David Tang
CMPE 255
Program 1 Report
Rank: 33
F1-Score: 0.4070

There were several things that I did to preprocess the training data.

1. Remove punctuations because they are rarely useful for what we are doing
2. Lower-case all words so that they are normalized
3. Change words like **aren't** -> **are not**. This is done via the *contractions* library.
4. Transform the document into an array of array of words.
5. Sort the array by the first word in each document. In our case this is our label (1 - 5). This is important because it allows us to do more efficient slicing when we construct our CSR matrix later. We can slice out an entire group of labels to use.
6. Iterate through each document and remove the labels. As we are removing the labels,construct an array of indexes to keep track of when each label starts and ends. We need this to slice the CSR matrix later.
7. Remove stop words. These are common words that a generic and don't give much information about the context.
8. Stem each word. For instance, change **digusting** -> **disgust**.
9. Change digits to words. For instance, **47 -> forty seven**.

After preprocessing the data, I had to:

1. Use the build_matrix function provided in one of the activities. I had to modify it so that it can handle normalizing test data with no labels. I used the same function to build a CSR matrix for the test data, but I had to provide the **word-to-tid** mapping that was generated from the training data.
2. Normalize the data by the inverse document frequency.
3. Normalize the data by the L2-norm.

Then the major headaches begin...

I initially tried to calculate the cosine similarity of each test row vs every other training row. While I could wait hours for it to complete, it would take too long for each iteration for me to see results. I decided to separate the rows with the same labels into equal partitions. I create a new document vector that is the average of each unit within the same partition. This new document vector is a centroid that will represent that partition of labeled data.

When I was first trying to train my model, I set k=1 and made all the documents in the same label be represented by a single centroid. I discovered that my algorithm to generate the centroid was too slow since I did a lot of single row slicing on the CSR matrix. When I was word-normalizing the document, I made a dictionary of **label -> list of rows** that would be used for slicing. This turned out to be extremely inefficient. I had to rethink my process and eventually came up with **#5** in the preprocessing step. Despite this optimization, performing a dot product one at a time was still slow enough where generating to centroids were too slow. I eventually found the function call **<2d_matrix>.mean(axis=0)** that immediately computed the centroid for me efficiently.

I have 3 metaparameters: **p** (partition), **k** (neighbors), and **e** (epsilon). **p** is used to determine how many centroids we should construct to represent the entire same-label documents. **k** represents how many nearest centroids we should use to determine what class the document belongs to. When we compute the cosine similarity between each test document against our centroids, we will discard those values if they are less than **e**. However, in the case where the set of cosine similarities all fall below **e**, we will keep them all instead. This is to prevent the rare case where we filter out everything and won't be able to classify the document at all.

Once I got a basic implementation working, I wrote a **self_train** method that splits the training data so that 80% is used for training and the other 20% for testing. This way I can validate how my model is working. Unfortunately, most tuning I tried on my 3 parameters didn't seem to produce good results. They all fell within the 40-50% accuracy range.

I spoke with some classmates on how they approached the problem. It seems like the ones I talked to went with the most simple and straightforward approach of doing a dot product of each test data with every row of training data. I learned that there is an efficient way to compute the dot product by multiplying a 1D array (test row) against the transpose of the 2D array (training data). This was the information that I was missing so it led me to believe I needed to use centroids so I can test data against less vectors.

It was already to late when I learned this information since I can't change my code at the last minute. However, I did try to create many centroids per partition to simulate having a centroid be approximately 1 training data. This also failed miserably because it took around 10 minutes just to generate the centroids. Computing against that many centroids was even slower because there is generic logic to handle **len(centroids) != len(data)**.

My code uses several 3rd party libraries that do not come by default with Jupyter Notebook (Python 3.7.1). Assuming you have pip installed, you will need to run "*pip install contractions inflect*" in order to properly run the code.