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)).

|  |
| --- |
| *Your output should go here.* |

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

|  |
| --- |
| *Your output should go here.* |

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

|  |
| --- |
| *#The code ex* individual documents the i^th word in the dictionary appeared in      dfArray = zeroOrOne.reduce(lambda x1, x2: ("", np.add(x1[1], x2[1])))[1]      # Create an array of 20,000 entries, each entry with the value numberOfDocs (number of docs)      multiplier = np.full(numTopWords, numberOfDocs)      # Get the version of dfArray where the i^th entry is the inverse-document frequency for the      # i^th word in the corpus  Anjana check      idfArray = np.log(np.divide(multiplier, dfArray))        # Finally, convert all of the tf vectors in allDocsAsNumpyArrays to tf \* idf vectors      allDocsAsNumpyArraysTFidf = allDocsAsNumpyArrays.map(lambda x: (x[0], np.multiply(x[1], idfArray)))  *cerpt should go here.* |

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

Print out the results for the following queries:

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

|  |
| --- |
| *Prediction for Sport Basketball Volleyball Soccer: [('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), ('Sports\_venues\_in\_İzmir', 1), ('Medalists\_at\_the\_1984\_Summer\_Olympics', 1)]* |

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

|  |
| --- |
| *YourPrediction for What is the capital city of Australia?: [('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))

|  |
| --- |
| *How many goals Vancouver score last year?: [('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), ('Articles\_to\_be\_expanded\_from\_February\_2015', 1), ('North\_Vancouver\_(city)', 1)]* |

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

|  |
| --- |
| *The code excerpt s*def getPrediction (textInput, k):          # Create an RDD out of the textIput          myDoc = sc.parallelize (('', textInput))          # Flat map the text to (word, 1) pair for each word in the doc          wordsInThatDoc = myDoc.flatMap (lambda x : ((j, 1) for j in regex.sub(' ', x).lower().split()))          # This will give us a set of (word, (dictionaryPos, 1)) pairs          allDictionaryWordsInThatDoc = dictionary.join (wordsInThatDoc).map (lambda x: (x[1][1], x[1][0])).groupByKey ()          # Get tf array for the input string          myArray = buildArray (allDictionaryWordsInThatDoc.top (1)[0][1])          # Get the tf \* idf array for the input string          myArray = np.multiply(myArray,idfArray)          # Get the distance from the input text string to all database documents, using cosine similarity          distances = allDocsAsNumpyArraysTFidf.map (lambda x : (x[0], cousinSim (x[1],myArray)))          # get the top k distances          topK = distances.top (k, lambda x : x[1])            # and transform the top k distances into a set of (docID, 1) pairs          docIDRepresented = sc.parallelize(topK).map (lambda x : (x[0], 1))          docIDwithWikiCat = wikiCats.join(docIDRepresented).map(lambda x: (x[1][0], x[1][1]))          # now, for each docID, get the count of the number of times this document ID appeared in the top k          numTimes = docIDwithWikiCat.reduceByKey(lambda x,y: x+y)            # Return the top 1 of them.          # Ask yourself: Why we are using twice top() operation here?          return numTimes.top(k, lambda x: x[1])  *hould go here.* |

**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.

|  |
| --- |
| *Your output should go here.* |

**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.

|  |
| --- |
| *Your output should go here.* |

**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.

|  |
| --- |
| *Stop words won't provide with any insights, and because they are frequently used throughout texts, their frequency is larger than that of other helpful terms . As a result, stop words will be given more weight than other words. This will have an impact on the model's performance on TF-IDF-based methods ( Term Frequency- Inverse Document Frequency).  From the above result stop words "the, is, of are given more weight then other words and yes results would change by removing stop words and get better 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>

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

|  |
| --- |
| *Stemming, in general, combines various tokens into one (for example* game”,”gaming”,”gamed”,”games” *==> game). There is an overall improvement in recall (i.e., discovering more hits from the general pool) and a decrease in precision in information retrieval and classification (i.e. more hits are false positives, as the stemmed signal is diluted). KNN will perform better without stemming if definition of success is either precision or something that gives precision a high weight.* |

**Spark History Output:**

**Task 1: and Task2**

|  |
| --- |
| *.* |

**Task 2:**

|  |
| --- |
| *Your screen capture should go here.* |

**Task 3:**

|  |
| --- |
| *Your screen capture should go here.* |