Data Science Tools and Models: Spark kNN

1 Description

In this assignment, you will be implementing a kNN classifier to classify text documents. "Classification" is the task of labeling documents based upon their contents. The implementation will be in Python, on top of Spark, **USING RDDs**. You will be asked to perform three subtasks, covering data preparation and classification.

2 Data

You will be dealing with MEDLINE/PubMed data:
(`https://www.nlm.nih.gov/bsd/pmresources.html`).

MEDLINE is a bibliographic database of journal citations, abstracts, and metadata. The data set is composed of categories, identifier, and abstract triples. This data set has 26,754 such posts from 11 different categories, according to catagories assigned to the articles. The 11 categories are listed in the file `categories.txt`. The category name can be extracted from the name of the document. For example, the document with identifier `doc id=Wounds/24458063` is from the `Wounds` category. The document with the identifier `doc id=MedicalEducation/20662575` is from the `MedicalEducation` category. The data file has one line per document for a total of ˜38 MB of text. It can be accessed via Spark at: `s3://[DataLocation]/pubmed.txt`

A small subset of the data, for testing purposes, is in `pubmedSmall.txt`.

3 The Tasks

There are three separate tasks that you need to complete to finish the assignment.

3.1 Task 1

First, you need to write Spark code that builds a dictionary that includes the 20,000 most frequent words in the training corpus - this was part of the Spark Lab. When you do this, please start with the code provided with THIS assignment in `kNNstart.py` so that we know that everyone has the same dictionary. Then, you need to use this dictionary to create an RDD where each document is represented as one entry in the RDD. Specifically, the key of the document is the document identifier (like `BrainInjuries/17626737`) and the value is a NumPy array with 20,000 entries, where the ith entry in the array is the number of times that the ith word in the dictionary appears in the document.

Once you do this, print out the arrays that you have created for documents
`Wounds/23778438`,
`ParasiticDisease/2617030`, and
`RxInteractions/1966748`

Since each array is going to be huge, with a lot of zeros, the thing that you want to print out is just the non-zero entries in the array (that is, for an array `a`, print out `a[a.nonzero()]`.

3.2 Task 2

It is often difficult to classify documents accurately using raw count vectors. Thus, the next task is to write some more Spark code that converts each of those 26,754 count vectors to TF-IDF vectors "term

frequency-inverse document frequency vectors"). The ith entry in a TF-IDF vector for document d is computed as:

$$TF(i,d) \times IDF(i)$$

Where $TF(i,d)$ is:

$$\frac{\text{Number of occurrences of word } i \text{ in } d}{\text{Total number of words in } d}$$

Note that the "Total number of words" is not the number of distinct words. The "total number of words" in "Today is a great day today" is six. And the $IDF(i)$ is:

$$\log \frac{\text{Size of corpus (number of docs)}}{\text{Number of documents having word } i}$$

Again, once you do this, print out the arrays that you have created for documents:

`PalliativeCare/16552238`,
`SquamousCellCancer/23991972` and
`HeartFailure/25940075`

Again, print out just the non-zero entries.

3.3 Task 3

Next, your task is to build a pySpark kNN classifier, embodied by the Python function `predictLabel`. This function will take as input a text string and a number k, and then output the name of one of the 11 categories. This name is the new group that the classifier thinks that the text string is "closest" to. It is computed using the classical kNN algorithm. This algorithm first converts the input string into a TF-IDF vector (using the dictionary and count information computed over the original corpus). It then finds the k documents in the corpus that are "closest" to the query vector (where distance is computed using the L_2 norm), and returns the category label that is most frequent in those top k. Ties go to the label with the closest corpus document. Once you have written your function, run it on the following (each is an excerpt from a pubmed abstract, chosen to match one of the 11 categories). These function call are provided in the file `kNNQueries.py`.

4 Turn in

Submit two documents: one with your results (.txt) and one with your code (.py). Create a single document that has results for all three tasks. Turn in this document as well as all of your code. Please zip up all of your code and your text document (use .gz or .zip only, please!), or else attach each piece of code as well as your text results document to your submission individually. No PDFs of code, please!

5 Grading

Each task is worth 33% of the overall grade.