Task 1: Distributional Semantics

Method Description

The text cleaning and pre-processing steps are as follows: first the data from the review files is read into an array of string containing all 9 of the documents, done in the `read_data` function. Then the `preprocess` function is called, this sends each document to a helper function `process_raw`, this function splits the review file into its separate lines and then removes all instances of the "`[t]`" tag. Then for each line in the file the code first looks for the presence of the delimiter "`##`", which separates classification tags from the review line, if the delimiter isn't present the line is fine to process, if the delimiter is present the code only takes the data in the line after the delimiter. Then the line is converted to lowercase, non-alphanumeric characters (punctuation etc.) is removed, the line is split into tokens using `word_tokenize` from the `nltk` module, any stop words present in the `ENGLISH_STOP_WORDS` set provided by the `sklearn` module are removed, and finally the tokens left are stemmed using the `SnowballStemmer` provided by the `nltk` module. To explain these steps, the tags from the dataset are removed because, the "`[t]`" tags are not needed as they define the start of a review yet are not present correctly in every file (missing from iPod file and only 1 present in the Canon PowerShot SD500 file) so I didn't use them in the rest of the code, and use of splitting on the delimiter has been explained. Also, punctuation was removed as this isn't useful when clustering words with my model, I kept alphanumeric characters because the names of products contain both of these characters so are useful (for example the word "mp3" appears in my top 50 words). Finally, I decided to stem as this reduced the size of my vocabulary whilst not losing much meaning, so the construction of my sparse feature vector was faster.

The construction of my distribution sematic representations is done in the `construct_feature_mat` function, firstly it creates a term vector (`term_vec`) which is the 50 target words plus the 50 pseudo-words. For each term a feature vector is instantiated as a list of 0s the size of the vocab list, then it checks every instance of the term in all the pre-processed reviews and looks at a context window either side of the term. For each token (word) present in the context window the index for that token in the feature vector vocab list is replaced with a 1. So, it ends with a term being represented by the vocab list with a 1 if that word in the vocab list occurs in the context of the term. After this I get a sparse feature matrix of size 100 by length of the vocab list. This is then converted to dense representation for each term through singular value decomposition, to result in a matrix of size 100 by the `matrix_ran`k of the feature matrix.

Result Analysis

My clustering accuracy (the percentage of correct pairs) averaged 65.6% (in the supplied output and in Figure 1), it is usually in the range 65%-70%. This is obtained by randomly replacing the half the occurrences of the target words with their respective pseudo-words, constructing a feature matrix to represent each target/pseudo word, and then clustering them using Ward Hierarchical clustering provided by the `sklearn` module. This is repeated 10 times to assess how consistent and reproducible the clustering accuracy is. I believe that the result obtained is an acceptable accuracy for the terms given the scope of the task, and the relatively small dataset, it is also consistent this is shown in the small standard deviation of 3.2 (Figure 1) and it is a reproducible accuracy as seen in the list of accuracies (Figure 1) being close, with only a range of 10.

My solution does contain a hyper-parameter which is the context window used to determine the distributional semantic representation for each target/pseudo word. In my code I provided an experiment to determine a good choice for this hyper-parameter. I simply allow an input as a list of values for this hyperparameter, and then run the above-described method to obtain clustering accuracy, 5 times for each hyper-parameter value. Then the context window size with the highest average accuracy is selected as the value to use. In my testing I usually got the result of context window sized 40, this is used as the default value if hyper-parameter selection isn't run in the code. As you can see in Figure 2 the context window sized 40 achieved the highest accuracy of the hyper-parameter options listed, so was selected in the running of the clustering. TA see footnote[*].

[*] For the TA marking they don't need to run this if not needed as it takes around 8-10 minutes to run. The default values used (40 for the context window) are from the results of my hyper-parameter selection.

Task 2: Neural Network for Classifying Product Reviews

Method Description

The text cleaning and pre-processing steps are as follows: first the data from the review files is read into an array of string containing all 9 of the documents, done in the `read_data` function. Then the `preprocess` function is called, this sends each document to a helper function `process_raw`, this function splits the review file into its separate lines and then removes all instances of the "`[t]`" tag, as this information isn't relevant for my implementation. Then it passes the non-empty lines to another helper function `process_line`, this looks for the delimiter "`##`" which separates the classification from the actual review data, if no delimiter is present, it returns empty values which means that line isn't considered. For the data after the delimiter the line is converted to lower case, any punctuation is removed (this provides no extra information that can be used by my implementation) and the `nltk` module's `word_tokenize` is used to remove any empty whitespace and the string is re-joined together with space between each token. Then `process_line` returns the joined `line_tokens` (review data) string and the `review_info` string (classifications). `process_raw` takes these strings and converts the `review_info` into a positive or negative classification, the classifications are determined by tags of `[+/-1/2/3]` or just `[+/-]`, my code weights the classifications based off the numbers (tags without numbers are given weight 1) in order to calculate an overall classification for the review. This deals with any reviews that contain positive and negative classifications for the same line. Any classifications that are equal are discarded and any that are positive are represented in `self.classification` list as a 1 and 0 for the negative ones, the `processed_review` data is also added to an list `self.reviews`.

My classifier as defined in the `build_train_model` contains 4 layers all provided by the `tensorflow keras` module. The first layer, `encoder`, is a `TextVectorization` layer which converts the string review data from a sequence of tokens (words) to a sequence of token indices from the `encoder` vocabulary which is then padded so they are all the same size by using masking, this can be passed to the next layer. The second layer is the `Embedding` layer, this takes the sequence of token indices and converts it to a sequence of vectors which are trainable, the embedding layer stores one vector per word. The third layer is the recurrent neural network, I used a `LSTM` in order to preserve information, I also chose to use a `Bidirectional` wrapper in order to propagate the input forwards and backwards through the RNN, so the information is preserved from both past and future for each token. This layer also has a `recurrent_dropout` parameter in order to avoid overfitting for the model. Finally, the last layer is a `Dense` layer to convert the single vector output of the RNN into a single logit as the classification output, this uses the RELU activation function. See Figure 3 for a graphic of the structure of the model.

The model is trained on 80% of the data per fold using N fold Cross Validation, generated by the `nfold_cv` function (default number of folds is 5, which gives 80% for training and 20% for testing for each fold). This data is randomly selected, and a proportional number of positive and negative reviews is provided. For example, if there were 100 positive and 50 negative reviews, the training data would contain a random sample of 80 positive reviews and 40 negative reviews for each of the 5 folds.

Experiment and Result Analysis

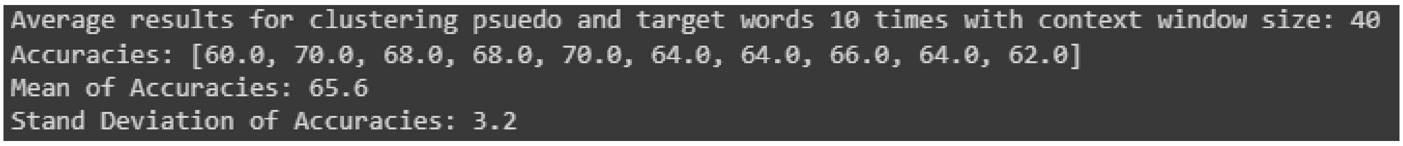
For my experiment I used 5-fold cross validation in order to train and test the model. For each, fold the classification accuracy of the test data is returned by the model, then after all the folds the average accuracy and the standard deviation is computed and shown. From Figure 4 you can see the model achieves an accuracy of 73.4% with a low standard deviation of 0.017, this seems to me to be a good result given the relatively small dataset, the low standard deviation shows how my model is consistent and my data splitting through the `nfold_cv` function is appropriate and random.

The model contains 2 hyper-parameters, the `recurrent_dropout` which drops nodes in the neural network to avoid overfitting, and the `BATCH_SIZE` which determines the number of samples that will be propagated through the network at one time. These hyper-parameters are decided by the `hyper_parameter_select` function, which used 5-fold CV for each option for the hyper-parameters in order to determine which option performs best. This is then used in the training and testing of my model in the experiment. The recurrent dropout parameter had no real change, as seen in Figure 5, so I left it as 0.2 in my experiment. The batch size performed best on smaller batches, and batch size 16 was the best, as seen in Figure 6, so this is used in my experiment. TA see footnote[†].

[†] For the TA marking they don't need to run this if not needed as it takes around 25 minutes to run. The default values used (0.2 for recurrent dropout and 16 for batch size) are from the results of my hyper-parameter selection.

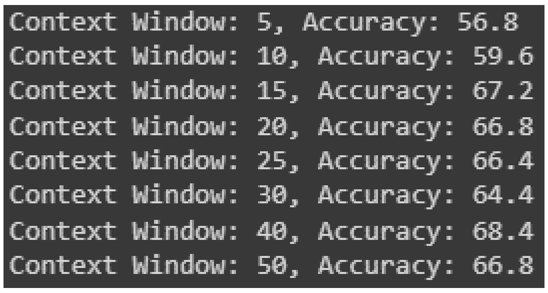
Appendix

Figure 1 – Results of Clustering Accuracy



```
Average results for clustering psuedo and target words 10 times with context window size: 40
Accuracies: [60.0, 70.0, 68.0, 68.0, 70.0, 64.0, 64.0, 66.0, 64.0, 62.0]
Mean of Accuracies: 65.6
Stand Deviation of Accuracies: 3.2
```

Figure 2 – Results of Hyper-parameter Selection for the Context Window



```
Context Window: 5, Accuracy: 56.8
Context Window: 10, Accuracy: 59.6
Context Window: 15, Accuracy: 67.2
Context Window: 20, Accuracy: 66.8
Context Window: 25, Accuracy: 66.4
Context Window: 30, Accuracy: 64.4
Context Window: 40, Accuracy: 68.4
Context Window: 50, Accuracy: 66.8
```

Figure 3 – Neural Network Model Structure

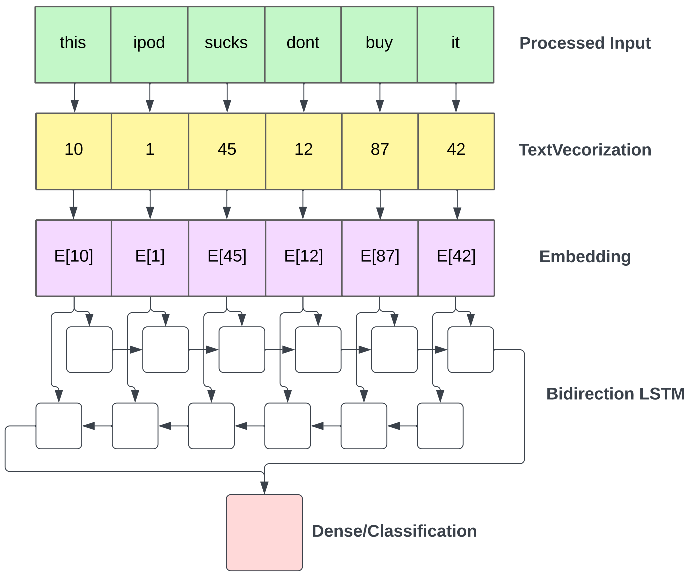
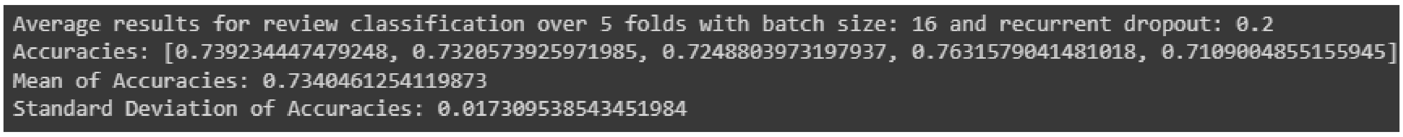
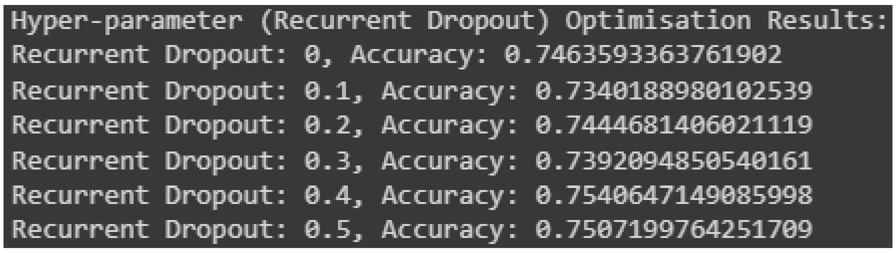


Figure 4 – Results of Neural Network for Classifying Product Reviews Experiment



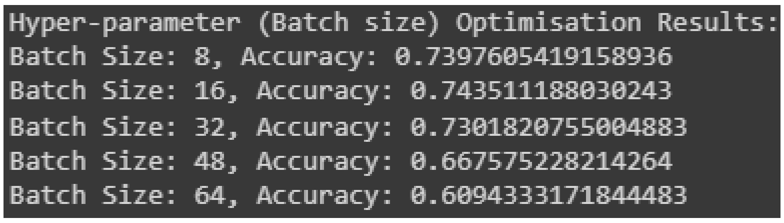
```
Average results for review classification over 5 folds with batch size: 16 and recurrent dropout: 0.2
Accuracies: [0.739234447479248, 0.7320573925971985, 0.7248803973197937, 0.7631579041481018, 0.7109004855155945]
Mean of Accuracies: 0.7340461254119873
Standard Deviation of Accuracies: 0.017309538543451984
```

Figure 5 - Results of Hyper-parameter Selection for the Recurrent Dropout



```
Hyper-parameter (Recurrent Dropout) Optimisation Results:
Recurrent Dropout: 0, Accuracy: 0.7463593363761902
Recurrent Dropout: 0.1, Accuracy: 0.7340188980102539
Recurrent Dropout: 0.2, Accuracy: 0.7444681406021119
Recurrent Dropout: 0.3, Accuracy: 0.7392094850540161
Recurrent Dropout: 0.4, Accuracy: 0.7540647149085998
Recurrent Dropout: 0.5, Accuracy: 0.7507199764251709
```

Figure 6 - Results of Hyper-parameter Selection for the Batch Size



```
Hyper-parameter (Batch size) Optimisation Results:
Batch Size: 8, Accuracy: 0.7397605419158936
Batch Size: 16, Accuracy: 0.743511188030243
Batch Size: 32, Accuracy: 0.7301820755004883
Batch Size: 48, Accuracy: 0.667575228214264
Batch Size: 64, Accuracy: 0.6094333171844483
```