NLP CW2

Horia Radu - k55592hr

Task 1:

Methods Approach 1:

When getting the data, I read through all the lines in the reviews, skip the tags and then select only the sentences in the lines which are split by the “##”. This way I don’t get the review words, tags and numbers from each line, only the review sentence text.

I used wordpunct_tokenizer for the tokenization, as it would perfectly separate words from their punctuation and other non syntax friendly characters, like hyphens, apostrophes. Then I removed the english stopwords, from nltk, plus “u” as it was used a lot instead of “you” and it was not in the initial stopwords. I remove all the tokens that do not contain letters (punctuation mostly) and those that are just numbers, while also case folding. All tokens that contain only 1 character are also removed

I decided on using lemmatization as this would help when finding matching features for words, while still mostly keeping the semantics and the syntax of the word and sentence. Also through testing I’ve noticed my distributional semantics approach performs better when I use it. The top 50 words are selected using a frequency distribution.

This approach is a sparse matrix approach, with a context feature matrix made up of the target words and a stemmed dictionary/vocabulary from the reviews.

For better performance, I use a positional index to better access those target words. I uniformly and randomly sample half of the occurrences of the target words and swap them with their pseudo words.

First, I create a vocabulary to use in the term feature matrix, then with the help of the positional index I iterate through all the occurrences of the target and pseudo words and look to the left and right a certain window size, that is set as a hyperparameter. After creating the 100 by vocabulary size term context matrix, I remove all the empty columns for words that do not appear at all near the target or pseudo words, in the respective window size. Thus I make the matrix more managable and less noisy.

Finally, using a K-means clustering algorithm, implemented in sklearn, I create 50 clusters into which I fit my matrix. I then get the cluster labels for my target and pseudo words, and check to see how many of them match.

Lastly, I repeat this process from uniformly randomly sampling the words, without the preprocessing, n number of times to get more data on how my system performs, getting a mean and standard deviation of the accuracies

Methods Approach 2:

The preprocessing and text cleaning is the same as in Approach 1.

This approach is a dense matrix approach.

Also the dictionary is created differently. Instead of using the stems of words, I decided to go without stemming so as to get a more accurate context of each word.

After I get the sparse feature matrix, I use the latent semantic indexing technique with the results from the truncated singular value decomposition, implemented in sklearn TruncatedSVD to transform it into a dense matrix. Also to this pipeline I add a normalizer to reduce each sample in the dense matrix to a more normal form, more similar to each other, avoiding issues which can be caused by larger values in the samples “taking over” clusters.

Then with this dense normalized matrix I use the same K-means clustering algorithm as in Approach 1.

Result Analysis:

Results for Approach 2 are significantly better than for Approach 1. This is the case as with the dense matrix we have less noise, thus the clustering is more accurate.

The stemming against the word for the feature matrix vocabulary were not as significant as the matrix difference. I noticed that when I did not stem the words, I would get a little better results which I believe is because we keep more of the context of each target and pseudo word.

Regarding the window size, from testing with multiple sizes, from 1 to 10, I found that a narrower window size gets higher clustering accuracy. As we don't have a very large dataset and very big sentences, with a lot of reviews with different topics, we can not capture semantics all that well, so the focus needs to be on the syntax. Both reasons made me choose 1 as the window size as it performed the best.

For the dense approach, going with Latent Semantic Indexing using a truncated SVD, through testing I found that a higher dimensionality dense matrix performs better, but the results stagnate at 100. After 100 the results did not improve as they did from 2 to 100, not noticing any difference in accuracies. Basically I did an explained variance on the dimensionality that the dense matrix should have and chose the best performance-cost implementation.

The accuracy of both approaches depended the most on how the clustering algorithm can handle the matrices received, the sparse being harder and dense being easier. While window size, context extraction and construction of the distributional semantic representation also matter, the biggest difference is seen in the matrix types, with differences in over 20% accuracy.

Task 2:

Method:

I decided on using a transformer as I found it most fit for the task at hand.

I implemented it in pytorch, with torchtext from the same library as to process the vocabulary for the transformer as a tensor. For the model I used I learnt more in depth from their official https://pytorch.org/ page. My transformer is made out of a positional encoder, multiple encoder layers which then go through the transformer encoder, an embedding and a decoder. I also initialize its weights based on uniform distribution values from Glorot initialization, while adding a bias.

The data is collected from all the reviews which have a score and are split appropriately, using RegEx. I considered the neutral reviews as positive as to get as much data for the transformer as possible. The only preprocessing I do to those reviews is case folding. I then split this data into learning data, for training and evaluating, and testing data. Using word punct tokenization from nltk I create a vocabulary for the model, to which I add 4 elements: "<unk>", "<pad>", "<bos>", "<eos>". They are for the transformer to classify as unknown, padding, beginning and end of sentence.

Next I create the transformer model using the vocabulary as embeddings, setting the number of classes to 2 (positive and negative), setting the dimension of the model, the number of attention heads, the dimension of the hidden layer, the number of encoder layers and the dropout to prevent overfitting in training mode.

I have a DataLoader function that trains and evaluates the model in batches as they are less noisy and easier to work with, while also putting a lot less pressure on the processing device (CPU or GPU).

My training function is teaching the model an epoch number of times, training and evaluating in each epoch. In this training function I make use of a Softmax function to calculate the accuracy, a stochastic gradient descent method for optimizing the training and a cross-entropy loss function to measure the performance of the classification model. I apply 5-fold cross validation on the learning data to train and evaluate it. Lastly, I find the model which proved to have the smallest loss and choose it as the best one.

Experiment and Result Analysis:

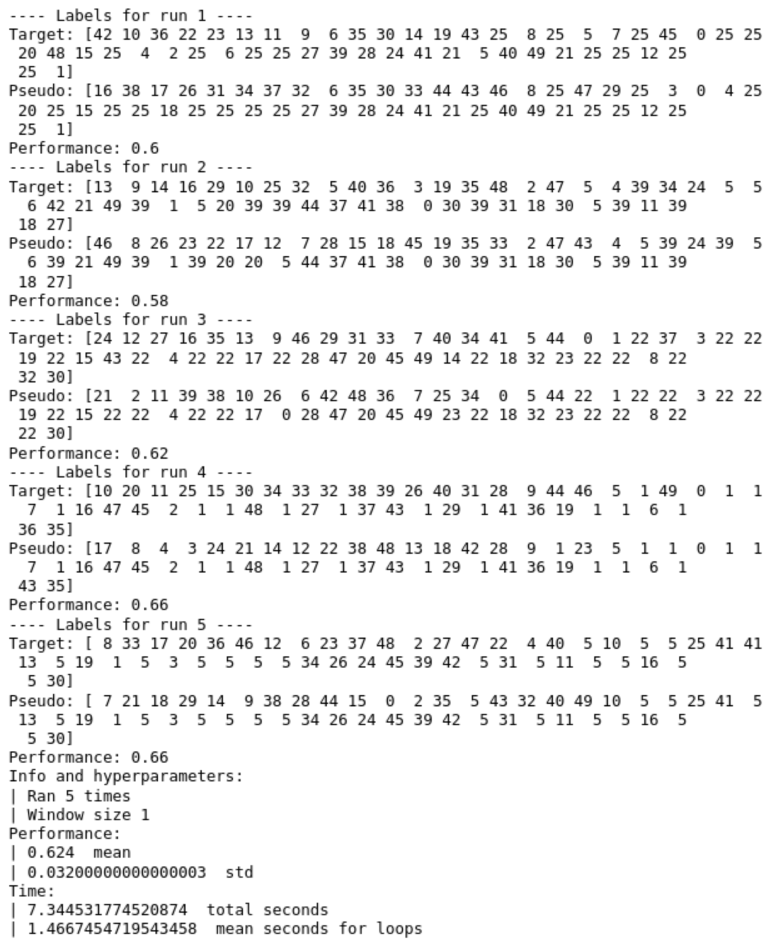
The most important hyperparameters are in the Transformer model. To this we can also add the number of epochs the model computes and the partitioning of the learning data into training and testing. Those also depend on the torch device used for processing CUDA

(GPU) or CPU, as with the GPU it is much faster and we can allow the model a bigger dimension with more heads and layers, which improve the performance, learning capacity and ability. Stochastic Gradient Descent optimization also has the learning rate, momentum and weight decay which dictate how the model improves in each training iteration. The epoch number can be changed, lower will mean worse accuracy, but without CUDA epoch will be fairly slow.

This neural network can be continuously improved as long as it has enough processing power and enough data. The k-fold cross validation proves to be a good way of training and evaluating the model, because it iterates through all the partitioned data and teached the model the most it can out of the learning data.

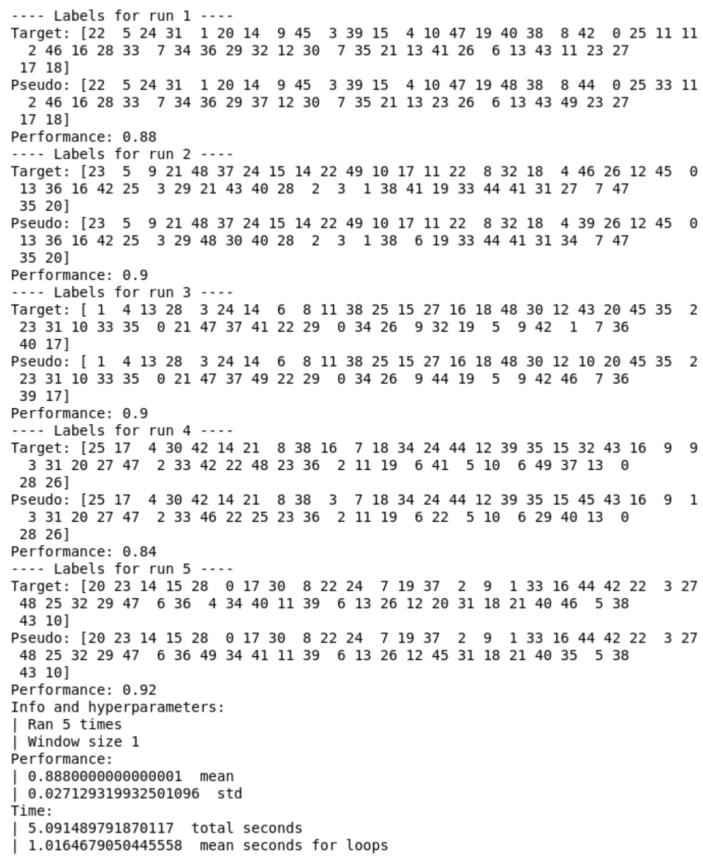
On the evaluation data I succeeded in getting over 95% accuracy with a very low loss function, but that is the case because we do the k-fold, though increasing the dropout from the transformer can force the model to learn more. This though proves that the NN works. Then after getting this in the evaluation model, I tested the model on a small testing set that I initially split from the main data. On this test set I got around 74-76 accuracy with 15 epochs. This is a huge improvement from my initial tests where I did not use k-cross and had a worse learning rate and no dropout, in which I was getting 50-60 accuracy on even more runs, and my training model constantly overfitted.

Through this experiment I proved that the transformer actually learns patterns and connections of words, getting better the more he runs.



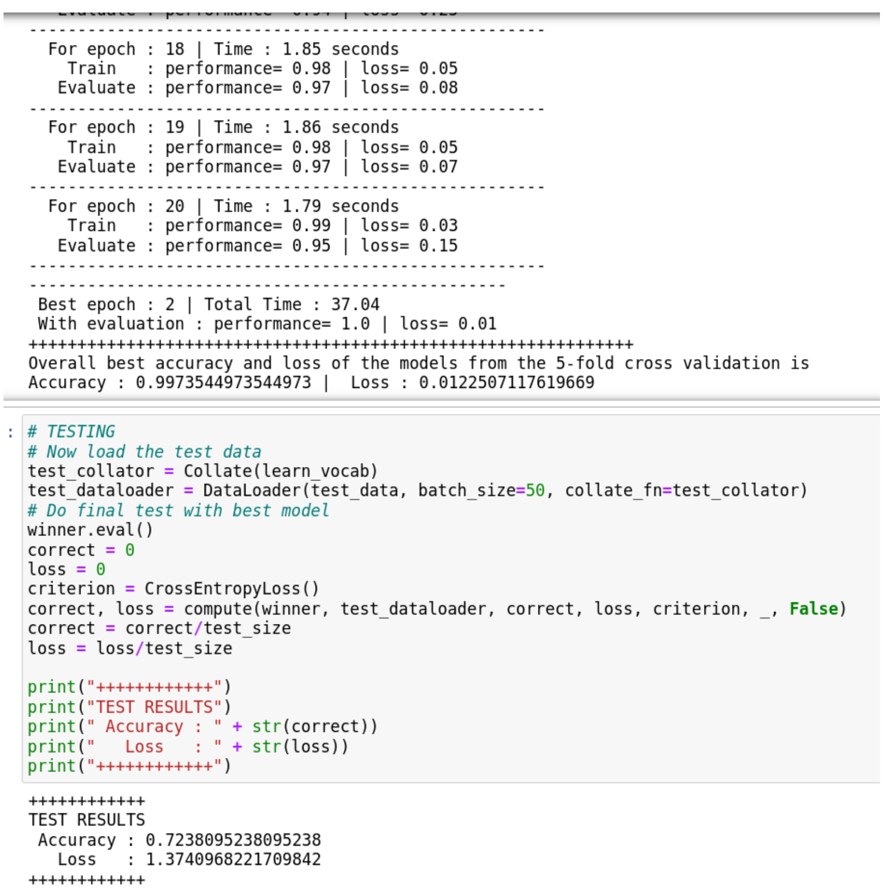
```
---- Labels for run 1 ----
Target: [42 10 36 22 23 13 11  9  6 35 30 14 19 43 25  8 25  5  7 25 45  0 25 25
 20 48 15 25  4  2 25  6 25 25 27 39 28 24 41 21  5 40 49 21 25 25 12 25
 25  1]
Pseudo: [16 38 17 26 31 34 37 32  6 35 30 33 44 43 46  8 25 47 29 25  3  0  4 25
 20 25 15 25 25 18 25 25 25 25 27 39 28 24 41 21 25 40 49 21 25 25 12 25
 25  1]
Performance: 0.6
---- Labels for run 2 ----
Target: [13  9 14 16 29 10 25 32  5 40 36  3 19 35 48  2 47  5  4 39 34 24  5  5
  6 42 21 49 39  1  5 20 39 39 44 37 41 38  0 30 39 31 18 30  5 39 11 39
 18 27]
Pseudo: [46  8 26 23 22 17 12  7 28 15 18 45 19 35 33  2 47 43  4  5 39 24 39  5
  6 39 21 49 39  1 39 20 20  5 44 37 41 38  0 30 39 31 18 30  5 39 11 39
 18 27]
Performance: 0.58
---- Labels for run 3 ----
Target: [24 12 27 16 35 13  9 46 29 31 33  7 40 34 41  5 44  0  1 22 37  3 22 22
 19 22 15 43 22  4 22 22 17 22 28 47 20 45 49 14 22 18 32 23 22 22  8 22
 32 30]
Pseudo: [21  2 11 39 38 10 26  6 42 48 36  7 25 34  0  5 44 22  1 22 22  3 22 22
 19 22 15 22 22  4 22 22 17  0 28 47 20 45 49 23 22 18 32 23 22 22  8 22
 22 30]
Performance: 0.62
---- Labels for run 4 ----
Target: [10 20 11 25 15 30 34 33 32 38 39 26 40 31 28  9 44 46  5  1 49  0  1  1
  7  1 16 47 45  2  1  1 48  1 27  1 37 43  1 29  1 41 36 19  1  1  6  1
 36 35]
Pseudo: [17  8  4  3 24 21 14 12 22 38 48 13 18 42 28  9  1 23  5  1  1  0  1  1
  7  1 16 47 45  2  1  1 48  1 27  1 37 43  1 29  1 41 36 19  1  1  6  1
 43 35]
Performance: 0.66
---- Labels for run 5 ----
Target: [ 8 33 17 20 36 46 12  6 23 37 48  2 27 47 22  4 40  5 10  5  5 25 41 41
 13  5 19  1  5  3  5  5  5  5 34 26 24 45 39 42  5 31  5 11  5  5 16  5
  5 30]
Pseudo: [ 7 21 18 29 14  9 38 28 44 15  0  2 35  5 43 32 40 49 10  5  5 25 41  5
 13  5 19  1  5  3  5  5  5  5 34 26 24 45 39 42  5 31  5 11  5  5 16  5
  5 30]
Performance: 0.66
Info and hyperparameters:
| Ran 5 times
| Window size 1
Performance:
| 0.624  mean
| 0.032000000000000003  std
Time:
| 7.344531774520874  total seconds
| 1.4667454719543458  mean seconds for loops
```

Task 1 Approach 1 with verbose to see all the loops clusters and performance



```
---- Labels for run 1 ----
Target: [22  5 24 31  1 20 14  9 45  3 39 15  4 10 47 19 40 38  8 42  0 25 11 11
  2 46 16 28 33  7 34 36 29 32 12 30  7 35 21 13 41 26  6 13 43 11 23 27
 17 18]
Pseudo: [22  5 24 31  1 20 14  9 45  3 39 15  4 10 47 19 48 38  8 44  0 25 33 11
  2 46 16 28 33  7 34 36 29 37 12 30  7 35 21 13 23 26  6 13 43 49 23 27
 17 18]
Performance: 0.88
---- Labels for run 2 ----
Target: [23  5  9 21 48 37 24 15 14 22 49 10 17 11 22  8 32 18  4 46 26 12 45  0
 13 36 16 42 25  3 29 21 43 40 28  2  3  1 38 41 19 33 44 41 31 27  7 47
 35 20]
Pseudo: [23  5  9 21 48 37 24 15 14 22 49 10 17 11 22  8 32 18  4 39 26 12 45  0
 13 36 16 42 25  3 29 48 30 40 28  2  3  1 38  6 19 33 44 41 31 34  7 47
 35 20]
Performance: 0.9
---- Labels for run 3 ----
Target: [ 1  4 13 28  3 24 14  6  8 11 38 25 15 27 16 18 48 30 12 43 20 45 35  2
 23 31 10 33 35  0 21 47 37 41 22 29  0 34 26  9 32 19  5  9 42  1  7 36
 40 17]
Pseudo: [ 1  4 13 28  3 24 14  6  8 11 38 25 15 27 16 18 48 30 12 10 20 45 35  2
 23 31 10 33 35  0 21 47 37 49 22 29  0 34 26  9 44 19  5  9 42 46  7 36
 39 17]
Performance: 0.9
---- Labels for run 4 ----
Target: [25 17  4 30 42 14 21  8 38 16  7 18 34 24 44 12 39 35 15 32 43 16  9  9
  3 31 20 27 47  2 33 42 22 48 23 36  2 11 19  6 41  5 10  6 49 37 13  0
 28 26]
Pseudo: [25 17  4 30 42 14 21  8 38  3  7 18 34 24 44 12 39 35 15 45 43 16  9  1
  3 31 20 27 47  2 33 46 22 25 23 36  2 11 19  6 22  5 10  6 29 40 13  0
 28 26]
Performance: 0.84
---- Labels for run 5 ----
Target: [20 23 14 15 28  0 17 30  8 22 24  7 19 37  2  9  1 33 16 44 42 22  3 27
 48 25 32 29 47  6 36  4 34 40 11 39  6 13 26 12 20 31 18 21 40 46  5 38
 43 10]
Pseudo: [20 23 14 15 28  0 17 30  8 22 24  7 19 37  2  9  1 33 16 44 42 22  3 27
 48 25 32 29 47  6 36 49 34 41 11 39  6 13 26 12 45 31 18 21 40 35  5 38
 43 10]
Performance: 0.92
Info and hyperparameters:
| Ran 5 times
| Window size 1
Performance:
| 0.8880000000000001  mean
| 0.027129319932501096  std
Time:
| 5.091489791870117  total seconds
| 1.0164679050445558  mean seconds for loops
```

Task 1 Approach 2 with verbose to see all the loops clusters and performance



```
  --------------------------------------------------
    For epoch : 18 | Time : 1.85 seconds
      Train    : performance= 0.98 | loss= 0.05
     Evaluate : performance= 0.97 | loss= 0.08
  --------------------------------------------------
    For epoch : 19 | Time : 1.86 seconds
      Train    : performance= 0.98 | loss= 0.05
     Evaluate : performance= 0.97 | loss= 0.07
  --------------------------------------------------
    For epoch : 20 | Time : 1.79 seconds
      Train    : performance= 0.99 | loss= 0.03
     Evaluate : performance= 0.95 | loss= 0.15
  --------------------------------------------------
  --------------------------------------------------
   Best epoch : 2 | Total Time : 37.04
   With evaluation : performance= 1.0 | loss= 0.01
  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  Overall best accuracy and loss of the models from the 5-fold cross validation is
  Accuracy : 0.9973544973544973 |  Loss : 0.0122507117619669
```

```
: # TESTING
  # Now load the test data
  test_collator = Collate(learn_vocab)
  test_dataloader = DataLoader(test_data, batch_size=50, collate_fn=test_collator)
  # Do final test with best model
  winner.eval()
  correct = 0
  loss = 0
  criterion = CrossEntropyLoss()
  correct, loss = compute(winner, test_dataloader, correct, loss, criterion, _, False)
  correct = correct/test_size
  loss = loss/test_size

  print("++++++++++++")
  print("TEST RESULTS")
  print(" Accuracy : " + str(correct))
  print("   Loss   : " + str(loss))
  print("++++++++++++")
```

```
  ++++++++++++
  TEST RESULTS
   Accuracy : 0.7238095238095238
     Loss   : 1.3740968221709842
  ++++++++++++
```

Task 2 with 20 epochs, k-fold cross evaluation and then testing on a small data set that wasn't used at all (90% learning and evaluating (k-fold) and 10% testing)