Katherine Olson

STA 141C Homework #1 – Quora Question Pairs

This assignment works on the Quora question pairs problem. Given two Quora questions, determine if they are asking the same thing or not.

Problem 1

Problem: Write a function in python to pre-process a sentence (string).

Coding Process:

The function for this problem can be seen below. It takes and string and removes all of the punctuation that could get in the way when matching words in the future.

```
def preprocess(str_in):
    “””
    Input: A string
    Output: That string with all lowercase letters, unwanted punctuation replaced with whitespace, and all “-“ removed.
    “””
    # Catch NaNs:
    try: str_out = str_in.lower() # Convert to lower case
    except: return str_in

    unwanted = [“?”, “!”, “.”, “(“, “)”, “‘”, “’”, “:”, “,”]

    # Remove unwanted puncutaion:
    for char in unwanted:
        str_out = str_out.replace(char, “ “) # Replace with space

    str_out = str_out.replace(“-“, “”) # Remove

    return str_out
```

Full code can be downloaded here: https://katherineolson.github.io/141C_HW1_problem1.py

Problem 2

Problem: Write a program, given "training.csv" as input, output the overlapping score for each row. Assume the question pair q_1 and q_2 are the preprocessed strings after Problem 1, and $q_1 = [w_{11}\ w_{12} \cdots w_{1n}]$, $q_2 = [w_{21}\ w_{22} \cdots w_{2m}]$ (the strings are split into words using white space), then we define the overlapping scores as the number of overlapping words divided by m + n (sum of number of words). More specifically,

$$\text{overlapping score } (q_1, q_2) := \left(\sum_{i=1}^{n} I(w_{1i} \in q_2) + \sum_{i=1}^{m} I(w_{2i} \in q_1)\right)/(m+n)$$

where $I(w_{1i} \in q_2) = 1$ if $w_{1i} \in q_2$, otherwise $I(w_{1i} \in q_2) = 0$.

Coding Process:

My function for computing scores can be seen below. It takes in two questions and gives back a score that represents how many words those two questions have in common.

```
def ComputeOneScore(quest1, quest2):
    from problem1 import preprocess
    """
    Input: Two questions - each as a string.
    Output: A score for how much those questions overlap.
    """
    score1, score2 = 0.0, 0.0 # Set to zero to start
    q1 = preprocess(quest1).split()
    q2 = preprocess(quest2).split()

    # Compute score:
    for word1 in q1:
        if word1 in q2: # If a word from question 1 is in question 2, increase the score
            score1 = score1 + 1
    for word2 in q2:
        if word2 in q1: # If a word from question 2 is in question 1, increase the score
            score2 = score2 + 1
    score = (score1 + score2) / (len(q1) + len(q2)) # Total score
    return  score
```

For my main function I read in the data and send each question pair from each row to the ComputeOneScore function. I use a try and except to catch any NaNs. If I find an NaN, then I set the score to zero. Then each score is printed.

Results:

The first ten scores
0.961538461538
0.52380952381
0.0
0.4
0.676470588235
0.5
0.75
0.444444444444
0.107142857143
0.588235294118

Minimum	Median	Maximum
0.0	0.5	1.0

Looking at the results, the minimum and maximum score are not surprising. Within the dataset we have at least one question pair with none of the same words and we also have at least one question pair with all of the same words. The median of 0.5 tells us that half of the question pairs have a score below 0.5 and the other half have a score above 0.5. The first ten scores give a good range and fit in with the summary statistics.

Full code can be downloaded here: https://katherineolson.github.io/141C_HW1_problem2.py

Problem 3

Problem: After having the overlapping scores, we can predict whether two questions are the same by thresholding. More specifically, we can predict the label for a question pairs (q_1, q_2) by

$$\text{sign}(\text{overlapping score}(q_1, q_2) - \text{thr}),$$

where thr is a positive real number for thresholding. The testing accuracy is defined as

(number of correct predicted pairs)/(total number of pairs)

Write a program to compute the testing accuracy for a given dataset and threshold.

Coding Process:

My function to compute the accuracy can be seen below. It decides if two questions match based on a given threshold, and then looks at the label to see if they truly match or not and returns a 1 if we were right and a 0 if we were wrong.

```
def ComputeAccuracy(row, thr, score):
```

```
    """
    Input: A row of data, the threshold, and the score.
    Output: 1 if the prediction for that row was correct and 0 otherwise.
    """
    import numpy as np

    # Find the label:
    label = np.sign(score - thr)

    # If correctly predicted a duplicate, increase correctCount:
    if label >=  0 and row["duplicate"] == 1:
        correct = 1
    # If correctly predicted a non-duplicate, increase correctCount:
    elif label < 0 and row["duplicate"] == 0:
        correct = 1
    # Prediction was wrong:
    else:
        correct = 0

    return correct
```

The main function for this problem gets the command line arguments and reads in the data. It then loops over each row of the data, gets the score using the function from problem 2, and keeps count of how accurate our predictions were using the function above. It returns the number correct divided by the total number of pairs.

Results:

1. Results for different thresholds:

Threshold	Accuracy
0.1	0.434780994688
0.2	0.498903349347
0.3	0.573914872686
0.4	0.635030331903
0.45	0.65581255781
0.5	0.663889722419
0.55	0.664684755473
0.6	0.660100167978
0.65	0.652991273197
0.7	0.646426836851
0.8	0.633233000368
1.0	0.634062062074

The threshold of 0.55 gives me the best result.

2. Use the best threshold, and run the code on validation.csv. What is the validation accuracy?
The accuracy is 0.663478625728. This is very close to the training accuracy with 0.55.

3. Briefly discuss your findings.
The accuracy improves as the threshold increases until it reaches 0.55 where the accuracy peaks and the decreases slowly. So a question pair with an overlapping score of at least 0.55 gives the highest accuracy of predictions. This is interesting that an overlapping score just above 50% does better than any of the higher scores. Overall, this method of matching Quora question pairs is not that successful. The best accuracy is only 66%.

Full code can be downloaded here: https://katherineolson.github.io/141C_HW1_problem3.py

Problem 4

Problem: One problem of our approach is that the overlapping score might be dominated by the "stop words", such as "is", "a", "the", "what". Now we try to improve our model by removing the stop words. Note that if all the words in both questions are stop words, just report overlapping score to be 0. Now we define the stop words by all the words that appeared more than 10000 times in "training.csv". Now, try to modify ComputeAccuracy.py by removing the stop-words before computing overlapping scores.

Coding Process:

My function for finding the stop words can be seen below. It returns a list with all of the words that appear more than 10,000 times in training.csv.

```
def StopWords():
    """
    Input: A dataframe containing the questions and some other info.
    Output: A list of stop words (all words that appear more than 10000 times).
    """
    from problem1 import preprocess
    from collections import Counter
    import pandas as pd
    train = pd.read_csv("training.csv", header = None, names = ["qid", "qid2", "question1",
"question2", "duplicate"])

    counts = Counter(preprocess(" ".join(train['question2'].astype(str) +
train["question1"].astype(str))).split())
```

```
return [word for c, word in zip(counts.values(), counts) if c > 10000]
```

To modify the ComputeOneScore function from problem 2 to remove stop words, I just added in the following two lines in the middle of the function before the for loops. I also included the stop words as parameters of the function.

```
# Remove stop words from q1 and q2:
q1 = [word for word in q1 if word not in stopWords]
q2 = [word for word in q2 if word not in stopWords]
```

The main function for this problem does almost exactly the same thing as the main function from problem 3, but it calls the stop words function and then passes the stop words into ComputeOneScore.

Results:

1. The results for different thresholds:

Threshold	Accuracy
0.1	0.485319730122
0.2	0.529606474106
0.3	0.60063973866
0.4	0.642501167801
0.45	0.661207645929
0.5	0.664619791683
0.55	0.66578913991
0.6	0.664062959193
0.65	0.659756787943
0.7	0.654903064744
0.8	0.651311495188
1.0	0.66013419663

The best threshold is 0.55.

2. Use the best threshold, and run the code on validation.csv. What is the validation accuracy?

The accuracy is 0.663614374568. This is slightly lower than the result for the training data.

3. Briefly discuss your findings.

Overall, removing stop words slightly increased accuracy for all thresholds. The change was very small, but there was improvement. Using an overlapping score of at least 0.55 as matching has the best results again. So removing stop words did not have much of an effect on which thresholds did the best.

Full code can be downloaded here: https://katherineolson.github.io/141C_HW1_problem4.py

Problem 5

Problem: In problem 3 and 4, we try all the threshold manually. Is there a better way to automatically select the best threshold? Try implement your method and report your findings.

Coding Process:

I solved this problem with a shell script that loops through each threshold and prints out the best one. I got the same results as problems 2 and 3. Here is the script:

```
#!/bin/bash

program=$1
runOn=$2
result=0
shift 2

for i in $@; do # For each threshold
  resultNew=`python2.7 $program $runOn $i` # Get the accuracy
  improve="$resultNew > $result"
  #improve=$improve|bc
  improve=`bc <<< $improve` # Boolean greater than
  if [ $improve -eq 1 ] # If the accuracy beats all past accuracies
  then
        result=$resultNew # Save the new best accuracy
        thr=$i # Save the new best threshold
  fi
done

echo The best threshold is $thr with accuracy $result
```

Full code can be downloaded here: https://katherineolson.github.io/141C_HW1_problem5.sh