Boonrit Boonmarueng 61070701611

Final Examination 2/2019
CPE 603 Special Topic III Text Mining and Social Network Analytics

1. Explain the rational of the deep learning design of the work proposed in the paper in detail.

The goal of this study is to perform the sentiment classification on the Internet Movie Database (IMDb) review sentiment dataset. The author used the combination of multiple kernels from multiple branches, that each branch has the Convolutional Neural Networks (CNNs) layers together with Long Short-Term Memory (LSTM) layers. Since using all these techniques, the model accuracy reports as high as 89.5%.

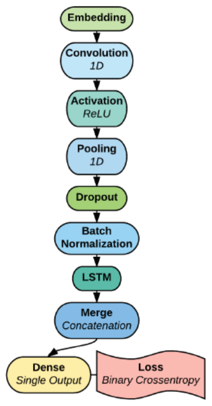


Figure 1 The structure of 1 branch.

Figure 1 shows an overview of the structure of a branch, which is a minor part of their designed system. A branch's structure can be distinguished into 7 main layers as follows: 1). Embedding Layer which is the matrix of trainable weights that produce the vectors for each word. 2). CNN Layer that slides 1-dimensional kernel size c over the inputs each branch. The main purpose of the CNN layer is to view word combinations of the kernel size c. 3). In the Activation layer, they always applied ReLu activation functions to each branch to convert negative outputs to zero value. In fact, this layer is used in order to create the nonlinearity into their network system. 4. The max pooling layer used for reducing data size. 5). The dropout layer, this layer randomly sets a portion of the inputs to zero value in order to prevent the overfitting issue. 6). The Batch Normalization, it can relieve the problem of internal covariate shift, where parameter initialization and changes in the distribution of the inputs of each layer affect the learning rate of the network. Finally, 7). the LSTM layers, that is the final layer for each branch. The LSTM is used to learn the relationship of the sequential data.

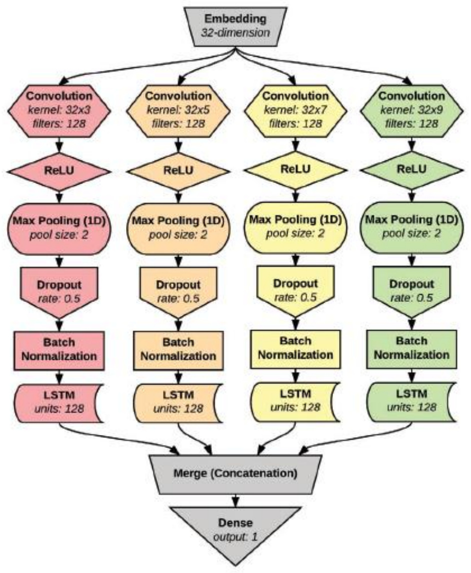


Figure 2 The best performing proposed model (Model_63).

As mentioned in the previous paragraph, Figure 1 is just a structure of only a branch. In this paper, many branches were combined. The best performing proposed model naming model_63 is shown in Figure 2. This structure consists of 4 branches and gives the reported accuracy is 89.5%. This model is a model in more than 60 models that the author attempted to tune parameters and hyperparameters to obtain the best accuracy, and avoid the over-fitting problems at the same time.

From figure 2, the results of embedding layers are fed to the inputs of each branch. In their experiment, the author tried to do the word2vec both with GoogleNews word2vec and Wikipedia fastText pre-trained embeddings. However, the results show that there is no effect in using this technique. Therefore, the author does not include these techniques in the best model. Since the 4 branches are used because the author found that there were 4 appropriate kernel sizes that can return the highest accuracy including three, five, seven, and nine. Moreover, the optimal number of filters in CNN layers was found to be 128, and also the ReLU activation provided higher accuracy than other activation functions.

In the max-pooling layer, they experimented and reported that the optimum kernel size of the max-pooling layer was 2 (reducing the input by half). After that, the best ratio of dropout was 50%.

Each branch's LSTM layer has 128 units. The author mentioned that any less or more units are affected to reduce accuracy and increase overfitting of the model. They experimented many times and explained that the optimizer did not have a profound effect. Finally, the RMSprop optimizer indicated the best results.

At the end of the LSTM layer, combine the results from all 4 branches and feed them into the Fully Connected Layer to create a predicted result. In the model_63, the optimum batch size is 128.

One of the major problems with deep learning is overfitting. Therefore, to seriously examine and guarantee this issue, this paper collected and plotted 4 graphs consisting of 1).training loss, 2).trainig accuracy, 3).validation loss and 4).validation accuracy at all times to confirm that no overfitting issues. To preventing the overfitting problem, this paper used a lot of layers, Max Pooling, Dropout, and Normalization techniques. Additionally, they performed the principle of regularization in their experiment, which is the essential technique for reducing overfitting by using the optimal kernel regularizer was L2 (0.01).

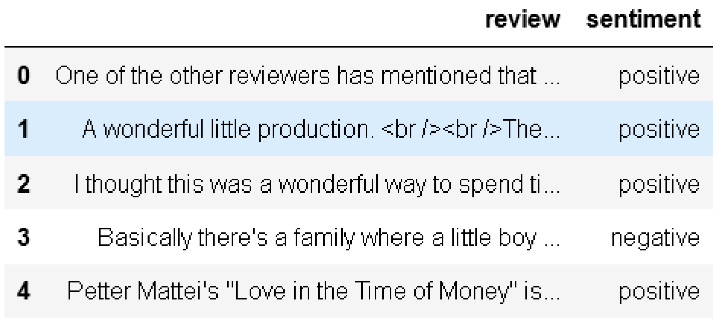
In summary, this research performed several experiments to adjust the parameters and hyperparameters to achieve the model that provided the highest accuracy, while also preventing overfitting at the same time. Finally, the proposed model with the best efficiency at 89.5% accuracy is model_63, which gathers the results from 4 branches(CNNs+LSTM) that using 128 kernels/branch with kernel size 3,5,7 and 9 respectively, and feeds the mixed result to the fully connected layer to classify the sentiment.

2. In this question, you have to build a deep-learning sentiment analysis model to predict the IMDB Dataset of 50K Movie Reviews (downloadable from the link below). Design your own deep-learning architecture and explain your reason for the design in detail. Provide proper performance measurements.

The procedures for analyzing and creating this sentiment analysis model that used to predict the IMDB Dataset of 50K Movie Reviews can be distinguished into 6 parts as follows 1). Data Exploration, 2). Data Preparation, 3). Performance Measurement, 4). The Deep Learning Architecture, and 5). Result.

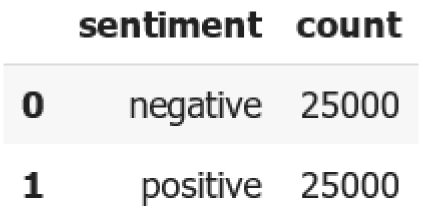
1). Data Exploration.

From figure 3, this data set consists of 2 columns consisting of 1. review which is the type of information written by the reviewer, and 2. sentiment indicating the class that previously rebelled. Subsequently, considering the distribution of the class in the sentiment column, figure 4 showed that there are 2 classes: negative and positive with each of the five thousand rows.



	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. <br /><br />The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

Figure 3 The sample of the dataset.

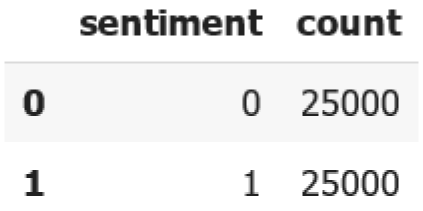


	sentiment	count
0	negative	25000
1	positive	25000

Figure 4 The proportion of classes in the Sentiment column.

2). Data Preparation

Since a deep learning model only requires inputs and outputs in the form of numeric datatype. Initially, I converted the sentiment column from category form to binary form. The result is shown in picture 5. After that, I wrote a program to remove the words, which are a type of non-alphanumeric characters, stop words, pronouns, and also spaces. Moreover, converting all words to lowercase. From the examination, I found that the maximum written text is 2470 words and 180,164 unique words. To be able to enable word-type data to be processed, I created the sequences by performing the tokenizer, padding. and setting the max length for each word at 300 words. I used 300 words only because the paper in question 1, the author suggested that only 200-300 words are sufficient.



	sentiment	count
0	0	25000
1	1	25000

Figure 5 The proportion of classes in the Sentiment column after converting to binary form.

To avoid bias from random splitting data, we always set the random state at 38 with a 50% training set and a 50% testing set.

3). Performance Measurement

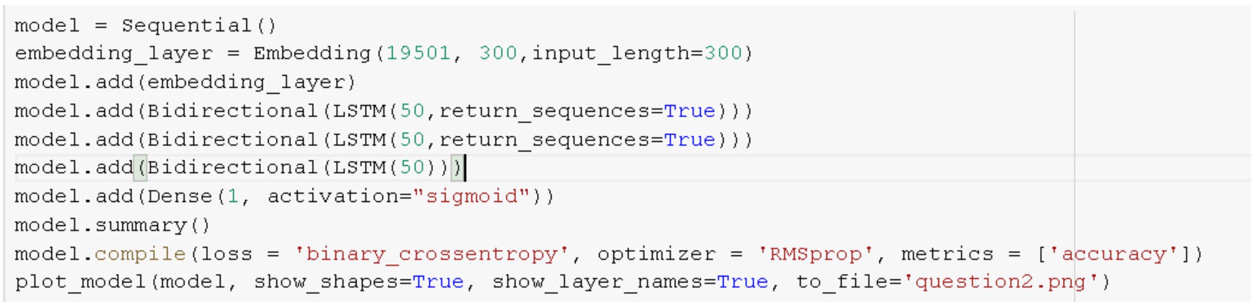
I mainly focused on the text classification based on text content with supervised learning. To evaluate the effectiveness of each model, the same training data were used to train the model, and the same testing data were used every time to reduce the bias that can occur. The performance metric used to measure the model's performance in this paper is the accuracy, which is the percentage that the predicted class matches the actual class.

4). The deep-learning architecture

In this experiment, I selected the Bidirectional Recurrent Neural Networks (BRNNs). BRNN is one type of recurrent neural network (RNN). In the BRNN's structure, it connects adjacent layers in two directions, consists of feeding forward and backward to create relationship patterns or word patterns in both directions. I conducted 3 consecutive layers of Long Short-Term Memory (LSTMs) in our architecture because the LSTM outperforms in tasks requiring modeling long-distance relations. There are 2 types of BRNN that I designed in this experiment including BRNN without l2 and with l2 regularization.

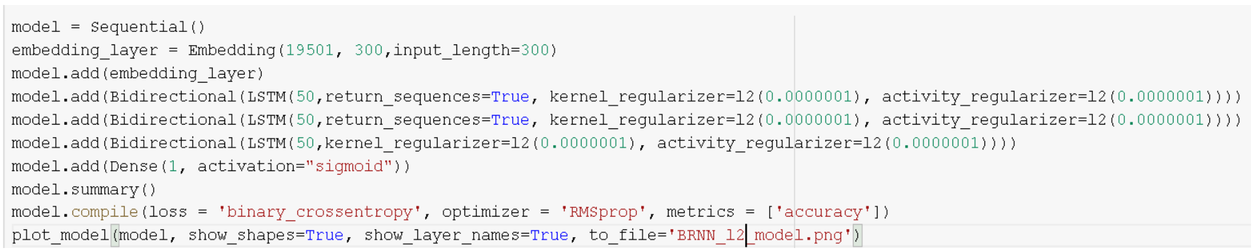
From figure 5, BRNN without L2. in the embedding later, input_dim, output_dim, and input_length was set at 19,500, 300, and 300 respectively. 3 layers of LSTM were connected with the number of nodes, each layer is 50. Finally, the result was set to 1 node using the Sigmoid activation function. From figure 6, BRNN with l2, all parameters are the same as figure 5, but adding l2 for each LSTM layer.

The loss function in this experiment used the binary cross-entropy with RMSprop optimizer. I chose this type of optimizer because in the paper in question 1, they showed that RMSprop optimizer provides the highest accuracy.



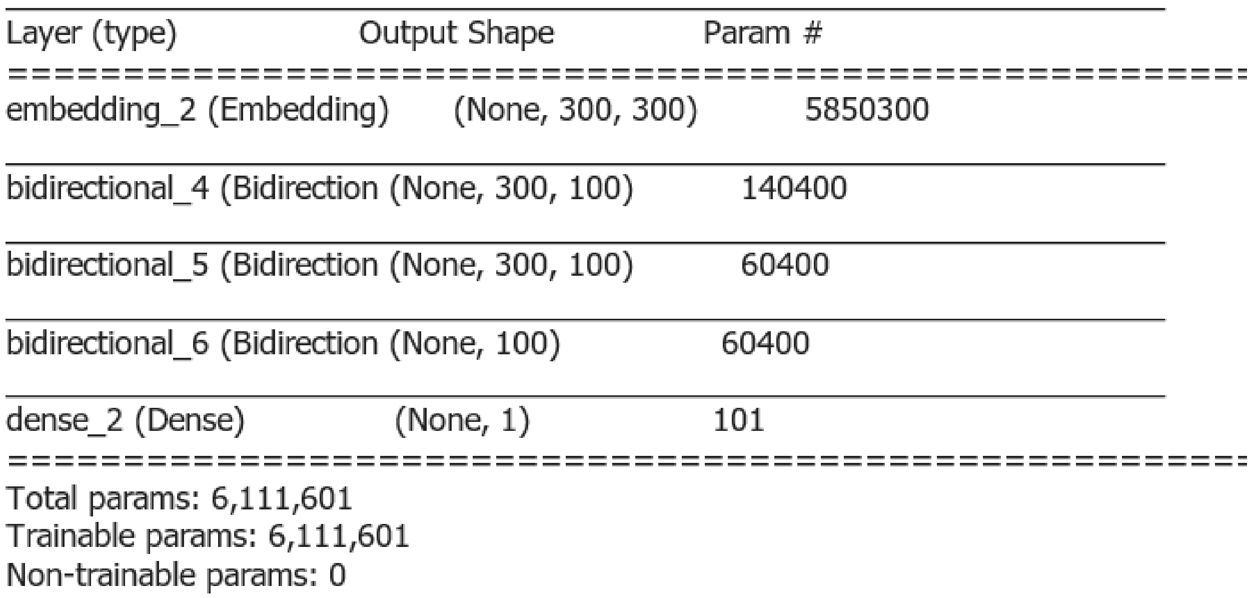
```
model = Sequential()
embedding_layer = Embedding(19501, 300,input_length=300)
model.add(embedding_layer)
model.add(Bidirectional(LSTM(50,return_sequences=True)))
model.add(Bidirectional(LSTM(50,return_sequences=True)))
model.add(Bidirectional(LSTM(50)))
model.add(Dense(1, activation="sigmoid"))
model.summary()
model.compile(loss = 'binary_crossentropy', optimizer = 'RMSprop', metrics = ['accuracy'])
plot_model(model, show_shapes=True, show_layer_names=True, to_file='question2.png')
```

Figure 5 The code for defining the structure of the BRNN model.



```
model = Sequential()
embedding_layer = Embedding(19501, 300,input_length=300)
model.add(embedding_layer)
model.add(Bidirectional(LSTM(50,return_sequences=True, kernel_regularizer=l2(0.0000001), activity_regularizer=l2(0.0000001))))
model.add(Bidirectional(LSTM(50,return_sequences=True, kernel_regularizer=l2(0.0000001), activity_regularizer=l2(0.0000001))))
model.add(Bidirectional(LSTM(50,kernel_regularizer=l2(0.0000001), activity_regularizer=l2(0.0000001))))
model.add(Dense(1, activation="sigmoid"))
model.summary()
model.compile(loss = 'binary_crossentropy', optimizer = 'RMSprop', metrics = ['accuracy'])
plot_model(model, show_shapes=True, show_layer_names=True, to_file='BRNN_l2_model.png')
```

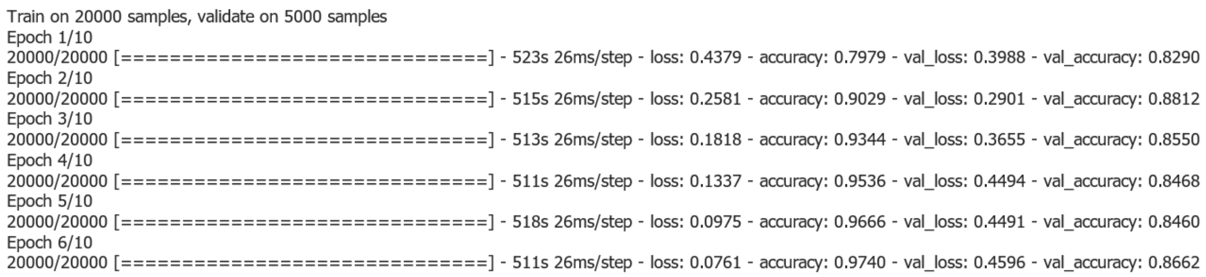
Figure 6 The code for defining the structure of the BRNN+l2 model.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 300, 300)          5850300
_________________________________________________________________
bidirectional_4 (Bidirection (None, 300, 100)          140400
_________________________________________________________________
bidirectional_5 (Bidirection (None, 300, 100)          60400
_________________________________________________________________
bidirectional_6 (Bidirection (None, 100)               60400
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 101
=================================================================
Total params: 6,111,601
Trainable params: 6,111,601
Non-trainable params: 0
```

Figure 7 The structure and number of parameters used in the BRNN and BRNN+l2 model.

Since the number of parameters used in this experiment is over 6 million, if using many batchSize, the memory used will not be enough. Therefore, it is only necessary to set a small batchSize of 100. However, the disadvantage is that the training duration will be longer (approximately 8.5 minutes/epoch as shown in figure 8). However, to avoid unnecessary training I set up the early stop condition with patience set = 4.



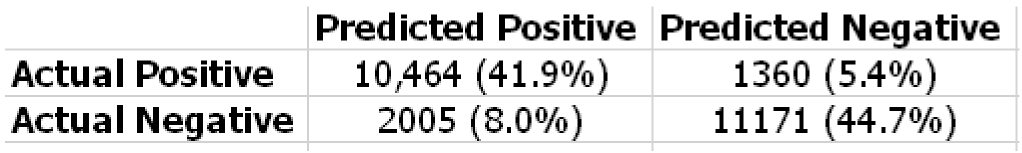
```
Train on 20000 samples, validate on 5000 samples
Epoch 1/10
20000/20000 [==============================] - 523s 26ms/step - loss: 0.4379 - accuracy: 0.7979 - val_loss: 0.3988 - val_accuracy: 0.8290
Epoch 2/10
20000/20000 [==============================] - 515s 26ms/step - loss: 0.2581 - accuracy: 0.9029 - val_loss: 0.2901 - val_accuracy: 0.8812
Epoch 3/10
20000/20000 [==============================] - 513s 26ms/step - loss: 0.1818 - accuracy: 0.9344 - val_loss: 0.3655 - val_accuracy: 0.8550
Epoch 4/10
20000/20000 [==============================] - 511s 26ms/step - loss: 0.1337 - accuracy: 0.9536 - val_loss: 0.4494 - val_accuracy: 0.8468
Epoch 5/10
20000/20000 [==============================] - 518s 26ms/step - loss: 0.0975 - accuracy: 0.9666 - val_loss: 0.4491 - val_accuracy: 0.8460
Epoch 6/10
20000/20000 [==============================] - 511s 26ms/step - loss: 0.0761 - accuracy: 0.9740 - val_loss: 0.4596 - val_accuracy: 0.8662
```

Figure 8 Training in the BRNN model.

5). Result

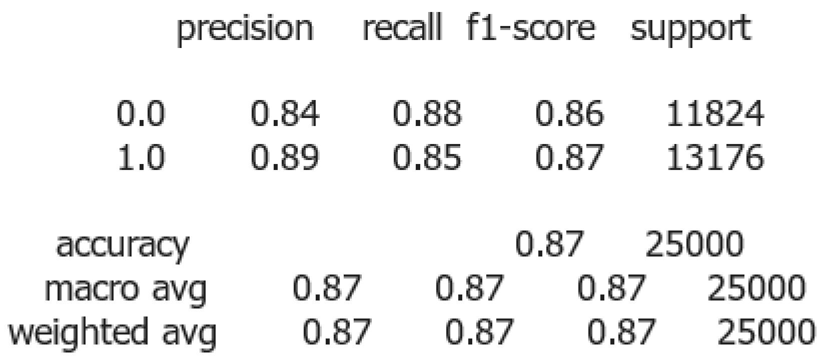
To test the effectiveness of the model, I tested the BRNN model by using testing datasets whose size is 25,000 rows. The confusion matrix is shown in Figure 9. From the results, it reported that the error percentage is around 13.5%. Therefore, the accuracy of the BRNN model is approximately 86.6%. While the accuracy of BRNN with l2 is 85%.

However, overfitting is the major issue in this experiment because the training accuracy is higher than both validation and testing accuracy.



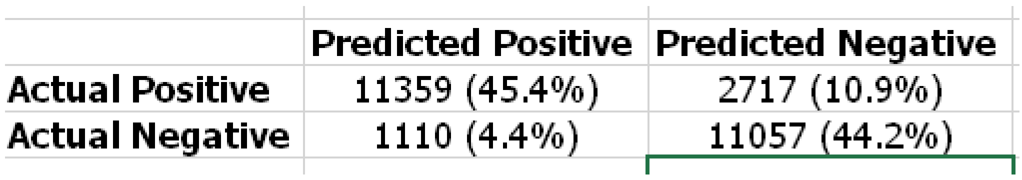
	Predicted Positive	Predicted Negative
Actual Positive	10,464 (41.9%)	1360 (5.4%)
Actual Negative	2005 (8.0%)	11171 (44.7%)

Figure 9 The confusion matrix of the BRNN model.



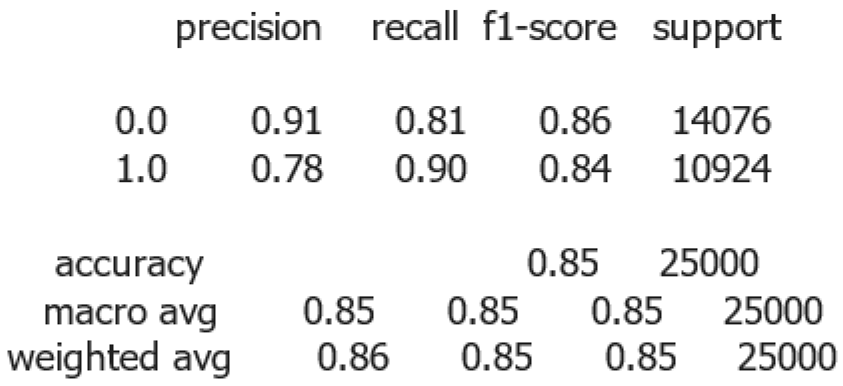
```
              precision    recall  f1-score   support

         0.0       0.84      0.88      0.86     11824
         1.0       0.89      0.85      0.87     13176

    accuracy                           0.87     25000
   macro avg       0.87      0.87      0.87     25000
weighted avg       0.87      0.87      0.87     25000
```

Figure 10 The classification report from the BRNN model.



	Predicted Positive	Predicted Negative
Actual Positive	11359 (45.4%)	2717 (10.9%)
Actual Negative	1110 (4.4%)	11057 (44.2%)

Figure 11 The confusion matrix of the BRNN+L2 model.



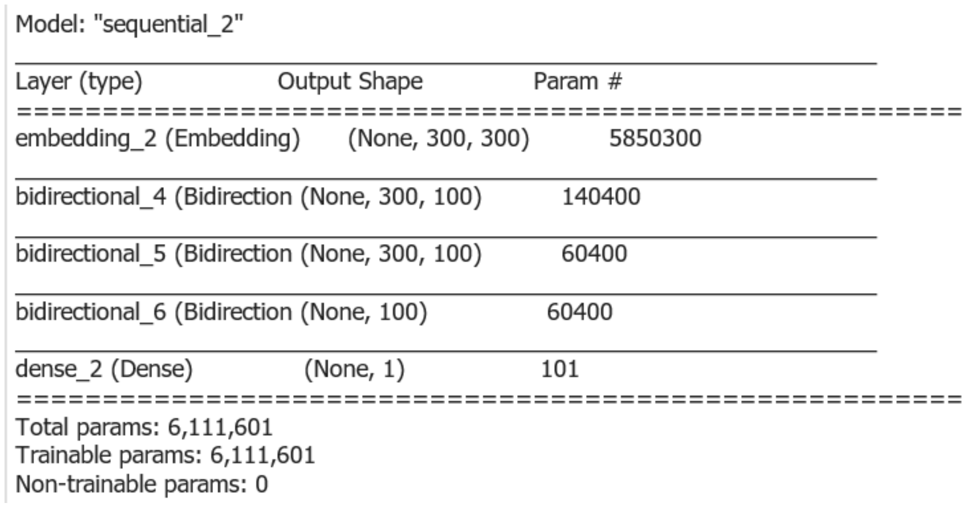
```
              precision    recall  f1-score   support

         0.0       0.91      0.81      0.86     14076
         1.0       0.78      0.90      0.84     10924

    accuracy                           0.85     25000
   macro avg       0.85      0.85      0.85     25000
weighted avg       0.86      0.85      0.85     25000
```

Figure 12 The classification report from the BRNN+L2 model.

3. Use the same dataset as in question 2, test to see the difference in model performance between the following text representation strategies. Note that everything else (parameters/model design) has to remain the same.
   a. **One-hot**
      i. One-hot without the l2 regularization

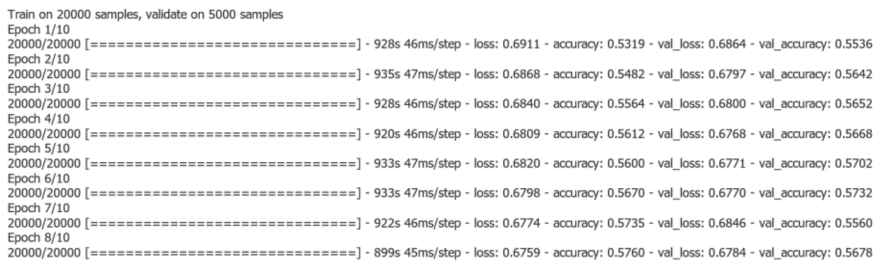
      The data preparation is still similar to the answer in question 2. However, as mentioned in the answer from question 2, the number of unique words is 180,164 words. Therefore, if using these words to create one-hot, it will not be able to train the model because of memory error. To solve this problem initially, I created a tf-idf matrix and selected the first 300 most important words, then used these words to create a One-hot matrix by using the one hot tokenizer from Keras. The shape of the one-hot result is (50,000, 300).

      The training/test data proportion, parameters, and optimizer are set to remain the same as the answer in question 2. From figure 13, the total parameters are fixed at 6,111,601 the same as figure 6.



```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 300, 300)          5850300
_________________________________________________________________
bidirectional_4 (Bidirection (None, 300, 100)          140400
_________________________________________________________________
bidirectional_5 (Bidirection (None, 300, 100)          60400
_________________________________________________________________
bidirectional_6 (Bidirection (None, 100)               60400
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 101
=================================================================
Total params: 6,111,601
Trainable params: 6,111,601
Non-trainable params: 0
```

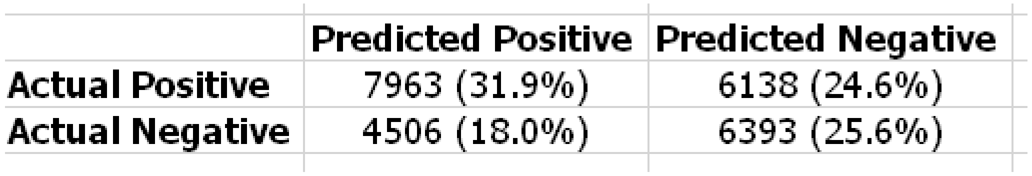
Figure 13 The structure and number of parameters used in the one-hot+BRNN model



```
Train on 20000 samples, validate on 5000 samples
Epoch 1/10
20000/20000 [==============================] - 928s 46ms/step - loss: 0.6911 - accuracy: 0.5319 - val_loss: 0.6864 - val_accuracy: 0.5536
Epoch 2/10
20000/20000 [==============================] - 935s 47ms/step - loss: 0.6868 - accuracy: 0.5482 - val_loss: 0.6797 - val_accuracy: 0.5642
Epoch 3/10
20000/20000 [==============================] - 928s 46ms/step - loss: 0.6840 - accuracy: 0.5564 - val_loss: 0.6800 - val_accuracy: 0.5652
Epoch 4/10
20000/20000 [==============================] - 920s 46ms/step - loss: 0.6809 - accuracy: 0.5612 - val_loss: 0.6768 - val_accuracy: 0.5668
Epoch 5/10
20000/20000 [==============================] - 933s 47ms/step - loss: 0.6820 - accuracy: 0.5600 - val_loss: 0.6771 - val_accuracy: 0.5702
Epoch 6/10
20000/20000 [==============================] - 933s 47ms/step - loss: 0.6798 - accuracy: 0.5670 - val_loss: 0.6770 - val_accuracy: 0.5732
Epoch 7/10
20000/20000 [==============================] - 922s 46ms/step - loss: 0.6774 - accuracy: 0.5735 - val_loss: 0.6846 - val_accuracy: 0.5560
Epoch 8/10
20000/20000 [==============================] - 899s 45ms/step - loss: 0.6759 - accuracy: 0.5760 - val_loss: 0.6784 - val_accuracy: 0.5678
```

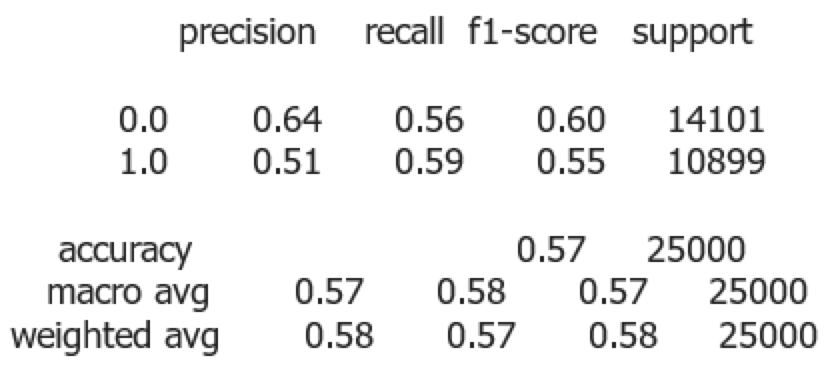
Figure 14 Training in the one hot+BRNN model.

The performance of the one hot + BRNN model shows that the error percentage is 42.6%. Therefore, the accuracy rate is 57% as shown in the classification report from figure 16.



	Predicted Positive	Predicted Negative
Actual Positive	7963 (31.9%)	6138 (24.6%)
Actual Negative	4506 (18.0%)	6393 (25.6%)

Figure 15 The confusion matrix of the one-hot+BRNN model.

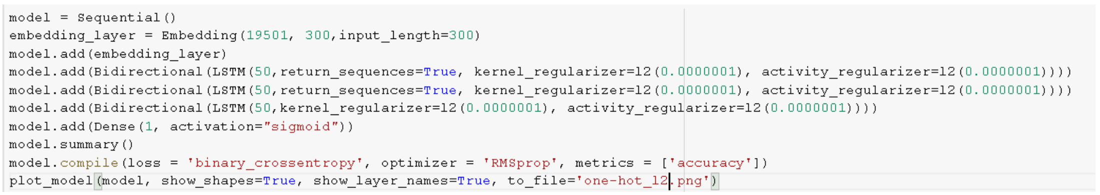


```
              precision    recall  f1-score   support

         0.0       0.64      0.56      0.60     14101
         1.0       0.51      0.59      0.55     10899

    accuracy                           0.57     25000
   macro avg       0.57      0.58      0.57     25000
weighted avg       0.58      0.57      0.58     25000
```

Figure 16 The classification report from the one-hot+BRNN model.

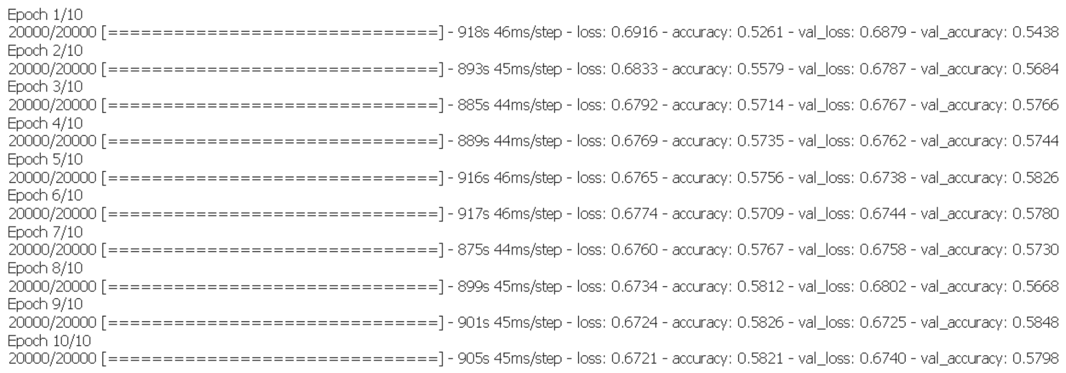
ii. One-hot with the l2 regularization

In this experiment I included the additional parameters by adding L2 to each LSTM layer (kernel_regularizer = l2 (0.0000001) and activity_regularizer = l2 (0.0000001)) as shown in figure 17. However, the structure and parameters remain the same as the previous models. After testing the model, the results revealed that the accuracy of the one-hot BRNN+l2 is 58% as shown in figure 20.



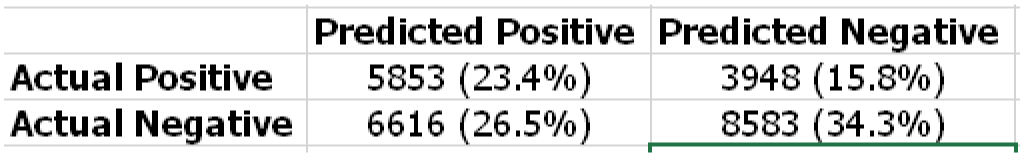
```
model = Sequential()
embedding_layer = Embedding(19501, 300,input_length=300)
model.add(embedding_layer)
model.add(Bidirectional(LSTM(50,return_sequences=True, kernel_regularizer=l2(0.0000001), activity_regularizer=l2(0.0000001))))
model.add(Bidirectional(LSTM(50,return_sequences=True, kernel_regularizer=l2(0.0000001), activity_regularizer=l2(0.0000001))))
model.add(Bidirectional(LSTM(50,kernel_regularizer=l2(0.0000001), activity_regularizer=l2(0.0000001))))
model.add(Dense(1, activation="sigmoid"))
model.summary()
model.compile(loss = 'binary_crossentropy', optimizer = 'RMSprop', metrics = ['accuracy'])
plot_model(model, show_shapes=True, show_layer_names=True, to_file='one-hot_l2.png')
```

Figure 17 The structure and number of parameters used in the one-hot+L2+BRNN model



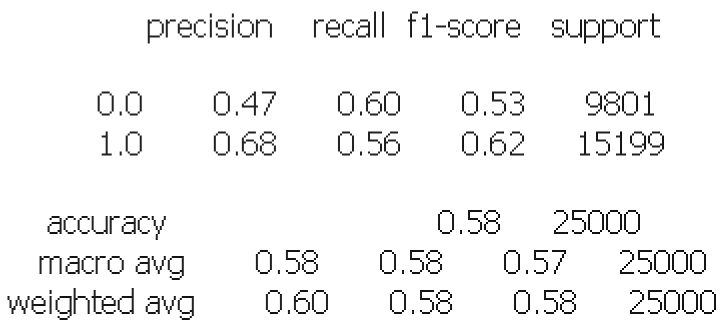
```
Epoch 1/10
20000/20000 [==============================] - 918s 46ms/step - loss: 0.6916 - accuracy: 0.5261 - val_loss: 0.6879 - val_accuracy: 0.5438
Epoch 2/10
20000/20000 [==============================] - 893s 45ms/step - loss: 0.6833 - accuracy: 0.5579 - val_loss: 0.6787 - val_accuracy: 0.5684
Epoch 3/10
20000/20000 [==============================] - 885s 44ms/step - loss: 0.6792 - accuracy: 0.5714 - val_loss: 0.6767 - val_accuracy: 0.5766
Epoch 4/10
20000/20000 [==============================] - 889s 44ms/step - loss: 0.6769 - accuracy: 0.5735 - val_loss: 0.6762 - val_accuracy: 0.5744
Epoch 5/10
20000/20000 [==============================] - 916s 46ms/step - loss: 0.6765 - accuracy: 0.5756 - val_loss: 0.6738 - val_accuracy: 0.5826
Epoch 6/10
20000/20000 [==============================] - 917s 46ms/step - loss: 0.6774 - accuracy: 0.5709 - val_loss: 0.6744 - val_accuracy: 0.5780
Epoch 7/10
20000/20000 [==============================] - 875s 44ms/step - loss: 0.6760 - accuracy: 0.5767 - val_loss: 0.6758 - val_accuracy: 0.5730
Epoch 8/10
20000/20000 [==============================] - 899s 45ms/step - loss: 0.6734 - accuracy: 0.5812 - val_loss: 0.6802 - val_accuracy: 0.5668
Epoch 9/10
20000/20000 [==============================] - 901s 45ms/step - loss: 0.6724 - accuracy: 0.5826 - val_loss: 0.6725 - val_accuracy: 0.5848
Epoch 10/10
20000/20000 [==============================] - 905s 45ms/step - loss: 0.6721 - accuracy: 0.5821 - val_loss: 0.6740 - val_accuracy: 0.5798
```

Figure 18 Training in the one-hot+L2+BRNN model.



	Predicted Positive	Predicted Negative
Actual Positive	5853 (23.4%)	3948 (15.8%)
Actual Negative	6616 (26.5%)	8583 (34.3%)

Figure 19 The confusion matrix of the one-hot+L2+BRNN model.



```
              precision    recall  f1-score   support

         0.0       0.47      0.60      0.53      9801
         1.0       0.68      0.56      0.62     15199

    accuracy                           0.58     25000
   macro avg       0.58      0.58      0.57     25000
weighted avg       0.60      0.58      0.58     25000
```

Figure 20 The classification report from the one-hot+L2+BRNN model.

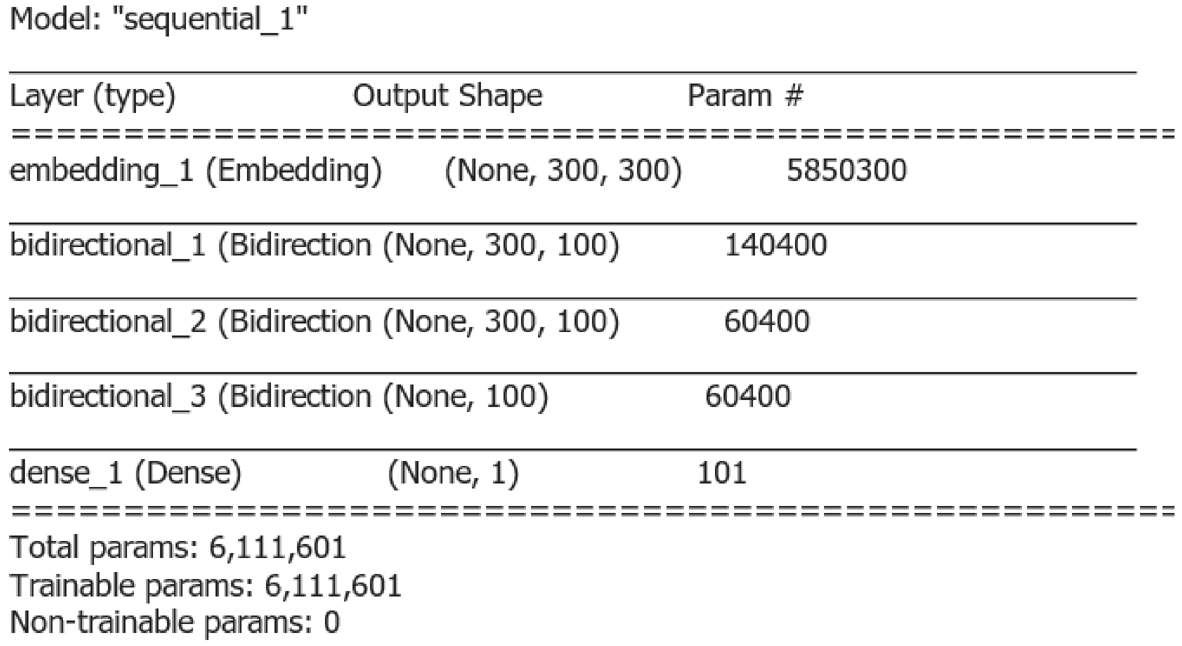
b. Word x Word

Before training the model, I created a wordxword or co-occurrence matrix by using the CountVectorizer from the sklearn library. I used the filtered data same as a one-hot model, Then, titting the data and transforming with the count vectorizer by assigning a max_feature to 300 words. Finally, the end result is a size matrix of 50000x300.

Various parameters, proportion of training/testing and optimizer are set the same as the previous structure.

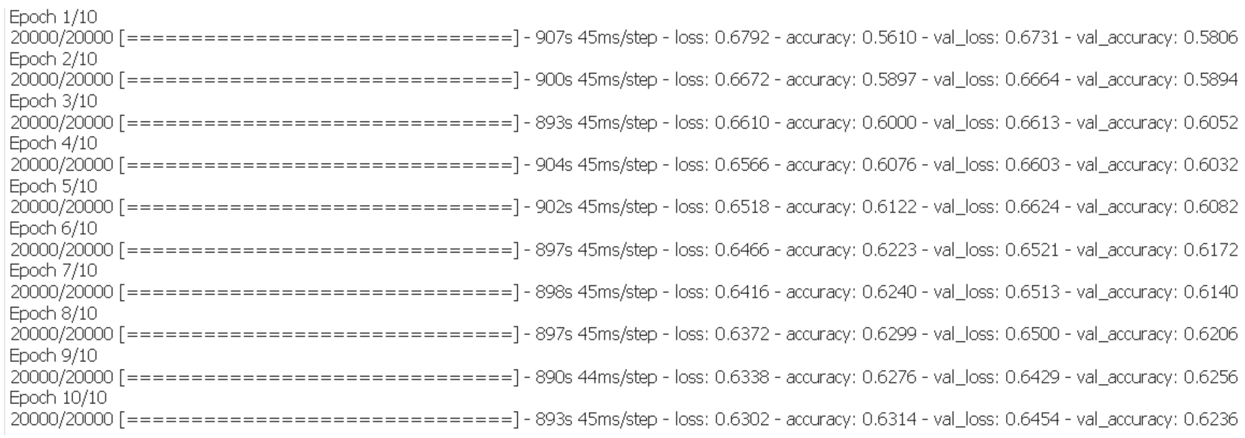
i. Word x Word without the l2 regularization

From picture 24, the efficiency of this model is more accurate than the previous 2 models (one-hot&one-hot+l2 BRNN model) with 62% accuracy.



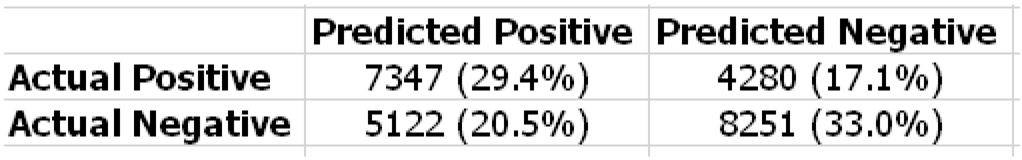
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 300, 300)          5850300
_________________________________________________________________
bidirectional_1 (Bidirection (None, 300, 100)          140400
_________________________________________________________________
bidirectional_2 (Bidirection (None, 300, 100)          60400
_________________________________________________________________
bidirectional_3 (Bidirection (None, 100)               60400
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 101
=================================================================
Total params: 6,111,601
Trainable params: 6,111,601
Non-trainable params: 0
```

Figure 21 The structure and number of parameters used in the WordxWord+BRNN model



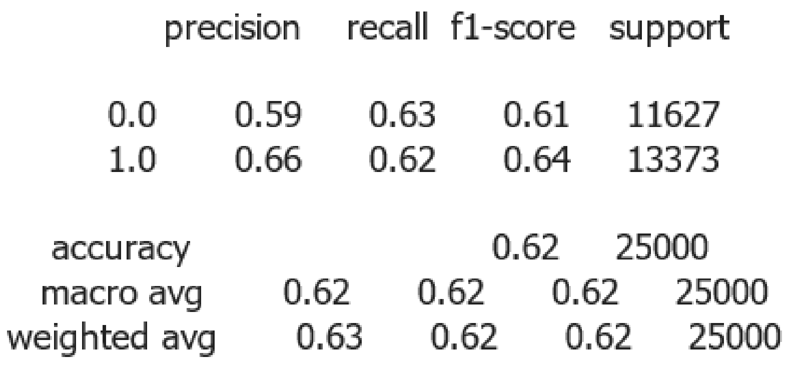
```
Epoch 1/10
20000/20000 [==============================] - 907s 45ms/step - loss: 0.6792 - accuracy: 0.5610 - val_loss: 0.6731 - val_accuracy: 0.5806
Epoch 2/10
20000/20000 [==============================] - 900s 45ms/step - loss: 0.6672 - accuracy: 0.5897 - val_loss: 0.6664 - val_accuracy: 0.5894
Epoch 3/10
20000/20000 [==============================] - 893s 45ms/step - loss: 0.6610 - accuracy: 0.6000 - val_loss: 0.6613 - val_accuracy: 0.6052
Epoch 4/10
20000/20000 [==============================] - 904s 45ms/step - loss: 0.6566 - accuracy: 0.6076 - val_loss: 0.6603 - val_accuracy: 0.6032
Epoch 5/10
20000/20000 [==============================] - 902s 45ms/step - loss: 0.6518 - accuracy: 0.6122 - val_loss: 0.6624 - val_accuracy: 0.6082
Epoch 6/10
20000/20000 [==============================] - 897s 45ms/step - loss: 0.6466 - accuracy: 0.6223 - val_loss: 0.6521 - val_accuracy: 0.6172
Epoch 7/10
20000/20000 [==============================] - 898s 45ms/step - loss: 0.6416 - accuracy: 0.6240 - val_loss: 0.6513 - val_accuracy: 0.6140
Epoch 8/10
20000/20000 [==============================] - 897s 45ms/step - loss: 0.6372 - accuracy: 0.6299 - val_loss: 0.6500 - val_accuracy: 0.6206
Epoch 9/10
20000/20000 [==============================] - 890s 44ms/step - loss: 0.6338 - accuracy: 0.6276 - val_loss: 0.6429 - val_accuracy: 0.6256
Epoch 10/10
20000/20000 [==============================] - 893s 45ms/step - loss: 0.6302 - accuracy: 0.6314 - val_loss: 0.6454 - val_accuracy: 0.6236
```

Figure 22 Training in the WordxWord+BRNN model.



	Predicted Positive	Predicted Negative
Actual Positive	7347 (29.4%)	4280 (17.1%)
Actual Negative	5122 (20.5%)	8251 (33.0%)

Figure 23 The confusion matrix of the WordxWord+BRNN model.

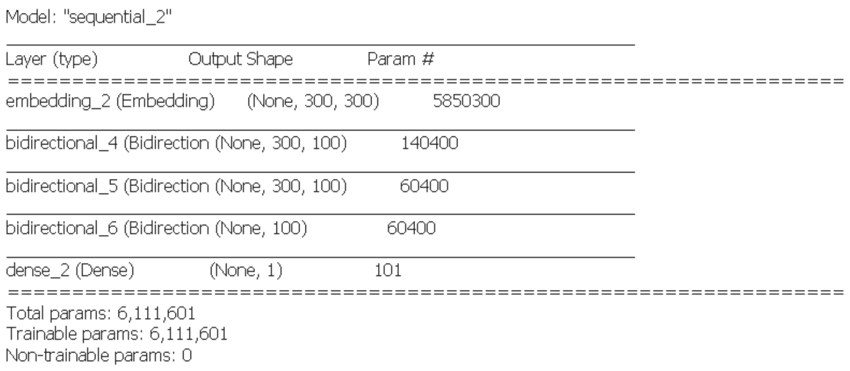


```
              precision    recall  f1-score   support

         0.0       0.59      0.63      0.61     11627
         1.0       0.66      0.62      0.64     13373

    accuracy                           0.62     25000
   macro avg       0.62      0.62      0.62     25000
weighted avg       0.63      0.62      0.62     25000
```

Figure 24 The classification report from the WordxWord+BRNN model.

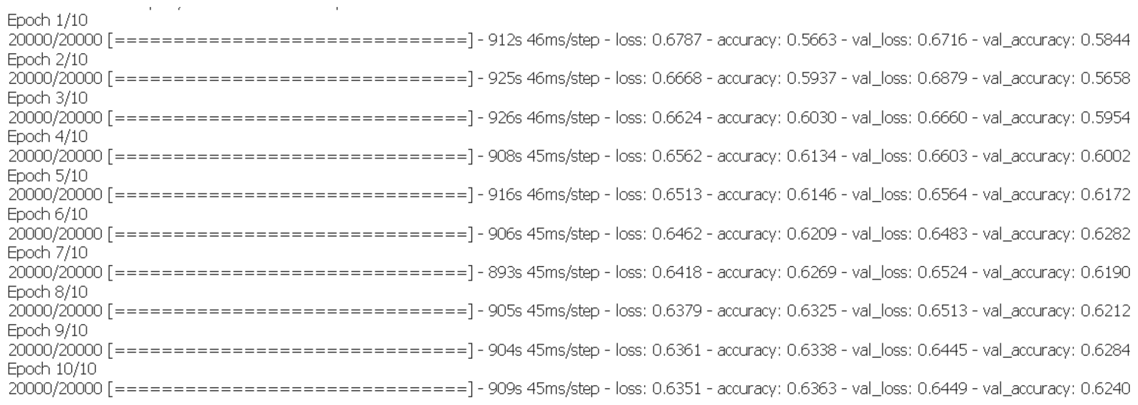
ii. Word x Word with the l2 regularization

In this experiment I included the additional parameters by adding L2 to each LSTM layer (kernel_regularizer = l2 (0.0000001) and activity_regularizer = l2 (0.0000001)). After testing the model, the results revealed that the accuracy of this model is 63% as shown in figure 28.



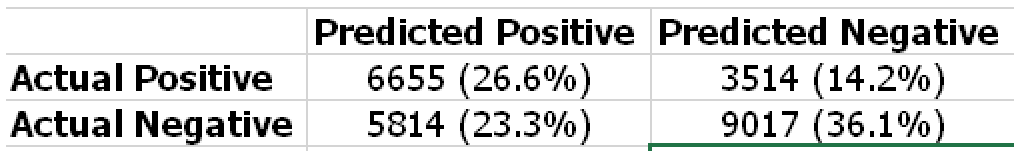
```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 300, 300)          5850300
_________________________________________________________________
bidirectional_4 (Bidirection (None, 300, 100)          140400
_________________________________________________________________
bidirectional_5 (Bidirection (None, 300, 100)          60400
_________________________________________________________________
bidirectional_6 (Bidirection (None, 100)               60400
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 101
=================================================================
Total params: 6,111,601
Trainable params: 6,111,601
Non-trainable params: 0
```

Figure 25 The structure and number of parameters used in the WordxWord+L2+BRNN model



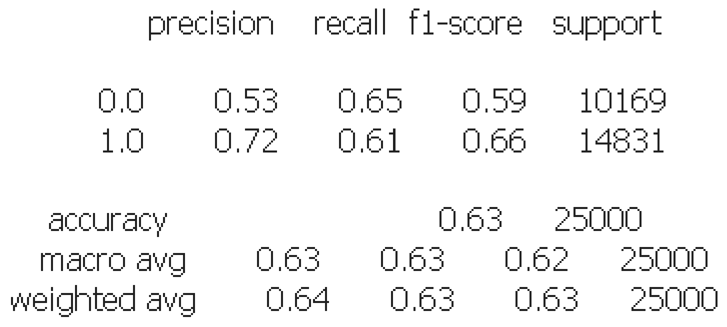
```
Epoch 1/10
20000/20000 [==============================] - 912s 46ms/step - loss: 0.6787 - accuracy: 0.5663 - val_loss: 0.6716 - val_accuracy: 0.5844
Epoch 2/10
20000/20000 [==============================] - 925s 46ms/step - loss: 0.6668 - accuracy: 0.5937 - val_loss: 0.6879 - val_accuracy: 0.5658
Epoch 3/10
20000/20000 [==============================] - 926s 46ms/step - loss: 0.6624 - accuracy: 0.6030 - val_loss: 0.6660 - val_accuracy: 0.5954
Epoch 4/10
20000/20000 [==============================] - 908s 45ms/step - loss: 0.6562 - accuracy: 0.6134 - val_loss: 0.6603 - val_accuracy: 0.6002
Epoch 5/10
20000/20000 [==============================] - 916s 46ms/step - loss: 0.6513 - accuracy: 0.6146 - val_loss: 0.6564 - val_accuracy: 0.6172
Epoch 6/10
20000/20000 [==============================] - 906s 45ms/step - loss: 0.6462 - accuracy: 0.6209 - val_loss: 0.6483 - val_accuracy: 0.6282
Epoch 7/10
20000/20000 [==============================] - 893s 45ms/step - loss: 0.6418 - accuracy: 0.6269 - val_loss: 0.6524 - val_accuracy: 0.6190
Epoch 8/10
20000/20000 [==============================] - 905s 45ms/step - loss: 0.6379 - accuracy: 0.6325 - val_loss: 0.6513 - val_accuracy: 0.6212
Epoch 9/10
20000/20000 [==============================] - 904s 45ms/step - loss: 0.6361 - accuracy: 0.6338 - val_loss: 0.6445 - val_accuracy: 0.6284
Epoch 10/10
20000/20000 [==============================] - 909s 45ms/step - loss: 0.6351 - accuracy: 0.6363 - val_loss: 0.6449 - val_accuracy: 0.6240
```

Figure 26 Training in the WordxWord+L2+BRNN model.



	Predicted Positive	Predicted Negative
Actual Positive	6655 (26.6%)	3514 (14.2%)
Actual Negative	5814 (23.3%)	9017 (36.1%)

Figure 27 The confusion matrix of the WordxWord+L2+BRNN mode

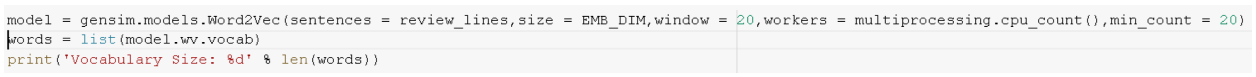


```
              precision    recall  f1-score   support

         0.0       0.53      0.65      0.59     10169
         1.0       0.72      0.61      0.66     14831

    accuracy                           0.63     25000
   macro avg       0.63      0.63      0.62     25000
weighted avg       0.64      0.63      0.63     25000
```

Figure 28 The classification report from the WordxWord+L2+BRNN model.

c. Word2vec

At the beginning, this model requires the word2vec's embedding index. To create the word2vec's embedding index, I defined the features that represent each word with setting the dimensions to 300 dimensions, window size = 50 and min_count to 20. The result from the word2vec's embedding index is matrix size 19500 (word) x300 (embedding dimension). According to our experiment, I found out that without adjusting the window_size and min_count, the size of the matrix will be larger, which will be a problem when used in BRNN will cause a memory error. .

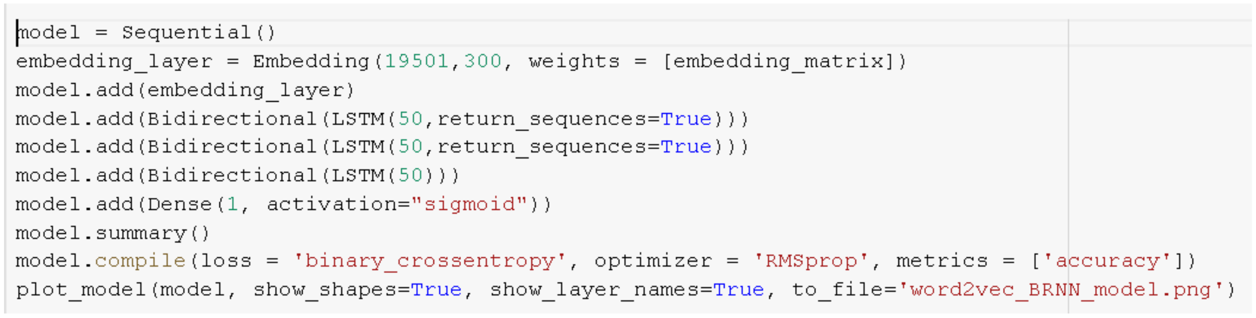


```
model = gensim.models.Word2Vec(sentences = review_lines,size = EMB_DIM,window = 20,workers = multiprocessing.cpu_count(),min_count = 20)
words = list(model.wv.vocab)
print('Vocabulary Size: %d' % len(words))
```

Figure 29 Code for building the word2vec

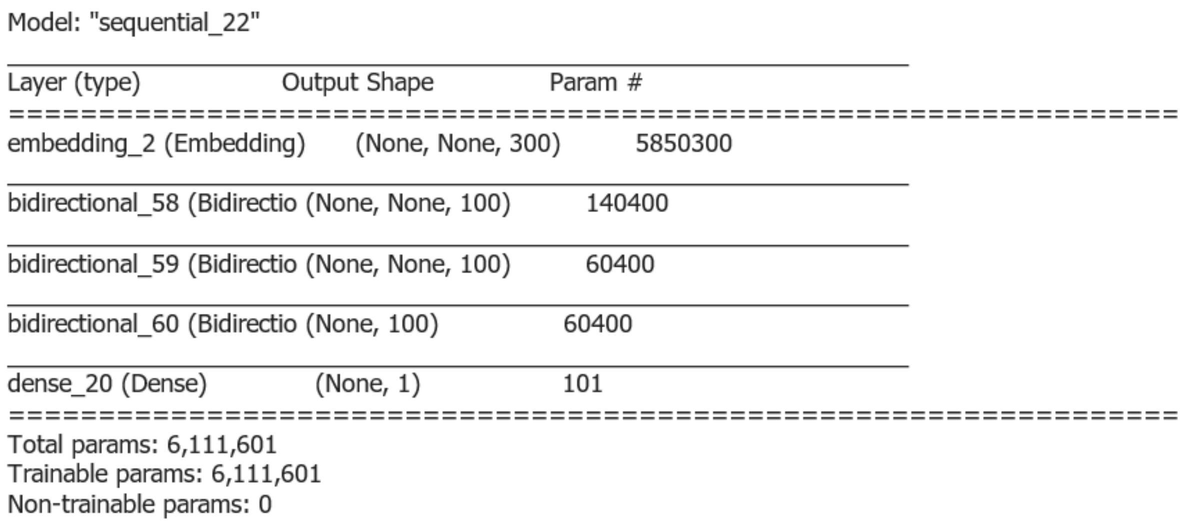
i. Word2Vec without the l2 regularization

I substituted the random matrix of weight in the embedding layer with the Word2Vec embedding matrix as shown in Figure 30. After the evaluation, this model can make predictions with an accuracy up to 88%.



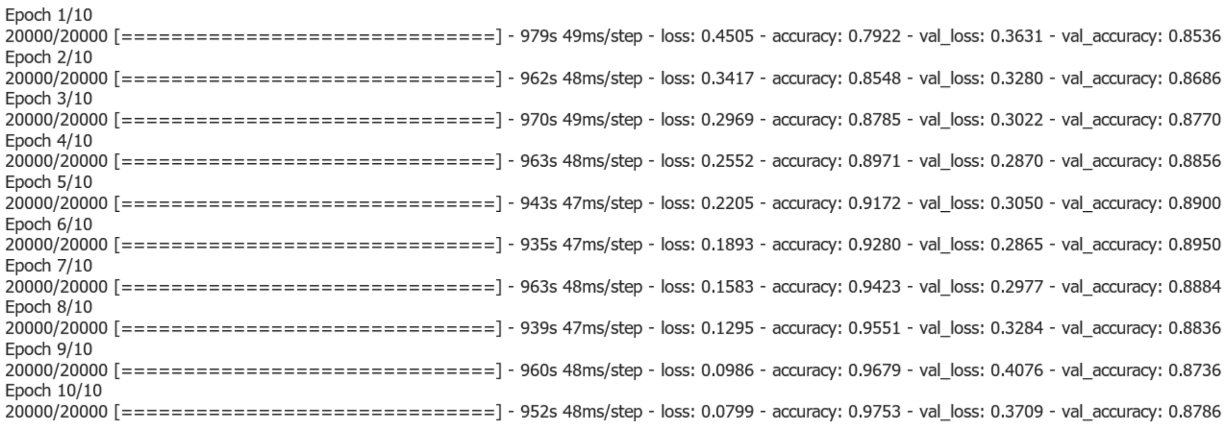
```
model = Sequential()
embedding_layer = Embedding(19501,300, weights = [embedding_matrix])
model.add(embedding_layer)
model.add(Bidirectional(LSTM(50,return_sequences=True)))
model.add(Bidirectional(LSTM(50,return_sequences=True)))
model.add(Bidirectional(LSTM(50)))
model.add(Dense(1, activation="sigmoid"))
model.summary()
model.compile(loss = 'binary_crossentropy', optimizer = 'RMSprop', metrics = ['accuracy'])
plot_model(model, show_shapes=True, show_layer_names=True, to_file='word2vec_BRNN_model.png')
```

Figure 30 The code for defining the structure of the word2vec+BRNN model.



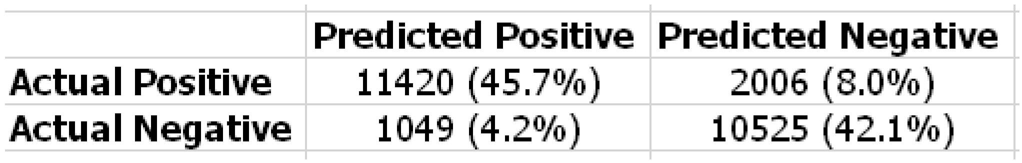
```
Model: "sequential_22"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, None, 300)         5850300
_________________________________________________________________
bidirectional_58 (Bidirectio (None, None, 100)         140400
_________________________________________________________________
bidirectional_59 (Bidirectio (None, None, 100)         60400
_________________________________________________________________
bidirectional_60 (Bidirectio (None, 100)               60400
_________________________________________________________________
dense_20 (Dense)             (None, 1)                 101
=================================================================
Total params: 6,111,601
Trainable params: 6,111,601
Non-trainable params: 0
```

Figure 31 The structure and number of parameters used in word2vec+BRNN model



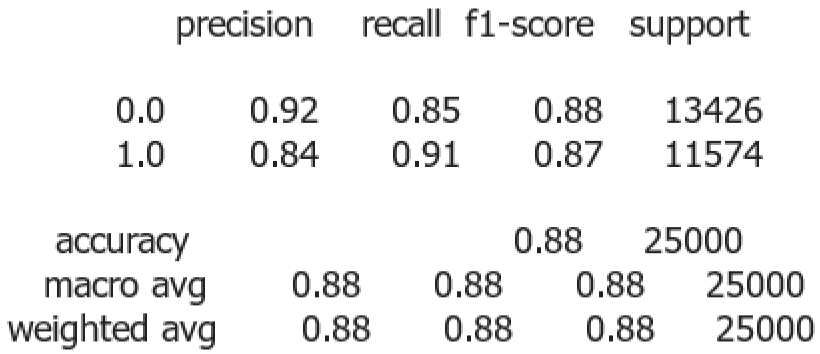
```
Epoch 1/10
20000/20000 [==============================] - 979s 49ms/step - loss: 0.4505 - accuracy: 0.7922 - val_loss: 0.3631 - val_accuracy: 0.8536
Epoch 2/10
20000/20000 [==============================] - 962s 48ms/step - loss: 0.3417 - accuracy: 0.8548 - val_loss: 0.3280 - val_accuracy: 0.8686
Epoch 3/10
20000/20000 [==============================] - 970s 49ms/step - loss: 0.2969 - accuracy: 0.8785 - val_loss: 0.3022 - val_accuracy: 0.8770
Epoch 4/10
20000/20000 [==============================] - 963s 48ms/step - loss: 0.2552 - accuracy: 0.8971 - val_loss: 0.2870 - val_accuracy: 0.8856
Epoch 5/10
20000/20000 [==============================] - 943s 47ms/step - loss: 0.2205 - accuracy: 0.9172 - val_loss: 0.3050 - val_accuracy: 0.8900
Epoch 6/10
20000/20000 [==============================] - 935s 47ms/step - loss: 0.1893 - accuracy: 0.9280 - val_loss: 0.2865 - val_accuracy: 0.8950
Epoch 7/10
20000/20000 [==============================] - 963s 48ms/step - loss: 0.1583 - accuracy: 0.9423 - val_loss: 0.2977 - val_accuracy: 0.8884
Epoch 8/10
20000/20000 [==============================] - 939s 47ms/step - loss: 0.1295 - accuracy: 0.9551 - val_loss: 0.3284 - val_accuracy: 0.8836
Epoch 9/10
20000/20000 [==============================] - 960s 48ms/step - loss: 0.0986 - accuracy: 0.9679 - val_loss: 0.4076 - val_accuracy: 0.8736
Epoch 10/10
20000/20000 [==============================] - 952s 48ms/step - loss: 0.0799 - accuracy: 0.9753 - val_loss: 0.3709 - val_accuracy: 0.8786
```

Figure 32 Training in the word2vec+BRNN model.



	Predicted Positive	Predicted Negative
Actual Positive	11420 (45.7%)	2006 (8.0%)
Actual Negative	1049 (4.2%)	10525 (42.1%)

Figure 33 The confusion matrix of the word2vec+BRNN model.

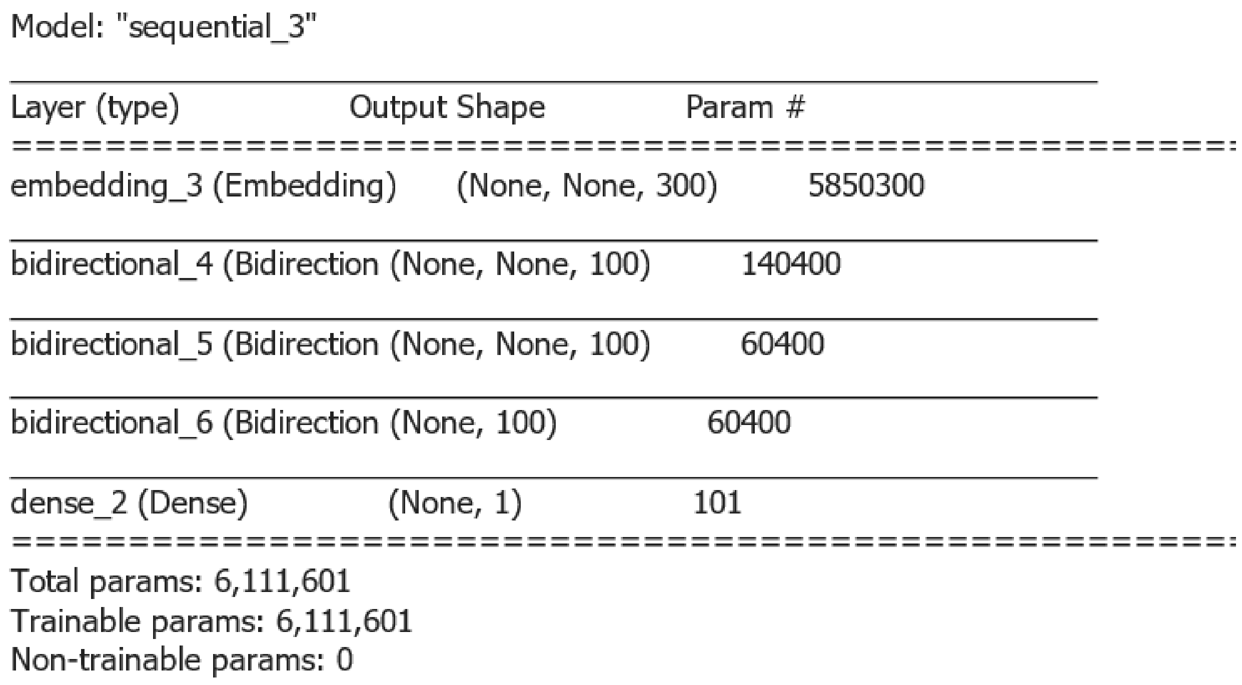


```
              precision    recall  f1-score   support

         0.0       0.92      0.85      0.88     13426
         1.0       0.84      0.91      0.87     11574

    accuracy                           0.88     25000
   macro avg       0.88      0.88      0.88     25000
weighted avg       0.88      0.88      0.88     25000
```

Figure 34 The classification report from the word2vec+BRNN model.

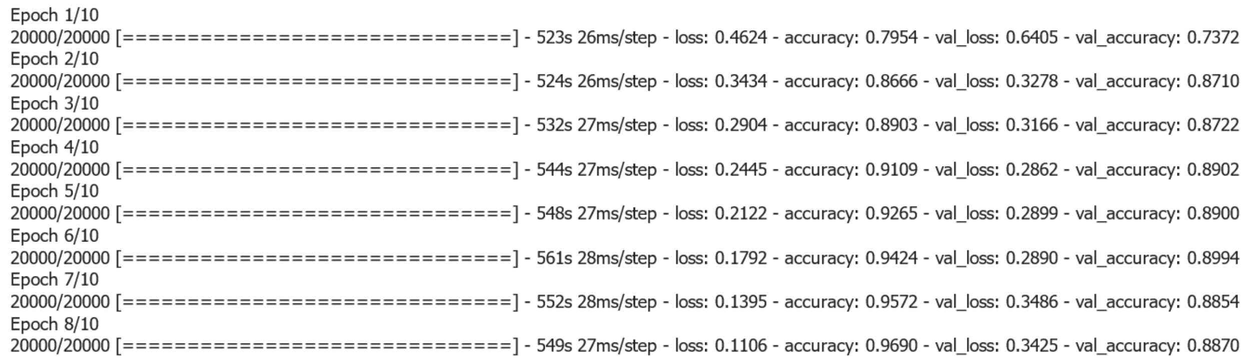
ii. Word2Vec with the l2 regularization

In this experiment I included the additional parameters by adding L2 to each LSTM layer (kernel_regularizer = l2 (0.0000001) and activity_regularizer = l2 (0.0000001)). Finally, this model can overcome the previous model with an accuracy of 89%.



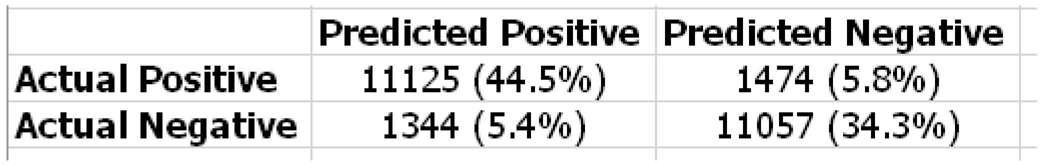
```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, None, 300)         5850300
_________________________________________________________________
bidirectional_4 (Bidirection (None, None, 100)         140400
_________________________________________________________________
bidirectional_5 (Bidirection (None, None, 100)         60400
_________________________________________________________________
bidirectional_6 (Bidirection (None, 100)               60400
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 101
=================================================================
Total params: 6,111,601
Trainable params: 6,111,601
Non-trainable params: 0
```

Figure 35 The structure and number of parameters used in word2vec+l2+BRNN model



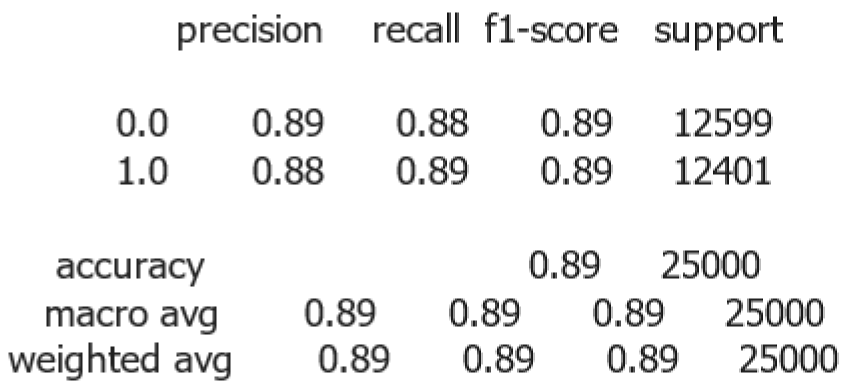
```
Epoch 1/10
20000/20000 [==============================] - 523s 26ms/step - loss: 0.4624 - accuracy: 0.7954 - val_loss: 0.6405 - val_accuracy: 0.7372
Epoch 2/10
20000/20000 [==============================] - 524s 26ms/step - loss: 0.3434 - accuracy: 0.8666 - val_loss: 0.3278 - val_accuracy: 0.8710
Epoch 3/10
20000/20000 [==============================] - 532s 27ms/step - loss: 0.2904 - accuracy: 0.8903 - val_loss: 0.3166 - val_accuracy: 0.8722
Epoch 4/10
20000/20000 [==============================] - 544s 27ms/step - loss: 0.2445 - accuracy: 0.9109 - val_loss: 0.2862 - val_accuracy: 0.8902
Epoch 5/10
20000/20000 [==============================] - 548s 27ms/step - loss: 0.2122 - accuracy: 0.9265 - val_loss: 0.2899 - val_accuracy: 0.8900
Epoch 6/10
20000/20000 [==============================] - 561s 28ms/step - loss: 0.1792 - accuracy: 0.9424 - val_loss: 0.2890 - val_accuracy: 0.8994
Epoch 7/10
20000/20000 [==============================] - 552s 28ms/step - loss: 0.1395 - accuracy: 0.9572 - val_loss: 0.3486 - val_accuracy: 0.8854
Epoch 8/10
20000/20000 [==============================] - 549s 27ms/step - loss: 0.1106 - accuracy: 0.9690 - val_loss: 0.3425 - val_accuracy: 0.8870
```

Figure 36 Training the word2vec+l2+BRNN model.



	Predicted Positive	Predicted Negative
Actual Positive	11125 (44.5%)	1474 (5.8%)
Actual Negative	1344 (5.4%)	11057 (34.3%)

Figure 34 The confusion matrix of the word2vec+l2+BRNN model.



```
              precision    recall  f1-score   support

         0.0       0.89      0.88      0.89     12599
         1.0       0.88      0.89      0.89     12401

    accuracy                           0.89     25000
   macro avg       0.89      0.89      0.89     25000
weighted avg       0.89      0.89      0.89     25000
```

Figure 37 The classification report from the word2vec+BRNN model.

In summary, all the parameters maintained throughout the 8 models are fixed as follows:

- Training 20,000 rows, validation 5,000 rows and testing 25,000 rows. Random state = 38.
- Embedding matrix size 19,501 x 300.
- Input length = 300 words.
- 3 consecutive BRNN layers (LSTM).
- 1 Dense with Sigmoid activation function.
- Loss = binary_cross entropy.
- The optimizer = RMSprop.
- Batch size = 100.
- Epoch = 10.
- Early stop with patience = 4.
- Total parameters are 6,111,601.

From Table 1, I experimented with 8 models and used the BRNN and BRNN + l2 models as baseline models. In comparison, I compared the results in 3 aspects including: 1). Number of epochs used in training, 2). Duration/epoch, and 3). Accuracy.

Firstly, the baseline model uses a minimum of 6 epochs to give an accuracy of 85-87%. From the experiment, the results reported that most models take up to 10 epoches as I have initially set. However, in the model one-hot + l2 + BRNN I found that models can continue to learn, but their learning is slower (training accuracy increases 0.01%/epoch) than other models because using one-hot technique has a lot of zero value that affects gradient.

Second, most models consume 520-979 seconds in the process of learning. The timing may vary, for example in the word2vec + BRNN model and the word2vec + l2 + BRNN model depending on the use of the GPU (sometimes I trained more than 2 models at the same time).

Finally, the model that can overcome other models with accuracy up to 89% is the word2vec + l2 + BRNN model. From the experiment, I discovered that adding l2 to the model improves the performance of the models except in the case of the baseline model.

Table 1 Overview of results from each model.

Model	**Number of epoch**	**Time/epoch (s)**	**Accuracy (%)**
BRNN	6	523	87
BRNN+l2	6	520	85
one-hot+BRNN	8	928	57
one-hot+l2+BRNN	10	918	58
wordxword+BRNN	10	907	62
wordxword+l2+BRNN	10	926	63
word2vec+BRNN	10	979	88
word2vec+l2+BRNN	8	552	89

4. Use the TED – Ultimate Dataset (only the file ted_talks_en.csv), to perform topic modeling. Explain the process of your work, the reason behind the technique that you used, and the results you obtained.

1). Data Exploration.

This data has a total of 4005 rows. I considered 2 columns which are Topic and Transcript columns. Firstly, I plotted the distribution of the number of topics as shown in figure 38 and calculated some statistical value. The result showed that the distribution is very positive skew, max topic = 39
min topic in 1 row, mean = 9.17, and median = 8 respectively.

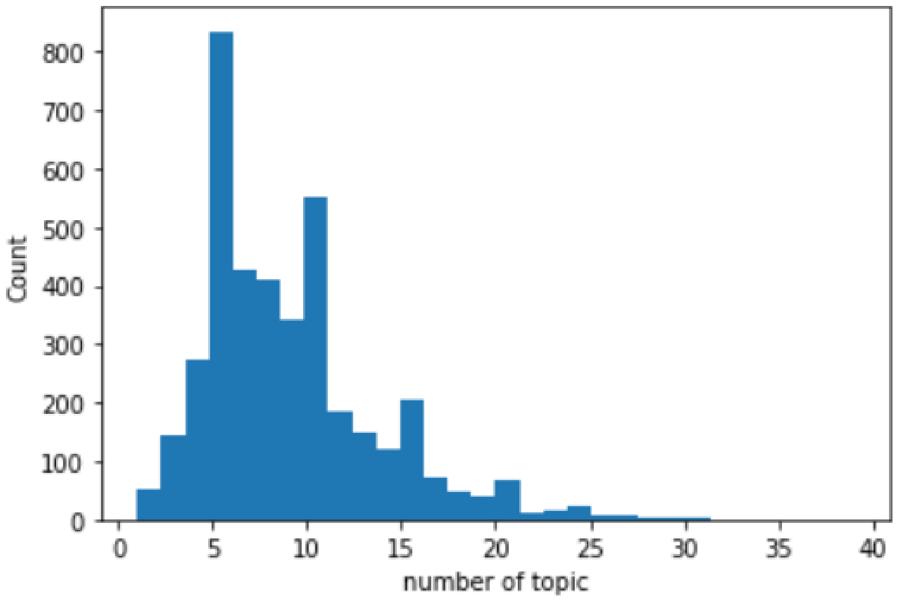


Figure 38 The distribution of the number of topics.

2). Data Preparation.

The textual data in the transcription column is the main input that I will use to perform the topic modeling.The transcription column is prepared by removing </br>, punctuation, not alphabetic, doing the lemmatization, and also lower case the character. Finally, the maximum number of results is 11446 words, the smallest 2, the mean 1822.66 and the median 1772, respectively.

3). Modeling.

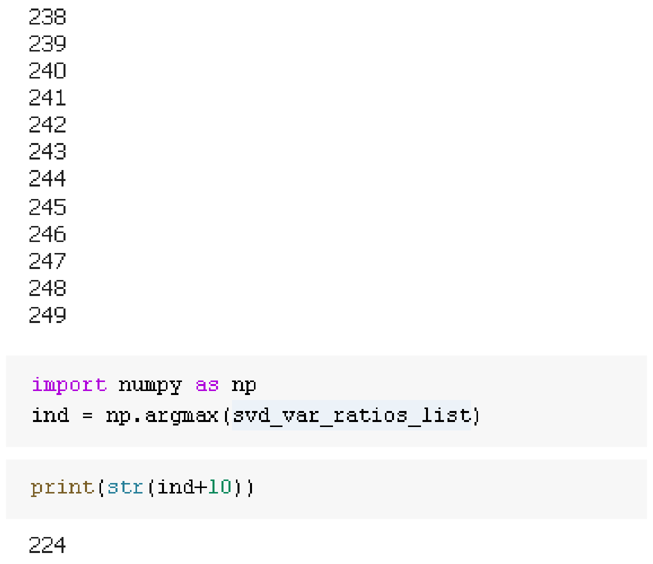
I performed 2 topic modeling algorithms including the Latent Semantic Analysis (LSA) and the Latent Dirichlet Allocation (LDA).

For the LSA algorithm, this model uses the concept of the Single Value Decomposition (SVD). The LSA model is fast and efficient to use. In fact, there is a parameter that I need to adjust, which is max_features or the number of topics. LSA performance measurement can be measured by the sum of variance_ratio, which high values are considered good performance.

In the second model, LDA has been mentioned that they are popular tools for text analysis, providing both a predictive and latent topic representation of the corpus. However, the main parameter that needs to be adjusted in this model is the appropriate number of latent topics, so the performance can be measured from the Coherence Score.

4). Results.

I used the Grid Search to choose the best n_component parameter in the LSA model (start at 10 topics to 250 topics with step = 1). The result of the highest sum of variance_ratio came from index 214, which refers to 224 topics is the number of topics suitable to divide. The LSA model is fast and efficient to use, but it does have a few disadvantages, for example: lack of interpretable embedding and less efficient representation.



```
238
239
240
241
242
243
244
245
246
247
248
249
```

```
import numpy as np
ind = np.argmax(svd_var_ratios_list)
```

```
print(str(ind+10))
```

```
224
```

Figure 39 The result from LSA Grid Search.

I also used the Grid Search again to choose the best parameter n_component parameter in the LDA model (start at 10 topics to 250 topics with step = 5, random_state=38). The highest Coherence Score is 0.271 as shown in figure 40 by using the number of topics at 35 topics.

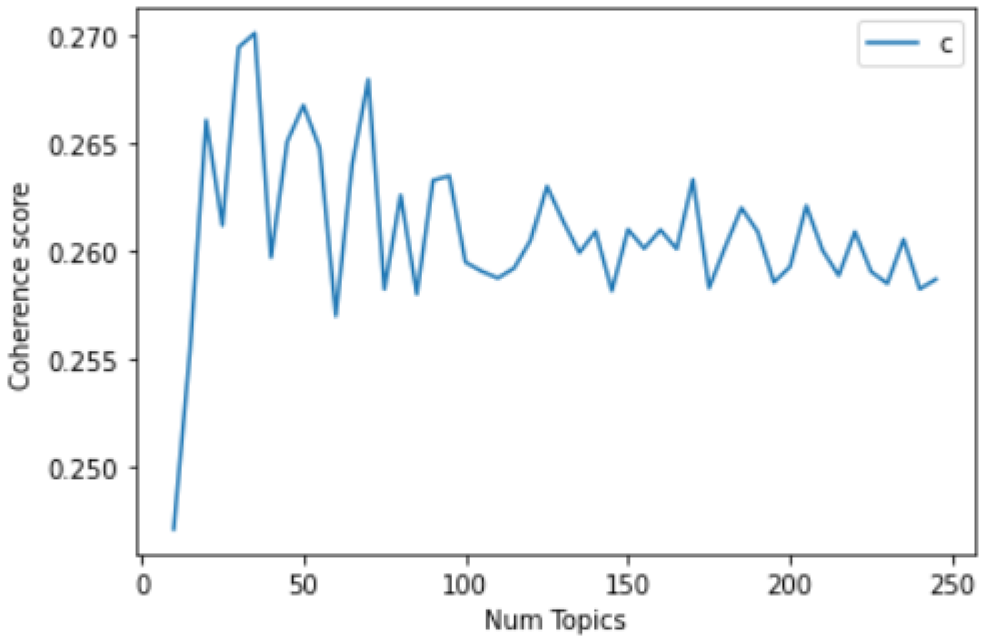
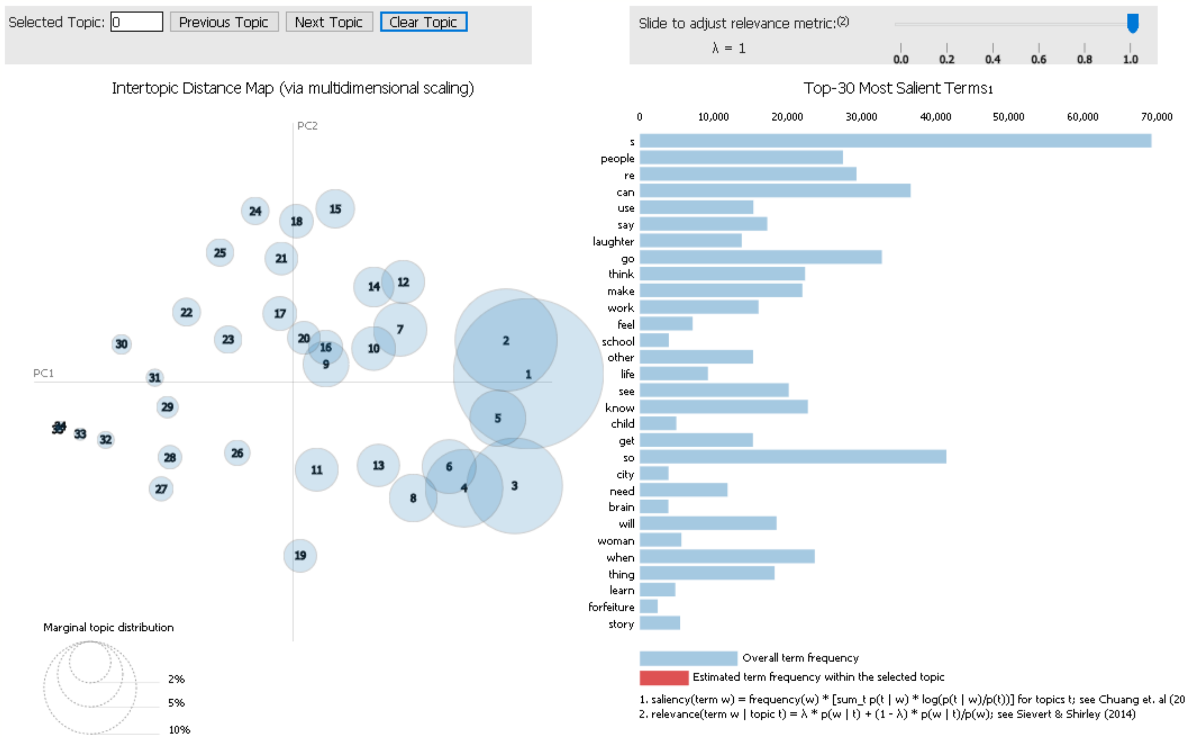


Figure 40 The Coherence Score for each topic.

The topic modeling visualization is shown in figure 41 in 2-dimensions. However, as we all know, the topic modeling is unsupervised learning, so if we want to know the meaning of each topic. It is necessary to consider the groups of key words in that group and infer the type of topic from their key words.

However, it has 35 topic groups in this topic modeling, difficult to explain all the information for each topic. Therefore, we have chosen only some topic. In figure 41, 28th topic, it uses keywords such as sound, piano, tone, musician, singing, etc. These words fall into the category of music. Therefore, group 28 should be able to infer that it is a music topic. In figure 42, from the keywords used, can be inplied as the health topic. The 30th topic from figure 43 can be predicted as the food topic. Finally, the 33rd topic can be inferred as the coronavirus disease topic.

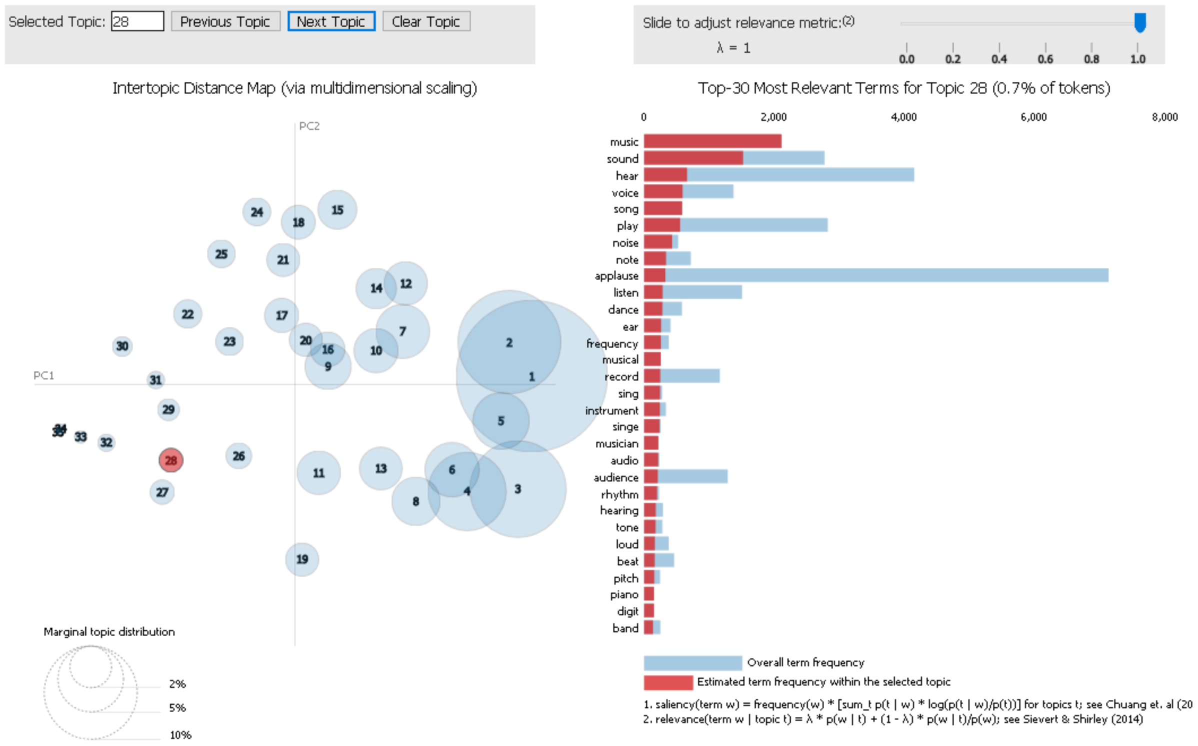


Selected Topic: 0

Slide to adjust relevance metric:(2) λ = 1

1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (20
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

Figure 40 All topics from the LDA modeling.

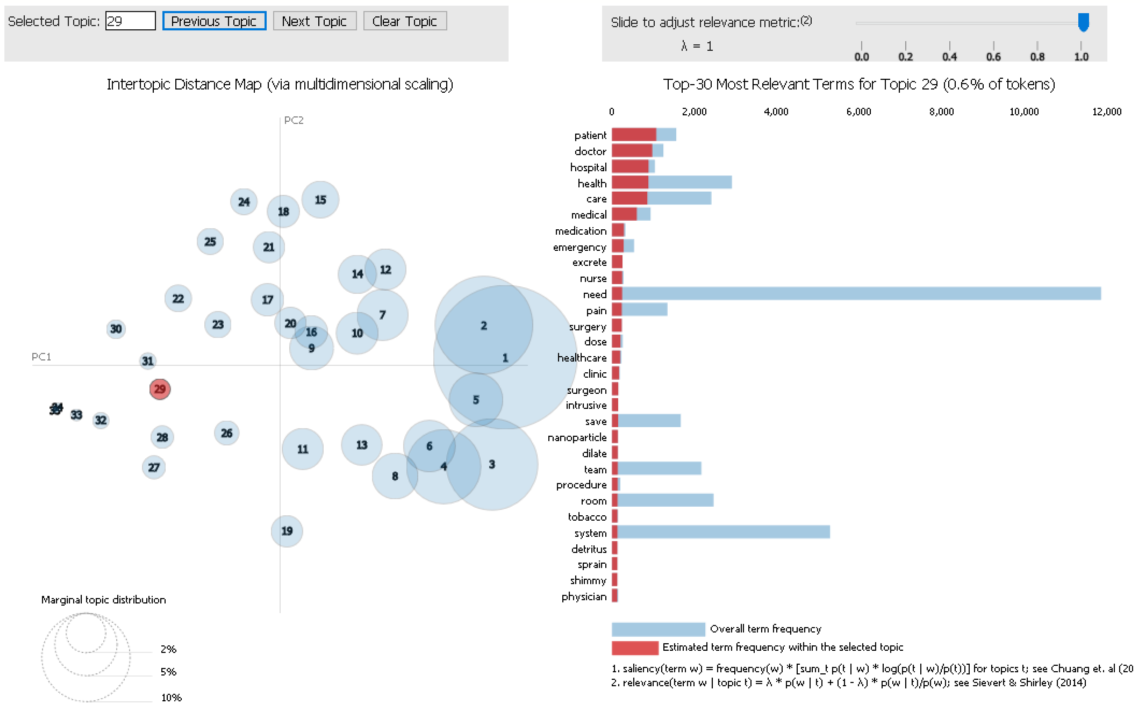


Selected Topic: 28

Slide to adjust relevance metric:(2) λ = 1

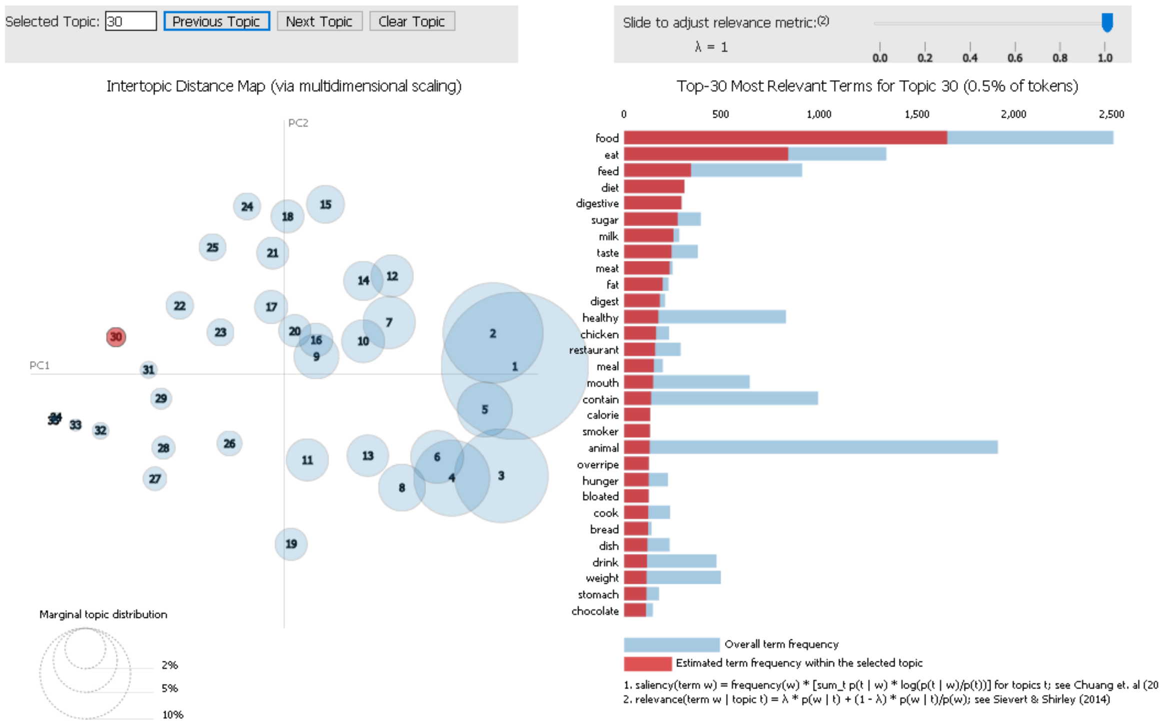
1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (20
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

Figure 41 28th topic from the LDA modeling.



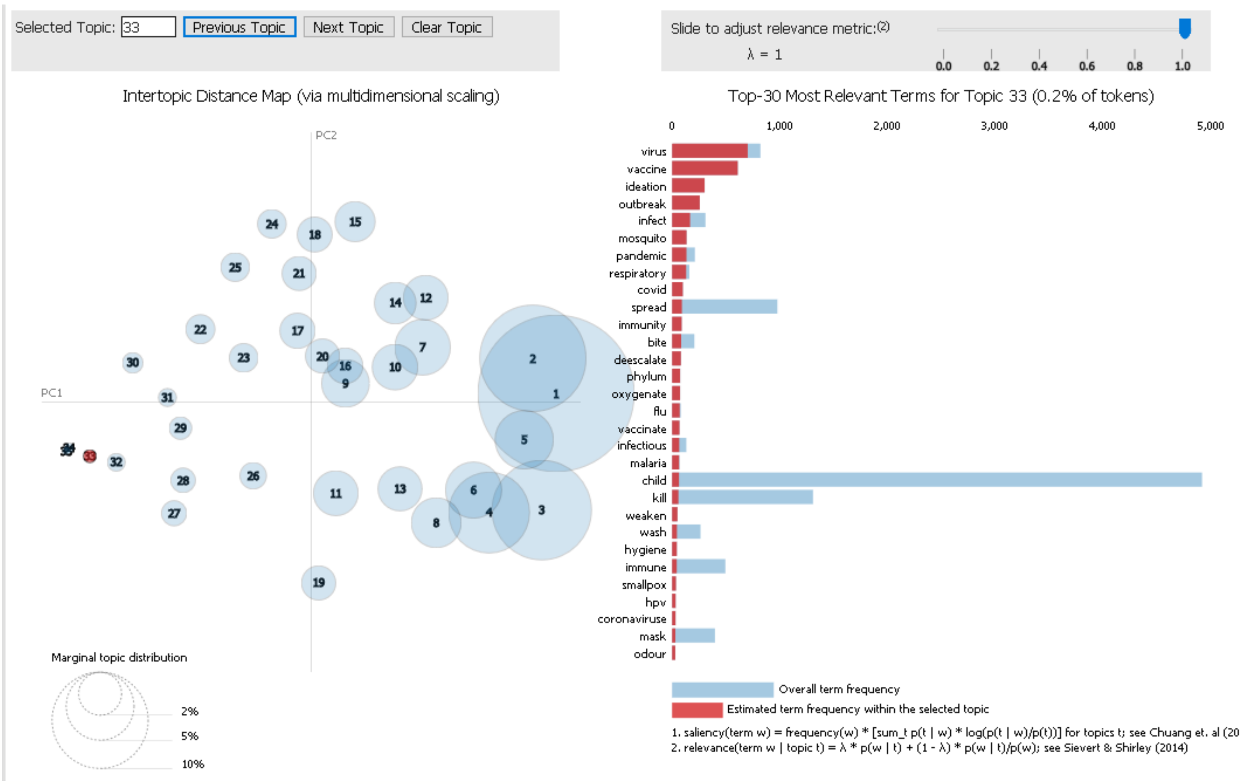
1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (20
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

Figure 42 29th topic from the LDA modeling.



1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (20
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

Figure 43 30th topic from the LDA modeling.



1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (20
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

Figure 44 33rd topic from the LDA modeling.

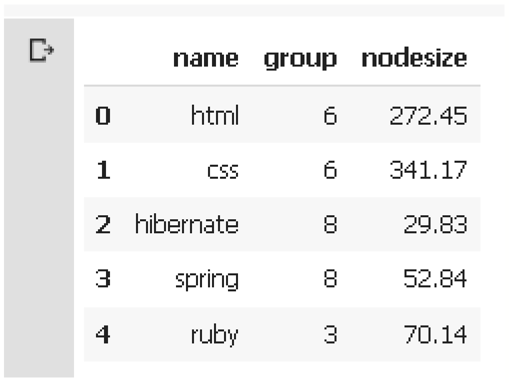
5. Use the Stack Overflow Tag Network dataset, build and analyze the network of tags in stack overflow. Explain the process that you use and the finding in detail.

   I set the objective of the analysis in this network is to explain **how technologies are connected and how they are used together.**

   I divided the topics in this work into 4 parts as follows: 1). Data understanding, 2) Social Networks Visualization, 3) Network-Level Analysis, 4) Group-Level Analysis and 5) Node-Level Analysis.

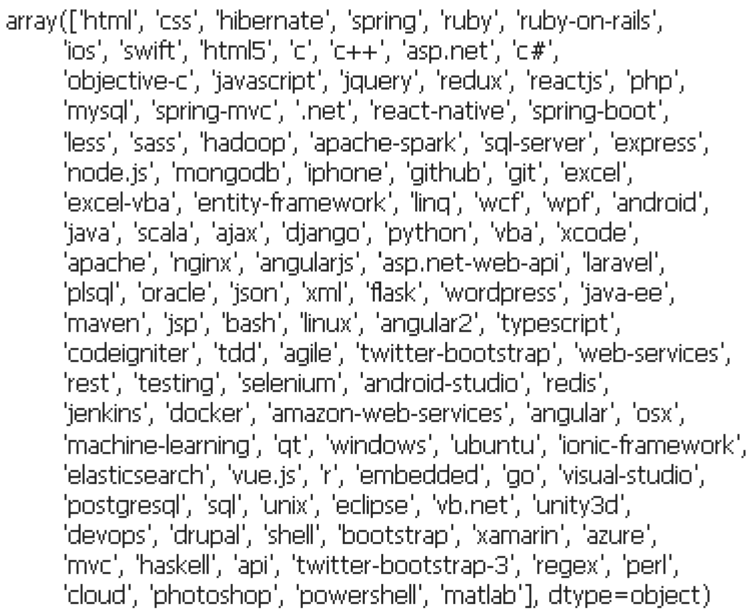
   1) Data understanding.
      a) Stack_network_nodes.csv node.

         This file consists of 115 entries with 3 columns including name, group and nodesize. There are 115 unique technologies in the name column. They can be distinguished into 14 groups (1-14), the distribution of each group is shown in figure 46. The top 3 technology groups are the 6th, 2nd, and 8th groups. However, all details of the technology names in each group are shown in Figure 47. For nodesize's distribution, it is a very positive skew.



	name	group	nodesize
0	html	6	272.45
1	css	6	341.17
2	hibernate	8	29.83
3	spring	8	52.84
4	ruby	3	70.14

Figure 44 The 5 rows of the Stack_network_nodes file.



```
array(['html', 'css', 'hibernate', 'spring', 'ruby', 'ruby-on-rails',
       'ios', 'swift', 'html5', 'c', 'c++', 'asp.net', 'c#',
       'objective-c', 'javascript', 'jquery', 'redux', 'reactjs', 'php',
       'mysql', 'spring-mvc', '.net', 'react-native', 'spring-boot',
       'less', 'sass', 'hadoop', 'apache-spark', 'sql-server', 'express',
       'node.js', 'mongodb', 'iphone', 'github', 'git', 'excel',
       'excel-vba', 'entity-framework', 'linq', 'wcf', 'wpf', 'android',
       'java', 'scala', 'ajax', 'django', 'python', 'vba', 'xcode',
       'apache', 'nginx', 'angularjs', 'asp.net-web-api', 'laravel',
       'plsql', 'oracle', 'json', 'xml', 'flask', 'wordpress', 'java-ee',
       'maven', 'jsp', 'bash', 'linux', 'angular2', 'typescript',
       'codeigniter', 'tdd', 'agile', 'twitter-bootstrap', 'web-services',
       'rest', 'testing', 'selenium', 'android-studio', 'redis',
       'jenkins', 'docker', 'amazon-web-services', 'angular', 'osx',
       'machine-learning', 'qt', 'windows', 'ubuntu', 'ionic-framework',
       'elasticsearch', 'vue.js', 'r', 'embedded', 'go', 'visual-studio',
       'postgresql', 'sql', 'unix', 'eclipse', 'vb.net', 'unity3d',
       'devops', 'drupal', 'shell', 'bootstrap', 'xamarin', 'azure',
       'mvc', 'haskell', 'api', 'twitter-bootstrap-3', 'regex', 'perl',
       'cloud', 'photoshop', 'powershell', 'matlab'], dtype=object)
```

Figure 45 List of all technologies.

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc6ef3a02b0>
```

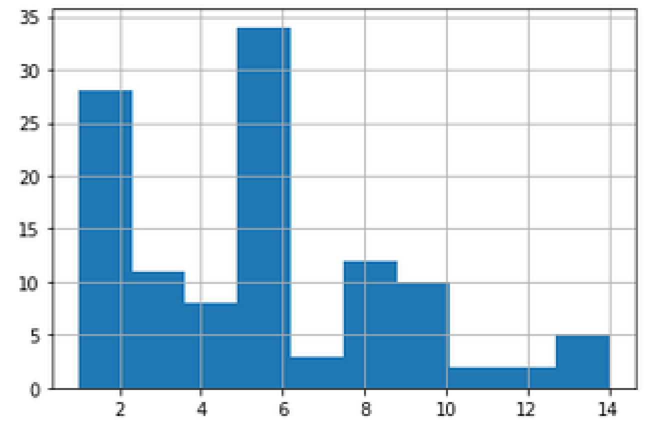
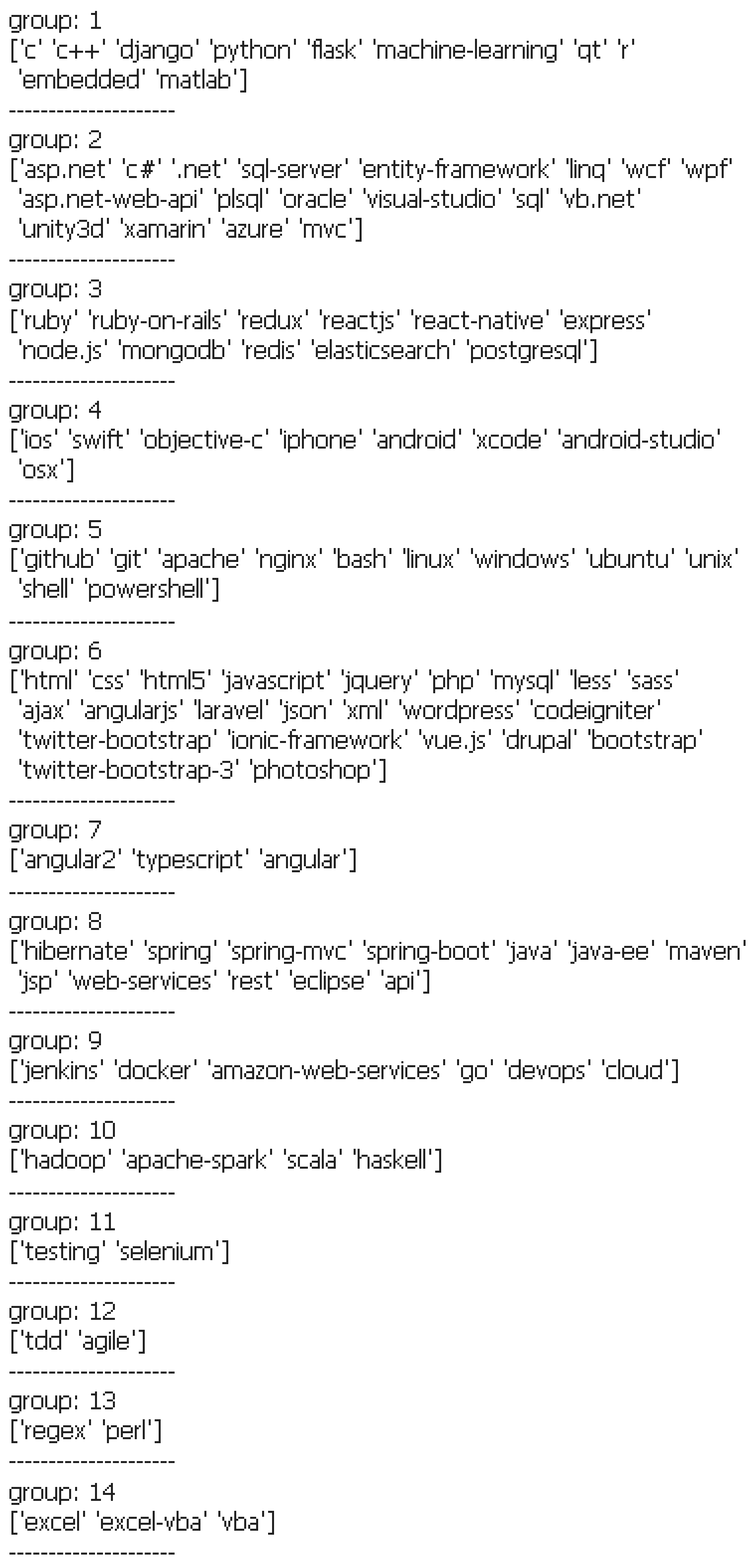
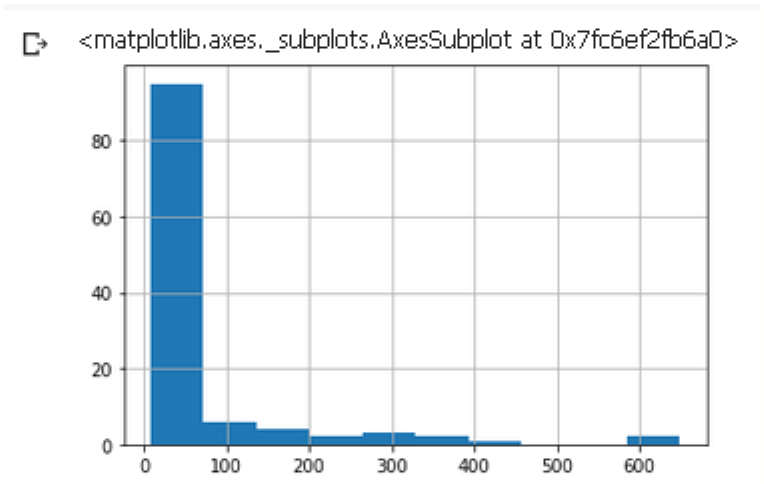


Figure 46 The distribution of each technology group.



```
group: 1
['c' 'c++' 'django' 'python' 'flask' 'machine-learning' 'qt' 'r'
 'embedded' 'matlab']
---------------------
group: 2
['asp.net' 'c#' '.net' 'sql-server' 'entity-framework' 'linq' 'wcf' 'wpf'
 'asp.net-web-api' 'plsql' 'oracle' 'visual-studio' 'sql' 'vb.net'
 'unity3d' 'xamarin' 'azure' 'mvc']
---------------------
group: 3
['ruby' 'ruby-on-rails' 'redux' 'reactjs' 'react-native' 'express'
 'node.js' 'mongodb' 'redis' 'elasticsearch' 'postgresql']
---------------------
group: 4
['ios' 'swift' 'objective-c' 'iphone' 'android' 'xcode' 'android-studio'
 'osx']
---------------------
group: 5
['github' 'git' 'apache' 'nginx' 'bash' 'linux' 'windows' 'ubuntu' 'unix'
 'shell' 'powershell']
---------------------
group: 6
['html' 'css' 'html5' 'javascript' 'jquery' 'php' 'mysql' 'less' 'sass'
 'ajax' 'angularjs' 'laravel' 'json' 'xml' 'wordpress' 'codeigniter'
 'twitter-bootstrap' 'ionic-framework' 'vue.js' 'drupal' 'bootstrap'
 'twitter-bootstrap-3' 'photoshop']
---------------------
group: 7
['angular2' 'typescript' 'angular']
---------------------
group: 8
['hibernate' 'spring' 'spring-mvc' 'spring-boot' 'java' 'java-ee' 'maven'
 'jsp' 'web-services' 'rest' 'eclipse' 'api']
---------------------
group: 9
['jenkins' 'docker' 'amazon-web-services' 'go' 'devops' 'cloud']
---------------------
group: 10
['hadoop' 'apache-spark' 'scala' 'haskell']
---------------------
group: 11
['testing' 'selenium']
---------------------
group: 12
['tdd' 'agile']
---------------------
group: 13
['regex' 'perl']
---------------------
group: 14
['excel' 'excel-vba' 'vba']
---------------------
```

Figure 47 the technology names that are grouped.

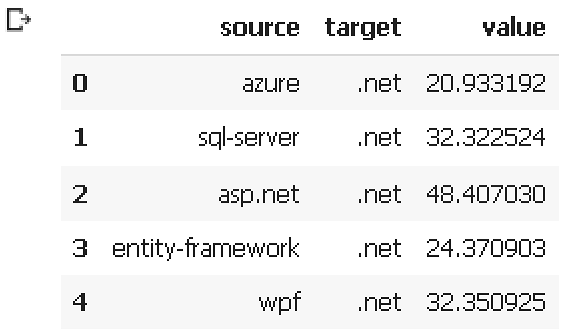


```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc6ef2fb6a0>
```

Figure 48 The distribution of nodesize column.

b) Stack_network_links.

This file stores information about each node's links as in Figure 49 with 490 entries. There are 3 columns including: source, target, and value. The result from value's distribution shows that it is a positive skew.



	source	target	value
0	azure	.net	20.933192
1	sql-server	.net	32.322524
2	asp.net	.net	48.407030
3	entity-framework	.net	24.370903
4	wpf	.net	32.350925

Figure 49 The top 5 of the Stack_network_links.

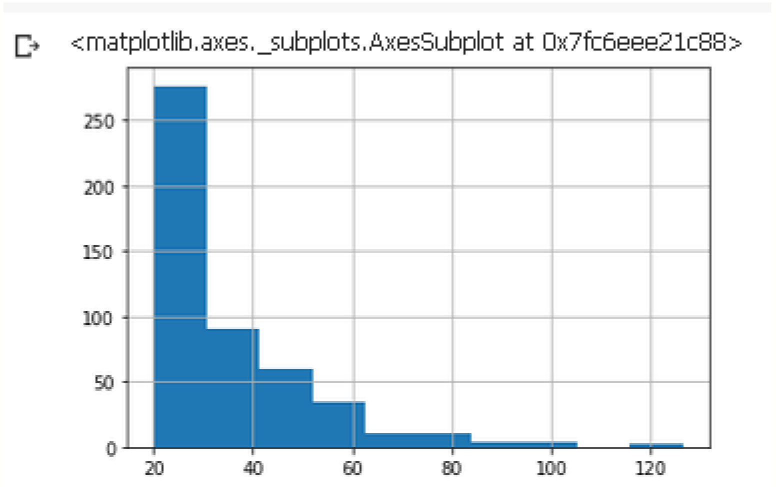

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc6eee21c88>
```

Figure 48 The distribution of value column.

2) Social Networks Visualization.

When I combined 2 files from the first section, and printed the information of the graph, the results show as below information:

Type: Graph
Number of nodes: 115
Number of edges: 245
Average degree: 4.2609

After that, I plotted the networks graph with results as in figure 50. I found that the results contained 6 sub-graphs: most graphs are in the middle to the right. A few are in the top left and bottom left areas.

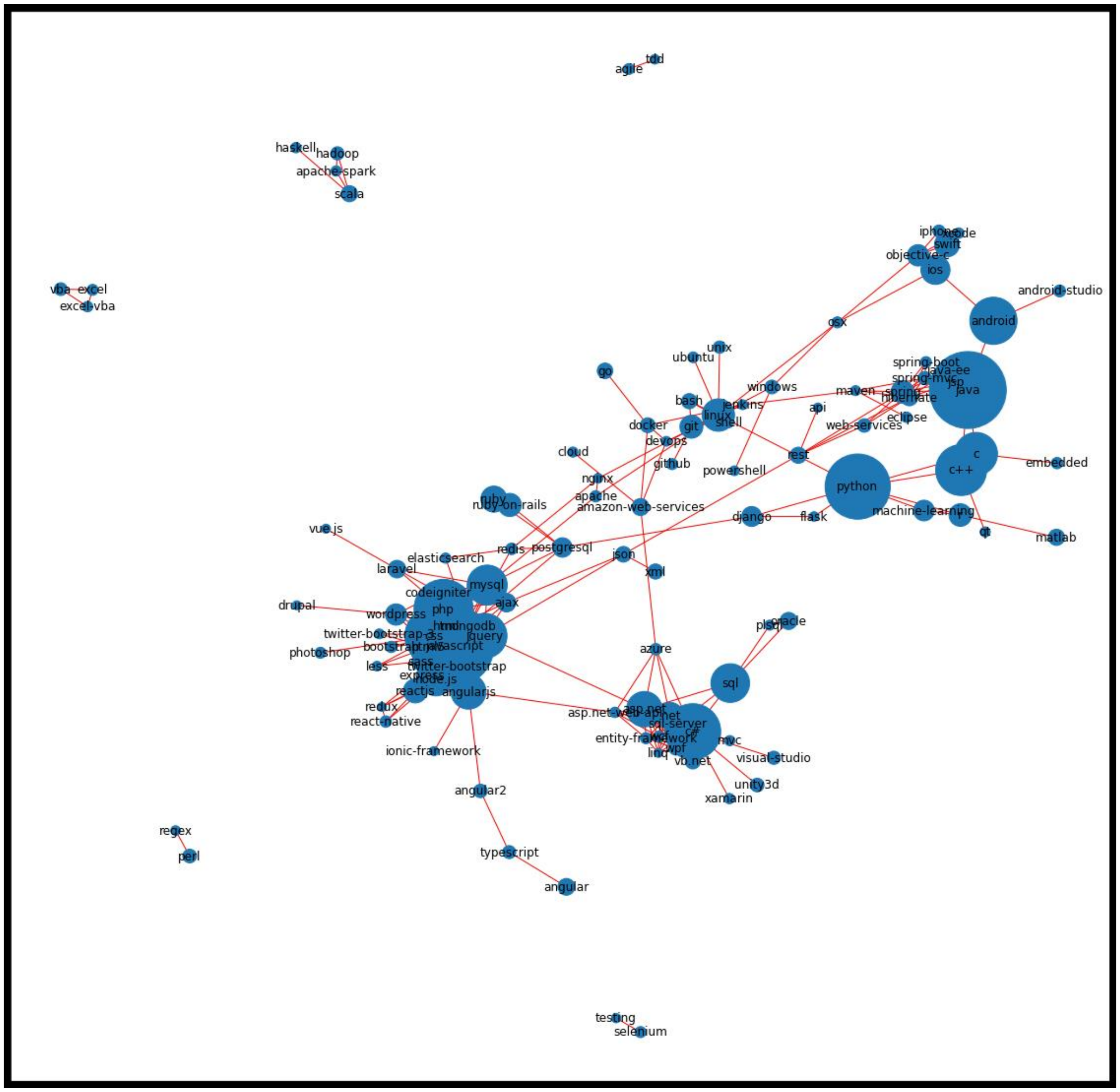


Figure 50 The networks graph of the Stack Overflow Tag Network dataset.

When we consider the 5 sub-graphs, it consists of the following information

1st group (in the top center of image 50) : agile and tdd.

2nd group (In the top center, indented to the left of image 50.) : haskell, hadoop, apache-spark and scala.

3rd group (On the left of figure 50) : vba, excel and excel-vba.

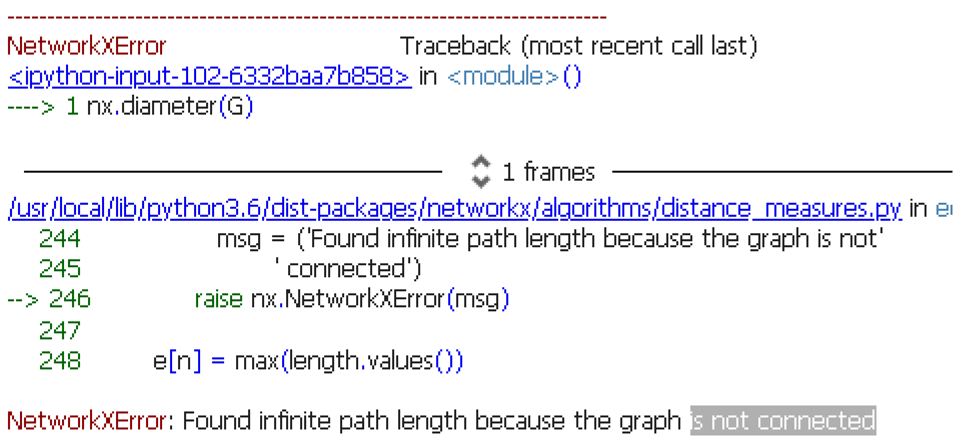
4th group (in the bottom left corner of the image) : regex and perl.

5th group (at the bottom of figure 50) : testing and selenium.

3) Network-Level Analysis

In this section, I focus on the network- level analysis. Initially, I used the code **“nx.number_connected_components(G)”** the result showed the number 6, which means there are 6 graphs that are not linked to each other, so this number confirms the results from image 50.

Next, I calculate the **density** of this network, the density is defined as the number of connections a participant has, divided by the total possible connections a participant could have. in that network. All values of density are between 0 and 1. The result report with small value at 0.03737, which means that 1 technology only connects with few technologies (0.37*115= [3-4] groups) when compared to all networks. However, with the reason that this network is not fully connected so we can not calculate the diameter of the network.

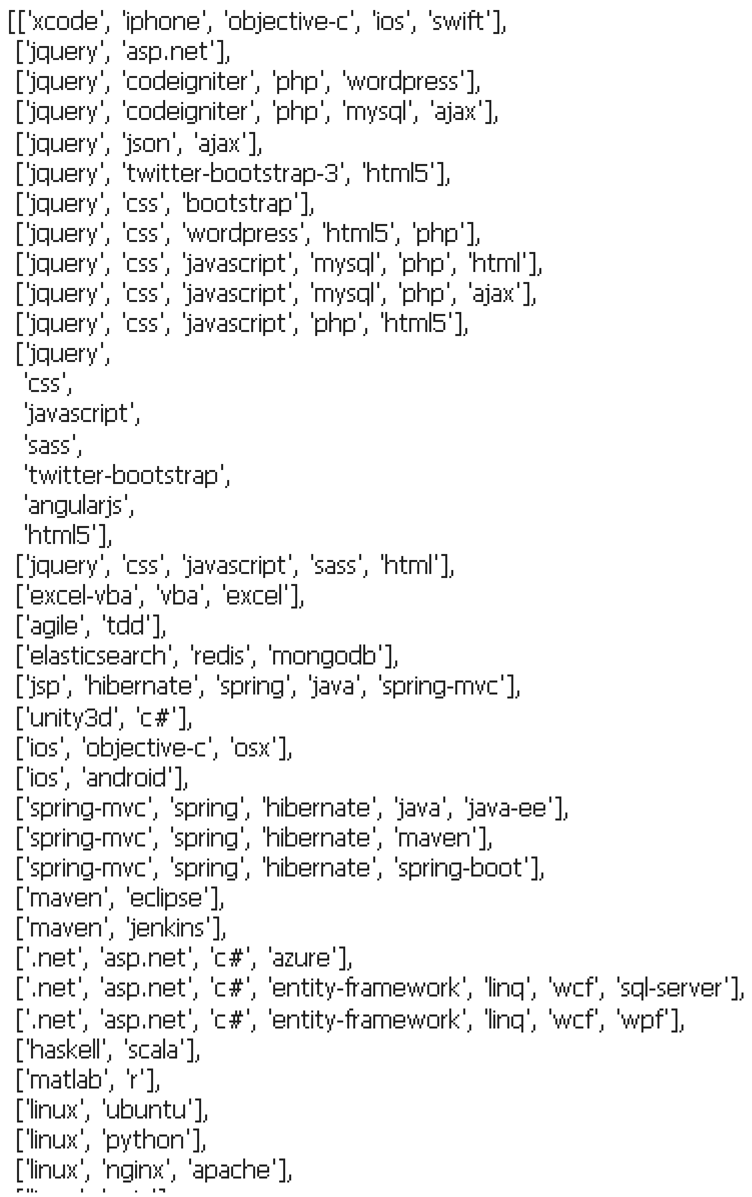


```
---------------------------------------------------------------------------
NetworkXError                             Traceback (most recent call last)
<ipython-input-102-6332baa7b858> in <module>()
----> 1 nx.diameter(G)

1 frames
/usr/local/lib/python3.6/dist-packages/networkx/algorithms/distance_measures.py in e
    244                 msg = ('Found infinite path length because the graph is not'
    245                        ' connected')
--> 246             raise nx.NetworkXError(msg)
    247 
    248         e[n] = max(length.values())

NetworkXError: Found infinite path length because the graph is not connected
```

Figure 51 The error message when running diameter function.

4) Group-Level Analysis

In this section, I demonstrate the techniques for detecting cohesive groups in the networks. First, I attempted to find mutual technology in this social network by using nx.find_cliques() function. There are 89 cliques in this network, for example in figure 51.



```
[['xcode', 'iphone', 'objective-c', 'ios', 'swift'],
 ['jquery', 'asp.net'],
 ['jquery', 'codeigniter', 'php', 'wordpress'],
 ['jquery', 'codeigniter', 'php', 'mysql', 'ajax'],
 ['jquery', 'json', 'ajax'],
 ['jquery', 'twitter-bootstrap-3', 'html5'],
 ['jquery', 'css', 'bootstrap'],
 ['jquery', 'css', 'wordpress', 'html5', 'php'],
 ['jquery', 'css', 'javascript', 'mysql', 'php', 'html'],
 ['jquery', 'css', 'javascript', 'mysql', 'php', 'ajax'],
 ['jquery', 'css', 'javascript', 'php', 'html5'],
 ['jquery',
  'css',
  'javascript',
  'sass',
  'twitter-bootstrap',
  'angularjs',
  'html5'],
 ['jquery', 'css', 'javascript', 'sass', 'html'],
 ['excel-vba', 'vba', 'excel'],
 ['agile', 'tdd'],
 ['elasticsearch', 'redis', 'mongodb'],
 ['jsp', 'hibernate', 'spring', 'java', 'spring-mvc'],
 ['unity3d', 'c#'],
 ['ios', 'objective-c', 'osx'],
 ['ios', 'android'],
 ['spring-mvc', 'spring', 'hibernate', 'java', 'java-ee'],
 ['spring-mvc', 'spring', 'hibernate', 'maven'],
 ['spring-mvc', 'spring', 'hibernate', 'spring-boot'],
 ['maven', 'eclipse'],
 ['maven', 'jenkins'],
 ['.net', 'asp.net', 'c#', 'azure'],
 ['.net', 'asp.net', 'c#', 'entity-framework', 'linq', 'wcf', 'sql-server'],
 ['.net', 'asp.net', 'c#', 'entity-framework', 'linq', 'wcf', 'wpf'],
 ['haskell', 'scala'],
 ['matlab', 'r'],
 ['linux', 'ubuntu'],
 ['linux', 'python'],
 ['linux', 'nginx', 'apache'],
```

Figure 51 The example results from nx.find_cliques() function.

Next, we searched for the maximum clique size, the result shownd 7. There are 3 maximum cliques including

[

['jquery', 'css', 'javascript', 'sass', 'twitter-bootstrap', 'angularjs', 'html5'],

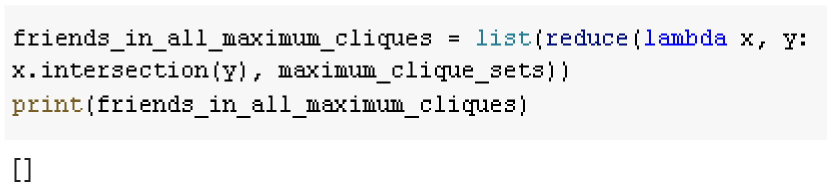
['.net', 'asp.net', 'c#', 'entity-framework', 'linq', 'wcf', 'sql-server'],

['.net', 'asp.net', 'c#', 'entity-framework', 'linq', 'wcf', 'wpf']

]

This means that in this network, we have 3 different maximum cliques in the graph, each of which consists of 7 nodes (technologies). This means that those 3 variations of 7 technologies all know one another.

After that, used the reduce function to find the nodes (technologies) that are common among all the maximum cliques. Remember that we have 3 maximum cliques in this network. The result showed that there is no technology in 3 maximum cliques.



```
friends_in_all_maximum_cliques = list(reduce(lambda x, y:
x.intersection(y), maximum_clique_sets))
print(friends_in_all_maximum_cliques)
```

```
[]
```

Figure 52 The empty result from finding technologies that are common among all the maximum cliques

Finally, I calculated the clustering coefficient, which is the fraction of the node’s neighbors that are also neighbors. Higher values of clustering coefficient indicate higher cliquishness which means highly connected segments of the network. There are 26 tag technologies (clustering coefficient=1) that these technologies connect to every tag technology in their sub-graph including: 'apache-spark': 1.0, 'bootstrap': 1.0, 'elasticsearch': 1.0, 'excel': 1.0, 'excel-vba': 1.0, 'flask': 1.0, 'hadoop': 1.0, 'iphone': 1.0, 'java-ee': 1.0, 'jsp': 1.0, 'less': 1.0, 'machine-learning': 1.0, 'oracle': 1.0, 'plsql': 1.0, 'ruby': 1.0, 'shell': 1.0, 'swift': 1.0, 'twitter-bootstrap': 1.0,'vb.net': 1.0, 'vba': 1.0, 'web-services': 1.0, 'wpf': 1.0, 'xcode': 1.0,

5) Node-Level Analysis

In this section, I concerned the node-level analysis to identify influential individuals technologies in this social network by using centrality measures including 1). Degree centrality is a measure of popularity, 2). Betweenness is based on the idea that a person is more important, 3). Closeness is a measure of reach, and 4). Eigenvector is a measure of related importance. The table 2 shows the top 10 technologies from each centrality measure:

Table 2 Top 10 technologies from each centrality measure.

Degree centrality	**betweenness_centrality**	**Closeness**	**Eigenvector**
jquery: 0.140 css: 0.123 c#: 0.123 asp.net: 0.114 angularjs: 0.114 javascript: 0.105 mysql: 0.096 html5: 0.088 php: 0.088 linux: 0.088	jquery: 0.256 linux: 0.208 mysql: 0.198 asp.net: 0.174 apache: 0.131 json: 0.123 angularjs: 0.123 rest: 0.114 python: 0.110 postgresql: 0.088	jquery: 0.290 mysql: 0.278 ajax: 0.259 css: 0.258 javascript: 0.257 angularjs: 0.257 apache: 0.255 php: 0.251 html: 0.247 asp.net: 0.247	jquery: 0.366 css: 0.339 javascript: 0.326 html5: 0.268 php: 0.265 angularjs: 0.265 sass: 0.252 mysql: 0.239 twitter-bootstrap: 0.207 html: 0.204

In conclusion, to explain the objective of how technologies are connected and how they are used together. **The Network-Level Analysis** provides an overview of network networks. This network has 3 subgraphs, the density shows that 1 technology usually connects to at least 3 technologies. **The Group-Level Analysis** reports that there are 89 cliques in this network with 3 different maximum cliques of 7 technologies. Finally, **the Node-Level Analysis** demonstrates that Jquery technology is the technology with the most influence in this social network among degree centrality, betweenness, closeness and Eigenvector centrality measures.

Link code :
https://drive.google.com/open?id=12DphEmKTarJgVo2MldGidYKeQqzEVVdV