STA 141: Big Data and High Performance Computing
Dayanara Lebron-Aldea
Prof. Chohsieh

HW1: Quora Pair Questions

The goal of this homework was to develop a code that determines whether two questions are the same depending on its content. We were given two datasets: training and testing, composed of 323,264 and 81,035 pair questions each with a label column. The label column have a binary response [0,1] as factors for " not equal" and equal" questions, respectively.

Data Pre-Process:

For Part 1 we were asked to develop a code that gets rid of punctuation characters and consequently, turns to lower case all letters in the string. For this I coded a method named *Pre-process()* that takes as input a string vector (i.e vector of strings for q1) and outputs the processed string vector. Although in the instructions it was asked that the method inputs a single string, I found out that the script takes longer to run when processing a single string at a time than when inputting the entire vector into the method. Pre_process(string_vector) is defined as:

```
def Pre_process(string_vector):
       #Pre_process is supposed to input each string not a list of strings
       characters=[",","?","'","!",".","(",")","\"",":"]
       new_strv=[]
       for s in string_vector:
              s=str(s)
              for c in characters:
                     if c in str(s):
                            s=s.replace(c," ")
              s=s.strip("-")
              new_strv.append(s.lower())

       return new_strv
```

The Problem1_v2.py script creates a Process_filename*.csv file with the processed content which will be the input for all the other scripts. It can be ran as follows:

```
python Problem1_v2.py training.csv
```

Overlapping Score:

Part 2 had as purpose to compute the overlapping score between pair of questions. The overlapping score is defined as:

$$os(q1,q2) = \sum \mathrm{I}(w_{1i} \in q_2) + \sum \mathrm{I}(w_{2i} \in q_1) \,/(m+n).$$

for this part a method named Compute_PairScore(q1,q2) was coded, where q1 and q2 are the pair of questions and Compute_PairScore() is defined as:

```
def Compute_PairScore(q1,q2):
    score_1=len(set(q1).intersection(q2))
    score_2=len(set(q2).intersection(q1))
    m=len(q1)
```

```
    n=len(q2)

    #Only compute the overlapping scores for
    if (m+n)>0:
        overlap_score=round((score_1+score_2)/(float(m)+float(n)),4)
    else:
    #Hw states that if all words are stop words set overlapping score to be 0
        overlap_score=0
    return overlap_score
```

This part was created as a method so it could later be imported as part of a Module for use in Part 3 and Part 4. To calculate the number of words similar per questions the intersection() command was used together with the set() method which gives the unique list of words in a given question. This was done to avoid counting a word twice if used in a loop. The ComputeScore.py takes as input the Processed_filename*.py and outputs the overlapping scores per pair of questions on the command line. The script can be ran in command line as:

python ComputeScore.py Processed_filename*.csv

Training Accuracy:

After having the overlapping scores we proceeded to make predictions for the pair of questions by making use of the label column and a specified threshold (thr) . For the interval $(-, thr)$ the prediction was given a negative label in this case 0 and for the interval $[thr, +)$ the prediction was given a positive label. This was then compared to the actual label given by the dataset in order to calculate the accuracy. Multiple thresholds were tested and accuracy for each one was recorded. To compute the accuracy we created a method named Compute_Accuracy() defined as:

```
def Compute_Accuracy(data, thr):
    label_pairs=[]
    labels=data[5]
    n=len(labels)
    count=0
    # Run Compute Score and susbtract the threshold
    for i in range(0,len(data[3])):
        os=cs.Compute_PairScore(data[3][i],data[4][i])
        predict=np.sign(os-float(thr))
        # Here we will say that (-) is 0 label and (+) is 1
        if predict < 0:
            label_pairs.append(0)
        else:
            label_pairs.append(1)

        if str(label_pairs[i]) is str(labels[i]):
            count+=1

    # Compute testing accuracy
```

```
    acc= round(count/float(n),4)
    return acc
```

The ComputeAccuracy.py can be write in the command line as stated in the hw pdf.

Removing Stop Words and Computing Accuracy:

Another approach for this problem was to remove the stop-words from the questions to only leave relevant words, in efforts of improving the model. By doing this we have to consider the case in which all the words in a question are stop words, and if that is the case, we just state that the overlapping score is 0. Stop words were defined as words occurring more than 10,000 times in the training.csv. The AccuracyRemoveStop.py script removes the stop words, calculates the predictions, assigns the labels and displays the accuracy for a given threshold. The computation of the accuracy follows the same method as the Compute_Accuracy() method above and to calculate the stop words the below lines of codes were used:

```
#First make a big list of all words in questions 1 and 2
list1=[val for sublist in [s.split() for s in data_stop[3]] for val in sublist]
list2=[val for sublist in [s.split() for s in data_stop[4]] for val in sublist]
full_list=list1+list2
#count occurrances and choose ones with more than 10,000 ocurrances as stop words
counts = pd.Series(full_list).value_counts()
list_counts=counts.index
stop_words=list_counts[counts>10000]
```

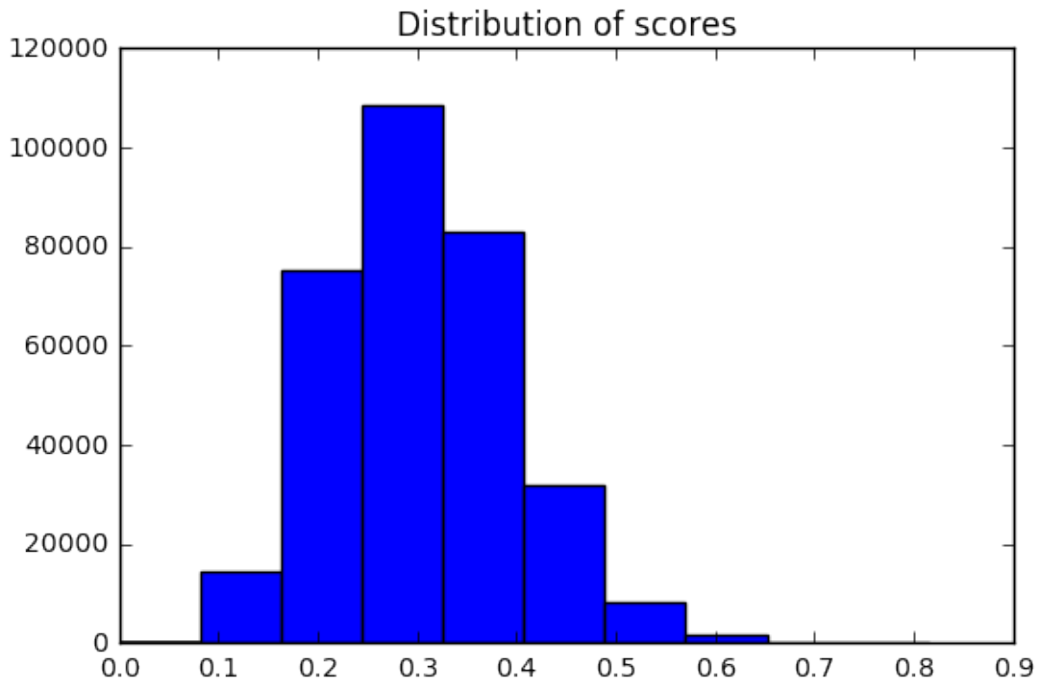
Results:

** All of the scripts were ran with python 2.7.12

Results for Part 2:

Overlapping scores for the first 10 lines were [0.3252, 0.2302, 0.2261, 0.2783, 0.2159, 0.3662, 0.3889, 0.3303, 0.2081, 0.3704] , the maximum, minimum and median over all the thresholds were: 0, 0.8148, 0.2989, respectively.

Discuss the results:



The scores are normally distributed as you can see in the graph above, most of the questions had a score between 0.15 and 0.5. No pair of questions yielded a perfect match, and scores did not transcend the 0.81 score.

Results for Part 3 and 4:

Threshold	**Accuracy for training**	**Accuracy without Stop Words for Training**
0.1	0.3703	0.4878
0.2	0.4625	0.539
0.3	0.6117	0.6131
0.4	0.6400	0.6504
0.45	0.6340	0.6670
0.5	0.6317	0.6682
0.55	0.6313	0.6677
0.6	0.6313	0.6652
0.65	0.6312	0.6598

0.7	0.6311	0.6550
0.8	0.6309	0.6539
1.0	0.6309	0.6585

The maximum accuracy of **0.6400** was achieved with a 0.4 threshold, running the training.csv with the ComputeAccuracy.py script, with that same threshold we achieved an accuracy of **0.6397** on the validation dataset . The maximum accuracy achieved with the AccuracyRemoveStops,py script was **0.6682** with a threshold of 0.5, and with that same threshold we achieved an accuracy of **0.6683** in the validation.csv. We see that in both columns the accuracies were low at the beginning and jumped to 0.61 when the threshold was 0.3, after that in both cases (with and without stop words) the accuracies started to slowly fall below the maximum when the threshold increased. We can see that overall there can be seen constant 2.5% increase from threshold 0.45 and above, and that the starting difference in accuracies between both methods were 11.75% in the 0.1 threshold. When using the ComputeAccuracy.py a lower threshold make more questions to be labelled as 1 than 0's, when in reality in our training dataset we have more 0's than 1's this is why the first thresholds achieved a low accuracy. When using the without stop-words method, we are taking out words that could change the sense of the sentence for example: "Will it rain today?" vs "Will it not rain today?". Removing stop words will probably remove: "will", "it" and "not" from the questions leaving both questions to seem equal this may be one of the reasons why accuracy is higher when using this method.

For another case we could also try to lemmatize the words in the questions, since the same question can be written in different tenses and still have the same meaning, and lemmatization could help with that problem.