PROGRAMMING ASSIGNMENT -2

CSE 574 Group: Programming Assignment 3

Submitted by:

- Sahil Kapahi (50317075)
- Nihar Patel (50318506)
- Salil Dabholkar (50321748)

APRIL 13, 2020

Contents

Table of Figures

List of Tables

CSE474/574: Programming Assignment 2

Non-linear Models for Supervised Learning

REPORT-1

The word feature in the approach 1 used in Task 1 of the assignment is the log of positive-negative ratio of individual words in the review. The log ratio is calculated only for words in the training data that occur more than 100 times throughout the sample training set as implemented in the part of code shown below.

```
for term,cnt in list(total_counts.most_common()):
    if(cnt > 100):
        # TODO: Code for calculating the ratios (remove the next
        neg = negative_word_count[term]
        neg = 1 if neg == 0 else neg

        pos_neg_ratios[term] = positive_word_count[term] / neg
```

Figure 1 Sample Implementation for calculating ratios of words occurring more than 100 times

The final calculated ratio is the logarithm of, the number of times the word occurred in a positive training example to the number of times the word occurs in negative training examples.

This turns out to be positive for the words that have a numeric ratio of more than one and negative for the words that have numeric ratio less than one as it is the property of log scale.

```
# take a log of the ratio
for word,ratio in pos_neg_ratios.most_common():
    pos_neg_ratios[word] = np.log(ratio)
```

Figure 2 Log of positive-negative ratio

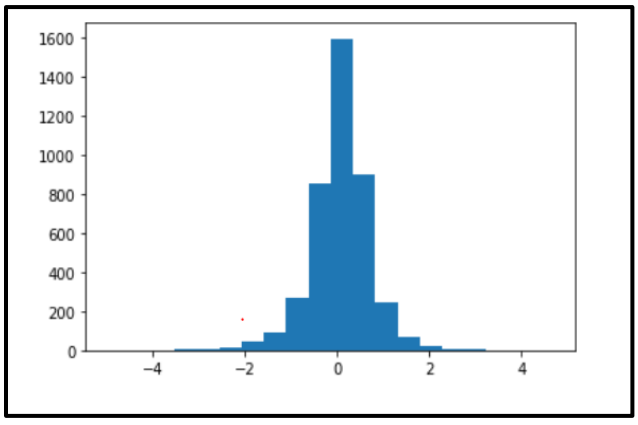

Figure 3 Histogram of log ratio scores

Histogram shows that the number of words with a score of 0 are the neutral words whereas those on the positive or negative side correspond to positive and negative scores respectively.

Approach

For classifying a review in a non-ML way, we simply sum the log scores of individual words and classify the whole review based on if the sum is positive or not. Some words are simply "neutral" but still might have small magnitudes. These words have to be skipped no matter how many there are. So, we had to fix a threshold. Experimenting with a lot of values, we found 0.65 as the best threshold.

Thus, the words in each of the reviews with **log ratio of more than a certain absolute value (absolute ratio of more than 0.65 in our implementation) become our features** as they contribute towards the sum of features calculated for each review. The words with log values above +0.65 in the implementation have been considered to be a positive whereas a score of less than -0.65 is considered negative. Anything in-between is considered neutral.

```
def nonml_classifier(review,pos_neg_ratios):
    '''
    Function that determines the sentiment for a given review.

    Inputs:
      review - A text containing a movie review
      pos_neg_ratios - A Counter object containing frequent words
                       and corresponding log positive-negative ratio
    Return:
      sentiment - 'NEGATIVE' or 'POSITIVE'
    '''
    # TODO: Implement the algorithm here. Change the next line.
    total = sum(filter(lambda x : abs(x) >= 0.65, [pos_neg_ratios[word] for word in review.split(' ')]))
    return 'NEGATIVE' if total < 0 else 'POSITIVE'
```

Figure 4 Log ratios of more than 0.65 absolute value considered as word features in the implementation

Classifier: The review is hence classified as NEGATIVE if the sum of these word features is a negative number, but is classified as POSITIVE if the sum of word features is not a negative number.

The final accuracy of the non-ML based classifier is calculated as the number of reviews rightly classified to the total number of reviews in the test data.

```
predictions_test = []
for r in reviews_test:
    l = nonml_classifier(r,pos_neg_ratios)
    predictions_test.append(l)

# calculate accuracy
correct = 0
for l,p in zip(sentiments_test,predictions_test):
    if l == p:
        correct = correct + 1

print('Accuracy of the model = {}'.format(correct/len(sentiments_test)))
```

```
Accuracy of the model = 0.821
```

Figure 5 Test accuracy calculation for the reviews based of the calculated log of pos-neg ratio

As can be seen from the above implementation that we are getting an accuracy of **82.1% on test-data** for the non-ML based classifier.

REPORT-2

A vanilla neural network has been used in the second approach for training and calculation of accuracy on test data. The implementation takes the complete vocabulary of words with total repetitions of more than 100 times in the entire data set and out of that ignores the words with absolute log pos-neg ratio of less than 0.2. This creates a word to index mapping for all the words which can act as a feature vector in the training process.

```
# create a word2index mapping from word to an integer index
word2index = {}
ignore_words = find_ignore_words(pos_neg_ratios)
vocab_selected = list(set(vocab_selected).difference(set(ignore_words)))
for i,word in enumerate(vocab_selected):
    if word not in ignore_words:
        word2index[word] = i
vocab_size = len(word2index)
```

Figure 6 Vocabulary size for input to neural network

The input vector to the neural network has a length equal to vocab_size and is in the form of a vector of counts for each word index as they occur in the training review data. The corresponding output label is a one-hot encoded label for 2 classes, POSITIVE and NEGATIVE.

In the first approach for this task, we developed a simple word-level features which is sum of the log ratio of the words whereas in this approach we develop a neural-net to learn the weights based on the input feature vector described above which ignores the words with log pos-neg ration of less than 0.2.

Using the Neural Network as a classifier the accuracy was achieved to **84% on the test-data** while **97.49% on the train-data**. The time it took for the Neural Network to complete all the epochs was **16.9543 seconds** on the Cloud GPU.

While **using the non-ML based classifier**, we are getting an accuracy of **82.1% on test-data**. Therefore, it is better than the rule-based classifier used in earlier approaches as it was a simple sum of the word features.

Below table shows the compared results:

Approach	**Test Accuracy**	**Train Accuracy**
Non-ML Based Classifier	82.1 %	N/A
Vanilla Neural Network	84%	97.49%

Table 1 Accuracy Summary of both classifiers

REPORT-3

We experimented with different epochs (50, 75, 100), number of hidden layers (1, 3, 5) and hidden layer width (10, 30, 50). We tried all combinations of the above, thus giving us 27 results. These have been tabulated on the next page.

From our experiments, we found the following trends. We have **highlighted** the best result in the table.

Epochs:

Increasing the epochs slightly increased the training accuracy with hardly any increase in the test accuracy. The training was already mostly done in 50 epochs, so increasing the number of epochs didn't help much in this case.

As expected, the time required for training increased with the number of epochs.

No. of hidden layers:

Increasing the number of layers almost always slightly decreased both the training and testing accuracies slightly. We believe this is due to the vanishing gradient problem.

Increasing the number of hidden layers increases the training time as there are (exponentially) more parameters to update.

Hidden layer width:

Increasing the number of units in each hidden layer slightly increased the training as well as test accuracies. The reason it happens because the model becomes more adaptive with increased number of neurons thus it can learn smaller details. Although in some cases it reduces the accuracy on the test data due to the over-fitting on training data and thus the generalization of the classification model can also decrease.

Increasing the number of units in each hidden layer didn't have any significant effects on the training time as the number of parameters to train increased only slightly.

Overall, we believe that for this dataset, increasing the hidden layer width had the most significant impact (comparatively) with the least drawbacks (least impact on time required for training).

Below are the results that we noted on varying the complexity of the model:

EPOCH: 50								
		No of Layers						
		1		3		5		
Unit Width of hidden layer	10	Test Accuracy:	84%	Test Accuracy:	84.50%	Test Accuracy:	83.38%
		Training Accuracy:	97.49%	Training Accuracy:	95.56%	Training Accuracy:	94.60%
		Time:	**16.9543(sec)**	Time:	18.317(sec)	Time:	19.974(sec)
	30	Test Accuracy:	84.75%	Test Accuracy:	84.63%	Test Accuracy:	83.25%
		Training Accuracy:	98.63%	Training Accuracy:	96.63%	Training Accuracy:	95.41%
		Time:	17.4688(sec)	Time:	18.88(sec)	Time:	20.13(sec)
	50	Test Accuracy:	**84.75%**	Test Accuracy:	83.25%	Test Accuracy:	84.50%
		Training Accuracy:	98.95%	Training Accuracy:	95.55%	Training Accuracy:	96.08%
		Time:	17.265(sec)	Time:	18.083(sec)	Time:	20.09(sec)

Table 2 Results for Epoch 50 with different no of layers and unit width.

EPOCH: 75							
		No of Layers					
		1		3		5	
Unit Width of hidden layer	10	Test Accuracy:	83.63%	Test Accuracy:	83.88%	Test Accuracy:	83.38%
		Training Accuracy:	97.81%	Training Accuracy:	95.77%	Training Accuracy:	94.63%
		Time:	25.8963(sec)	Time:	27.952(sec)	Time:	29.04(sec)
	30	Test Accuracy:	84.38%	Test Accuracy:	84.13%	Test Accuracy:	83.75%
		Training Accuracy:	**99%**	Training Accuracy:	97.02%	Training Accuracy:	95.94%
		Time:	25.1232(sec)	Time:	27.1159(sec)	Time:	49.37(sec)
	50	Test Accuracy:	84.50%	Test Accuracy:	83.63%	Test Accuracy:	82.63%
		Training Accuracy:	98.81%	Training Accuracy:	97.37%	Training Accuracy:	96.27%
		Time:	25(sec)	Time:	27.324(sec)	Time:	29.3562(sec)

Table 3 Results for Epoch 75 with different no of layers and unit width.

EPOCH: 100							
		No of Layers					
		1		3		5	
Unit Width of hidden layer	10	Test Accuracy:	83.38%	Test Accuracy:	83.88%	Test Accuracy:	83.25%
		Training Accuracy:	94.73%	Training Accuracy:	95.96%	Training Accuracy:	95.25%
		Time:	61.99(sec)	Time:	35.3638(sec)	Time:	65.335(sec)
	30	Test Accuracy:	84.38%	Test Accuracy:	82.13%	Test Accuracy:	83.38%
		Training Accuracy:	94.16%	Training Accuracy:	97.31%	Training Accuracy:	96.71%
		Time:	67.36(sec)	Time:	36.4988(sec)	Time:	68.668(sec)
	50	Test Accuracy:	83.38%	Test Accuracy:	84.88%	Test Accuracy:	83.63%
		Training Accuracy:	93.76%	Training Accuracy:	**97.72%**	Training Accuracy:	96.88%
		Time:	81.19(sec)	Time:	36.149(sec)	Time:	45.128(sec)

Table 4 Results for Epoch 100 with different no of layers and unit width.

REPORT-4

1. We have been provided with a subsampled Quick Draw data set consisting of 125,000 drawings across 10 classes. The data has been further segregated into 100,000 samples for training data and 25,000 for testing purpose.
   The sequential model API has been further used to create neural network in the form of multi-layer perceptron.

   The default setting with single hidden layer is used to train the model initially and the results have been attached as follows:

   Training Accuracy with single layer and resolution of 28x28 is **80.37%.**

   Test Accuracy with single layer and resolution 28x28 is **68.14%**

   Run Time – 01hr 07 min 30 secs

2. **3-Layers Neural Network Results**

   Training Accuracy with 3 layers and resolution of 28x28 is **23.98%.**

   Test Accuracy with 3 layer and resolution 28x28 is **25.71%**

   Run Time – 01hr 23 min 12 secs

   5-Layers Neural Network Results

   Training Accuracy with 5 layers and resolution of 28x28 is **9.94%.**

   Test Accuracy with 5 layer and resolution 28x28 is **10.00%**

   Run Time – 01hr 28 min 50 secs

   The performance went down on both training and testing data as more and more layers are added. This can be due to vanishing gradients with increasing the number of layers and the weights may not get updated appropriately as it gets difficult to avoid local minima. However, the test accuracy for these cases is very close to the training accuracy as model avoids overfitting and thus results in test accuracy very close to training accuracy.

   Further, excel file and models have been attached along with the report.

3. **1-Layers Neural Network Results**

Resolution (20 x 20)

Training Accuracy with 1 layers and resolution of 20x20 is **73.66%.**

Test Accuracy with 1 layer and resolution 20x20 is **66.41%**

Run Time – 01hr 05 min 1 secs

Resolution (15 x 15)

Training Accuracy with 1 layers and resolution of 15x15 is **75.15%.**

Test Accuracy with 1 layer and resolution 15x15 is **69.34%**

Run Time – 00hr 53 min 17 secs

Resolution (10 x 10)

Training Accuracy with 1 layers and resolution of 15x15 is **70.72%.**

Test Accuracy with 1 layer and resolution 15x15 is **64.24%**

Run Time – 00hr 53 min 23 secs

Resolution (5 x 5)

Training Accuracy with 1 layers and resolution of 15x15 is **68.79%.**

Test Accuracy with 1 layer and resolution 15x15 is **68.16%**

Run Time – 00hr 30 min 59 secs

TABULAR RESULTS

Train Accuracy:	Test Accuracy:	Epochs 500
28 x 28		
80.37%	68.14%	1h 7min 30s
20 x 20		
73.66%	66.41%	58m 1s
15 x 15		
75.15%	69.34%	53m 17s
10 x 10		
70.72%	64.24%	53m 23s
5 x 5		
68.79%	68.16%	30m 59s

TRAIN ACCURACY DECON LOW RESOLUTION

TRAIN TIME DECON LOW RESOLUTION

Table 5 Results

As can be seen from the above results that on decreasing the resolution of the original dataset, the training accuracy decreases drastically whereas the test accuracy remains close to training accuracy on decreasing resolution which makes it evident that it avoids overfitting, keeping training and test accuracy close. As the resolution of the image is reduced, the training time is also reduced as the input data vector is shortened requiring a smaller number of weight updates.