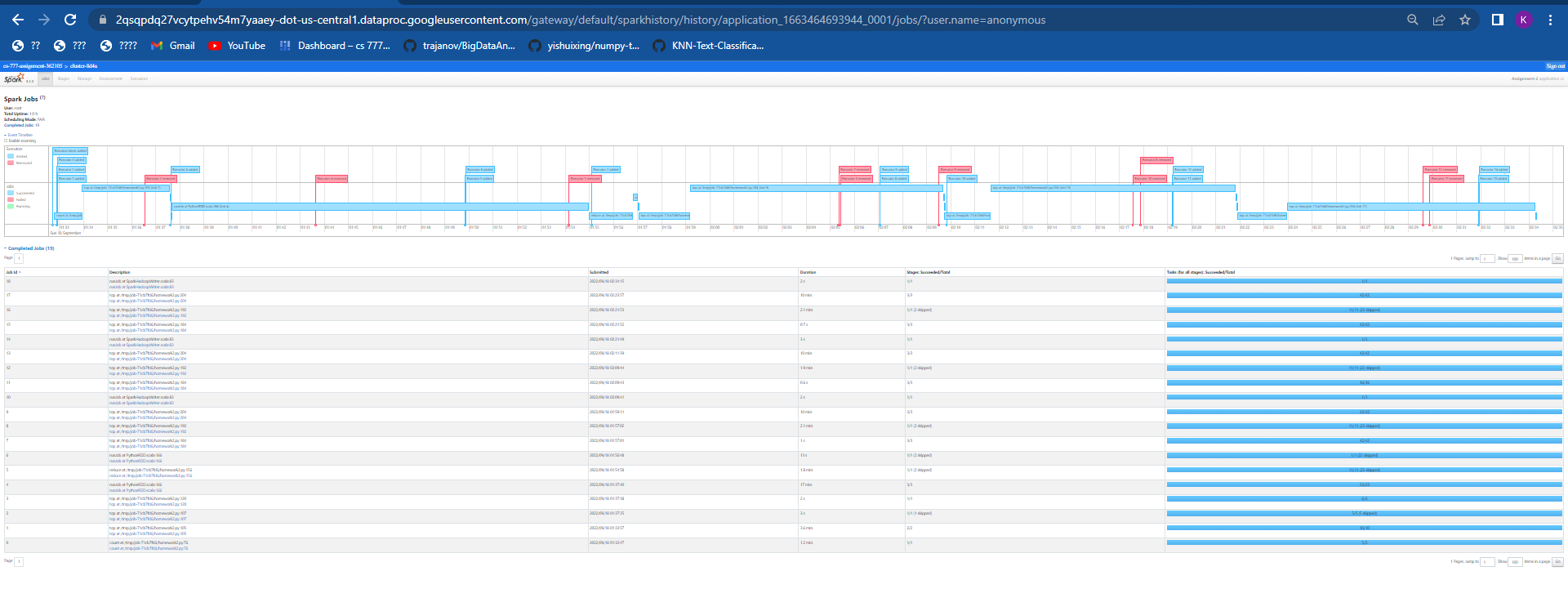
We can stem the words [”game”,”gaming”,”gamed”,”games”] to their root word ”game”. Read more about stemming here <https://en.wikipedia.org/wiki/Stemming>

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

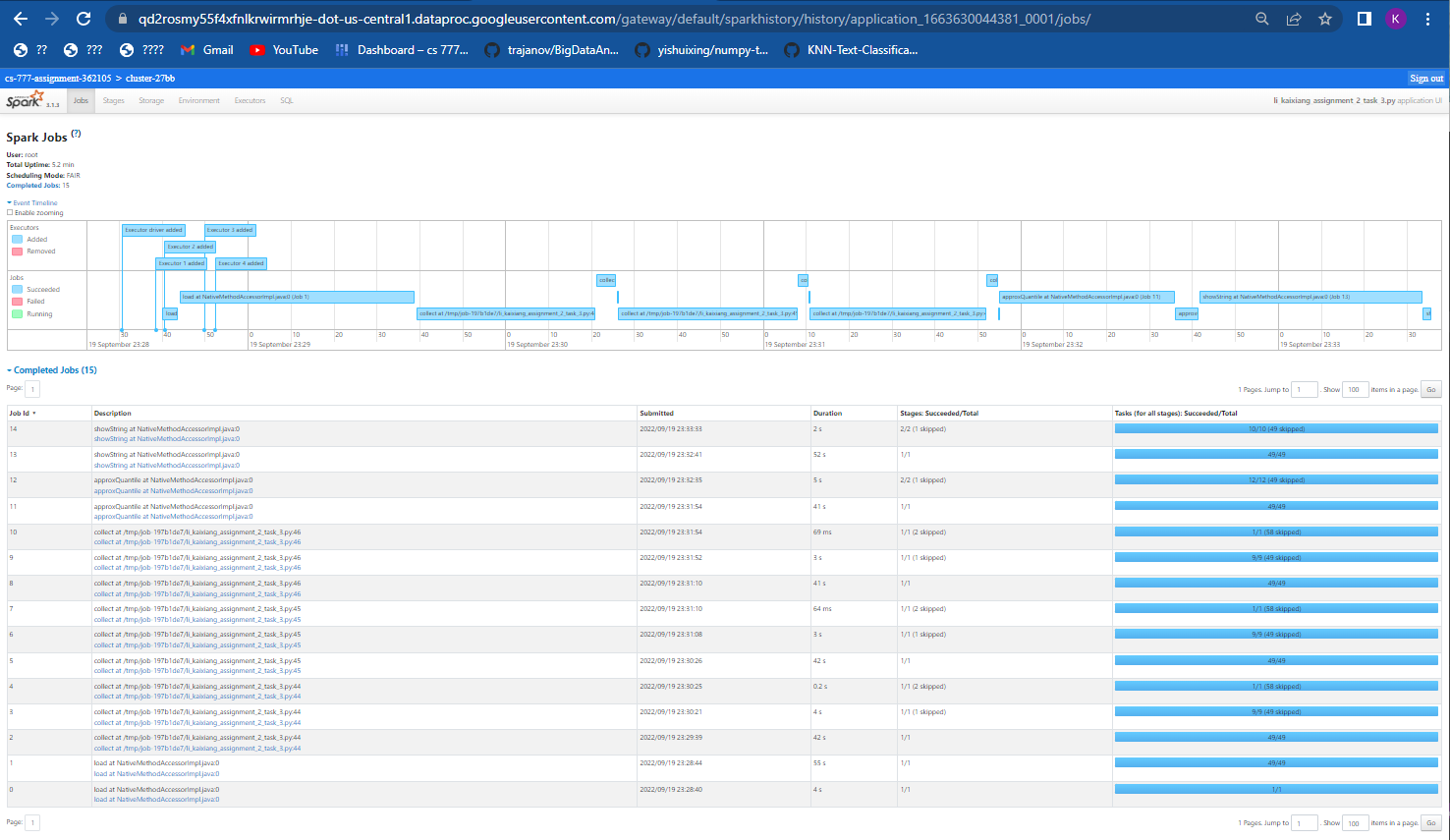
|  |
| --- |
| **I believe stem the words to their root will help us optimize the KNN result too.**  **After , all of the words back to their root, our KNN classifier could better**  **implement statistical calculation like the word frequency to compare difference between**  **texts.** |

**Spark History Output:**

**Task1 and 2**

****

**Task3:**

****