CSO 135: Project 02
May 07, 2021

Introduction:

This project will create binary classifiers that will generate the sentiment-labels for new sentences in order to facilitate automation of the sentiment assessment process. Each binary will be assigned a sentiment value of 1 for positive and 0 for negative. The three datasets are 1) 2400 training examples in CSV x and y format from three websites yelp.com, amazon.com, and imdb.com, 2) the 600 testing examples from the three same websites, and 3) 400,000 vocabulary words from pre-trained embedding vectors. I will performed three different models on both data sets to help gauge sentiment analysis of positive vs. negative. This project is divided into two parts: in part one, I will experiment with three models by fine tuning hyper-parameters to finalize the best model, and in part two, I will further develop and test the model using word embeddings.

Part One: Classifying Review Sentiment with Bag-of-Words Features :

1.1 BoW Data Pre-processing and Feature Vectors:

On the feature vector side, I used two classes within sklearn, TfidVectorizer and TfidTransformer. Before I can apply above pre-processing methods, the data included 4510 unique words inside a training dataset. After my pre-processing methods, the data had 3417 unique words inside training (2400) and testing (600) datasets, which is a reduction by 1093 words. As of now I used the nltk's stopwords "dictionary=english" that are not relevant in this dataset, since common words will be penalized in TfidVectorizer and TfidTransformer classes. One important note here is the Term Frequency (tf), which is useful to measure how many times a word shows up in the list. The tf is then normalized by dividing the value by the total number of terms in the list. The Inverse document frequency (idf) will help me determine how important a single word can be by not taking into the account articles words, such as 'it', 'he', 'as', and 'a'. This can be calculated by counting the total number of word list divided by the number of list where term t is located. Then, tf-idf is a product of 'tf' and 'idf'.

Before I can get started on BoW features, I need to process the data first. I loaded the data in pandas and used nltk (Natural Language ToolKit) for pre-processing the data loaded. Detailed below are the steps I took to clean the data:

a) I converted all statements in reviews to lowercase method via processed.append(text.lower()) for easy processing.
b) I removed common words via stop_words=set(stopwords.words('english')) since such words are not relevant in the final conclusion of positive vs negative determination of reviews.
c) I used appos dictionary to remove all apostrophes and expand the word, such as aren't into are not. I then removed punctuation via stripped.append(words). These steps will reduce the complexity of the data.

d) Finally, I used PorterStemmer and gensim package to transfer all words into a statement to base form, which will remove inflection and determine part of the speech.

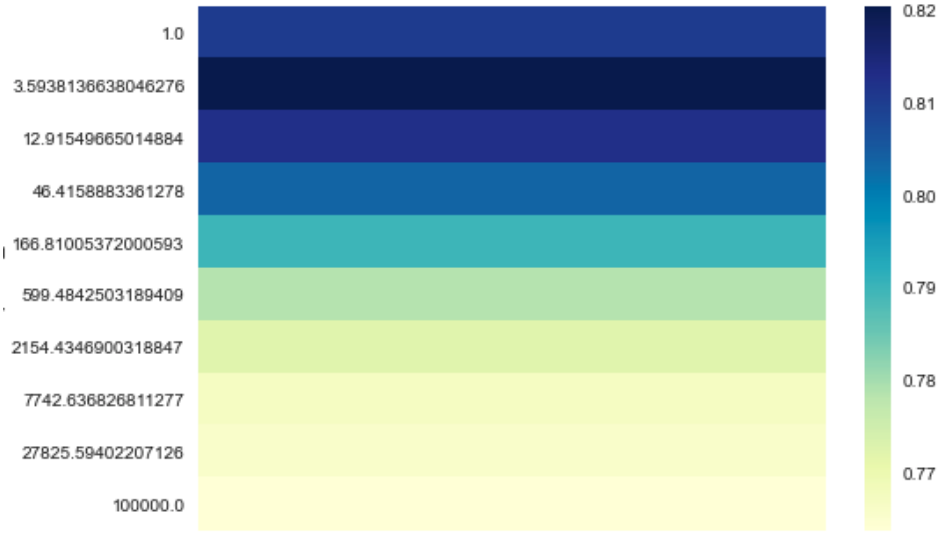
```
I'm very disappointed with my decision.
(2400, 4510)

['I', 'am', 'veri', 'disappoint', 'with', 'my', 'decis']
```

1.2 Logistic Regression Model on Training data for Bag-of-Words:

In this section, I ran Logistic Regression model via sklearn. On hyperparameters utilization to control such model, I set inverse penalty C range using np.logspace(0, 5, 10). This had 10 samples of 'C' starting from default 1.0 to 100,000. The final range was decided after experimenting with a few ranges, where the training data experienced reduced accuracy by other numbers compared to the numbers defined by the log space. I utilized both L1 (Ridge Method) and L2 (Lasso Method) regularization to identify any noticeable features or structures for overfitting. The L1 used an absolute value of magnitude of the coefficient for the penalty of the loss function, while L2 used a squared magnitude of the coefficient on penalty. Both regularization values for the hyperparameters resulted in 20 models but L2 gave the best results. With the help of GridSearch and 5 folds for best time efficiency, I cross-validated the values for the hyperparameters via clf = GridSearchCV(logreg, hyperparams, cv=5, verbose=0)

Figure 01: Logistic Regression Heatmap, x-axis as C-value and y-axis as L2 penalty



BEST SCORE: 0.8204
STANDARD DEVIATION FOR BEST SCORE: 0.0248
LEADERBOARD ERROR RATE: 0.1683
LEADERBOARD AUROC: 0.8317

In Figure 01, I have illustrated the best model for L2 regularization compare to L1, with C values that are closest to 1.0 having the best score. The highest accuracy of 0.8204 was around C=3.5938 with standard deviation of 0.0248. It seems the L2 model may have overfitted with

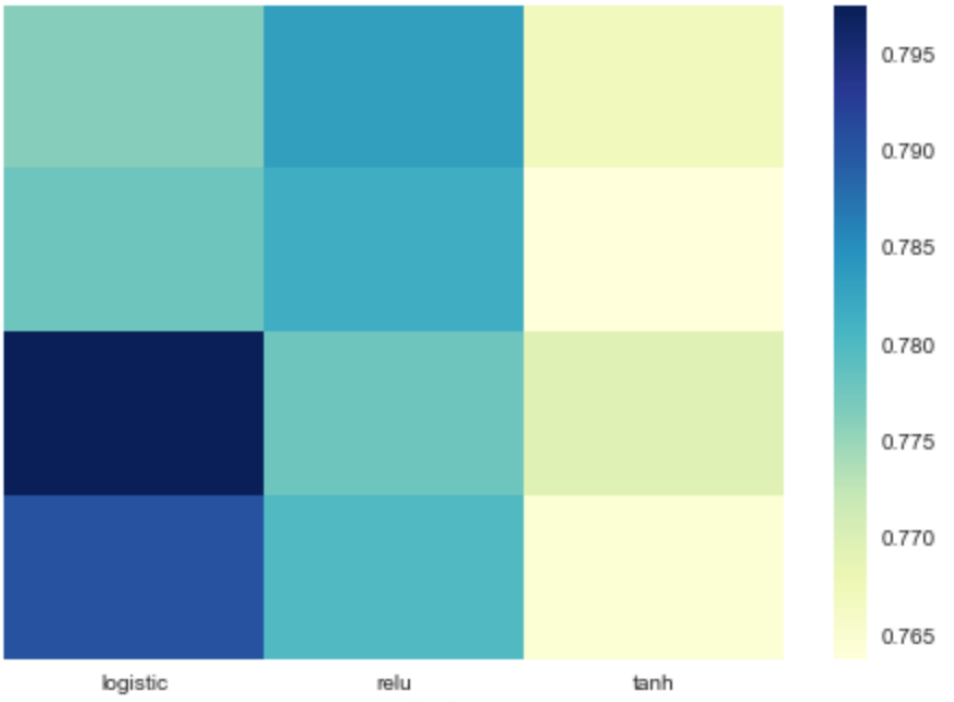
the C values that are close to 1.0. The model started performing worse at C value of 166.8100 and then onwards. The model had error rate of 0.1683 and an AUROC of 0.8317 calculated using Leaderboard.

1.3 Multilayer Perceptron Model on Training data with Bag-Of-Words:

In this section, I ran Multilayer Perceptron (MLP) model via sklearn. Then, I used both the activation functions and number of hidden layers with composition from neurons, as my hyperparameters. The three activation functions I used were 'Logistic', 'reLU', and 'Tanh'. On hidden layers, I chose a range of 1 to 3 after experimenting since 1 or 2 hidden layers had the best optimization. For neurons per hidden layer, I experimented with few numbers and decided 'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,), (256,)].

The logistic activation function was represented by Sigmoid S-shaped curve but had an issue of slower convergence and uncentered gradient. The reLU Activation Function converged better and helped solve the vanishing gradient problem from the Logistic Activation Function. However, the issue with this function was that it created weight updates and did not activate neurons. The Tanh Activation Function was better at converging compare to Logistic Activation Function and worse compared to reLU Activation Function, while still having a vanishing gradient issue. The three activation functions resulted in 12 different models and an easily available grid to compare. With the help of GridSearch and 5 folds for best time efficiency, I cross-validated the values for the hyperparameters via clf = GridSearchCV(logreg, hyperparams, cv=5, verbose=0)

Figure 02: MLP Heatmap, x-axis as hidden layer and y-axis as activation function



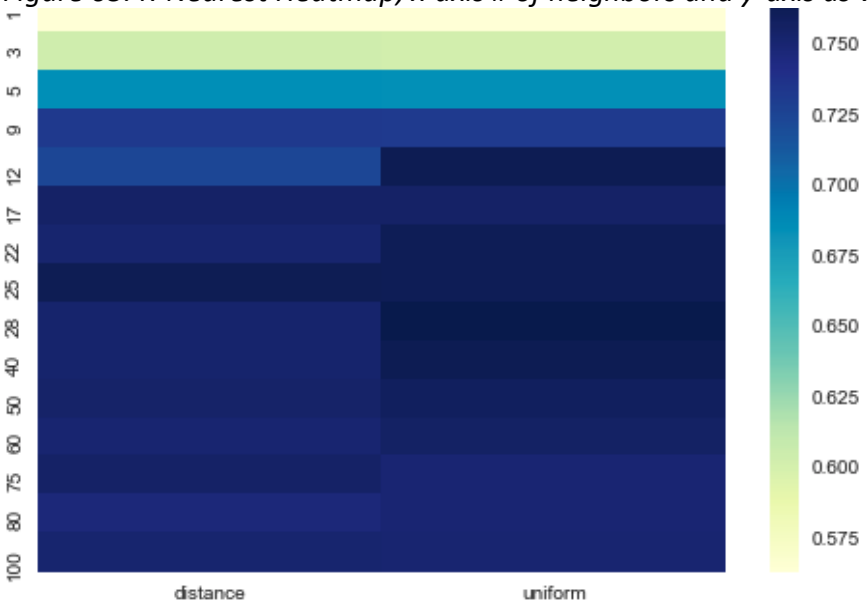
BEST SCORE: 0.7975
STANDARD DEVIATION FOR BEST SCORE: 0.0204
LEADERBOARD ERROR RATE: 0.1983
LEADERBOARD AUROC: 0.8824

In Figure 02, I have plotted all three activation functions with the above mentioned hyperparameters. The Logistic Activation Function has the best results with accuracy of 0.7975 and a standard deviation 0. 0204. The reLU and Tanh Activation Functions are lacking behind Logistics Activation Function when it comes to results per the figure heatmap. There is a possibility of that Logistics Activation Function is overfitting results. While the Tanh Activation Function is underfitting results. The model had error rate of 0.1983 and an AUROC of 0.8824 calculated using Leaderboard.

1.4 K-Nearest Neighbors on Training data with Bag-Of-Words:

In this section, I ran K-Nearest Neighbors clustering model with the two main hyperparameters in training data, optimizing the number of neighbors and the weight function. I chose k-number of neighbors from 1 to 100, where 5 is usually a default. I ran each iteration of the model to identify the optimal number of neighbors; thus I selected 'n_neighbors' : [1,3,5,9,12,17,22,28,25,40,50,60,75,80,100]. According to Figure 03, the ideal n_neighbors value is 17, after which the model had lower performance accuracy with increase in gradient. I used two weight functions – 'distance' and 'uniform' – as there wasn't much of a difference between the two by looking at the Figure 03. The uniform weight function had all the points clustered equally while the distance weight function had all the points inversely of their distance. This resulted in 28 total models. With the help of GridSearch and 5 folds for best time efficiency, I cross-validated the values for the hyperparameters via clf = GridSearchCV(logreg, hyperparams, cv=5, verbose=0)

Figure 03: K-Nearest Heatmap, x-axis # of neighbors and y-axis as weights function



BEST SCORE: 0.7675
STANDARD DEVIATION FOR BEST SCORE: 0.0204
LEADERBOARD ERROR RATE: 0.2617
LEADERBOARD AUROC: 0.8478

In Figure 03, I have plotted the two weight functions with all n values. I achieve the best results when number of neighbors were 17. The best accuracy of the model was at 0.7675 and a standard deviation of 0.0204. The model had error rate of 0.2617 and an AUROC of 0.8478 calculated using Leaderboard.

1.5 Best Results on Training data with Bag-Of-Words:

From the above discussed models, Logistic Regression has the best results out of all three (see a summary of results in Table 1). The other two had the poor results with K-Nearest Neighbors among the worst out of the two. I believe that KNN and MLP could have been better if I had a chance to fine tune their parameter and remove any underfitting issues. The Logistic Regression success in showing me best results were due to feature tuning where it had attributes unrelated to the output variable. In the start, I also removed the closely related attributes from the input set and using stopwords that helped normalize the input data into feature vectors using TFIDF.

Table 01: Results across all three models

Classifier	Score	Std. Dev
Logistic Regression	0.82042	0.02484
MLP	0.79750	0.02043
K Nearest	0.76750	0.02035

To predict the positive review through my Logistic Regressions model, Amazon reviews performed the best when it comes to accuracy where the True Positive Rate was 0.9675 compare to the other two reviewer types (Yelp and IMDB) that had lower. At the same time, the model had False Positive Rate highest for IMDB reviews compared to other two (Amazon and Yelp). The True Negative Rate is the same for Amazon and Yelp at 0.9775, with IMDB had a lower at 0.97. Therefore, Amazon reviews performed better with Logistic Regression due to better usage of English language words to determine positive vs. negative reviews.

Table 02: Calculated Accuracy, TPR, TNR, FPR, FNR

Websites	Accuracy	True Positive	True Negative	False Positive	False Negative
Amazon	0.9725	0.9675	0.9775	0.0325	0.0225
Yelp	0.9625	0.9475	0.9775	0.0525	0.0225
IMDB	0.9525	0.9350	0.9700	0.0650	0.0300

1.6 Leaderboard Results on Best Classifier:

Similar to my results on the three discussed classifiers performed on the training data, Logistic Regression model gave the best results in Bag-of-Words Leaderboard. The Logistic Regression had the lowest error rate at 0.1683 among the three main models. While AUROC was 0.8317, which isn't the highest as we should expect. In conclusion, the training and test datasets had the best performance on Logistic Regression. One final assumption to make here is that there is a possibility that Logistic Regression may have overfit on the training data, giving me better results on the testing data.

Table 03: Results across all three models and Leaderboard on Training data

Classifier	Score	Std. Dev	Error Rate	AUROC
Logistic Regression	0.82042	0.02484	0.16833	0.83167
MLP	0.79750	0.02043	0.19833	0.88238
K Nearest	0.76750	0.02035	0.26167	0.84773

Part Two: Classifying Review Sentiment with Word Embeddings:

2.1 BoW Data Pre-processing and Pre-trained Embedded Vectors GloVe:

I processed the data similar to Part One. Detailed below are the steps I took to clean the data:

a) I converted all statements in reviews to lowercase method via processed.append(text.lower()) for easy processing.
b) I removed common words via stop_words=set(stopwords.words('english')) since such words are not relevant in the final conclusion of positive vs negative determination of reviews.
c) I used appos dictionary to remove all apostrophes and expand the word, such as aren't into are not. I then removed punctuation via stripped.append(words). These steps will reduce the complexity of the data.
d) Finally, I used PorterStemmer and gensim package to transfer all words into a statement to base form, which will remove inflection and determine part of the speech.

By using the given 50 dimensions feature vector for each word in GloVE.txt, I calculated the average of all the words within the reviews to have a vector of reviews. This way, I can have a good standard measure of input data. While other options, such as sum and concatenate, were available to me. They would have made the data more complex and create discrepancies in words' length. Afterwards, I used TfidfVectorizer inside sklearn to calculate TFIDF weight by each feature vector and the TFIDF weight. This helped me conclude 50 dimensions on test and training data, compare to 3417 in Part One due to GloVE.txt.

```
60 print(x_test.shape)
61 print(x_train.shape)
62
```

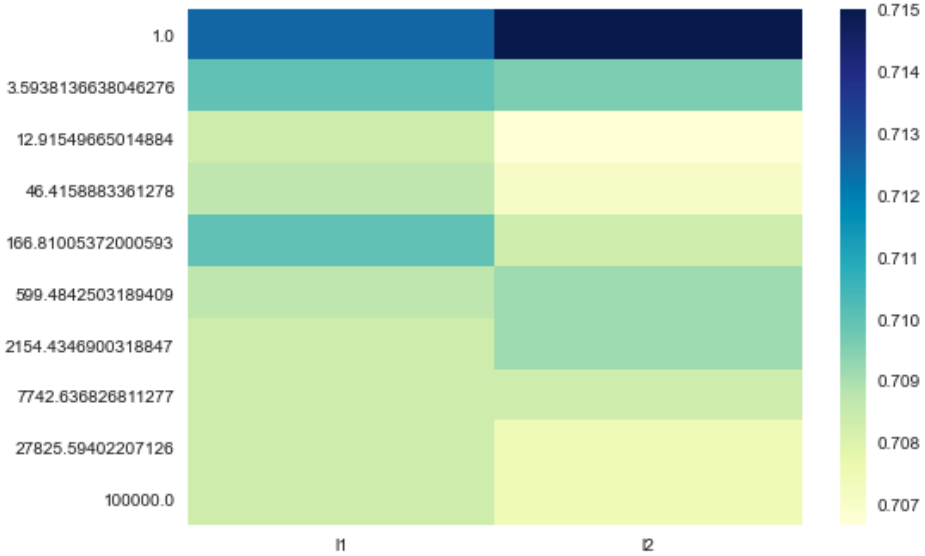
```
<ipython-input-8-7a338c18b
  return sent_vec / np.sqr
```

```
(600, 50)
(2400, 50)
```

2.2 Logistic Regression Model on Training data with Word Embeddings:

In this section, I ran Logistic Regression model via sklearn. The two hyperparameters were inverse penalty C and type of regularization (L1 vs L2). Similar to Part One, I used inverse penalty C (0, 5, 10) with 12 samples of C from 1.0 to 100,000. Here I ran both L1 and L2 regularization to show the difference between the two with total of 20 different models, compared to L2 of 10 models in Part One. In this section, there is a better visualization of accuracy in the model with parameter and inverse penalties, where accuracy is more closely tied together than in Part One. With the help of GridSearch and 5 folds for best time efficiency, I cross-validated the values for the hyperparameters via clf = GridSearchCV(logreg, hyperparams, cv=5, verbose=0).

Figure 04: Logistic Regression Heatmap, x-axis as C value and y-axis as penalty



BEST SCORE: 0.7150
STANDARD DEVIATION FOR BEST SCORE: 0.01923
LEADERBOARD ERROR RATE: 0.2933
LEADERBOARD AUROC: 0.7787

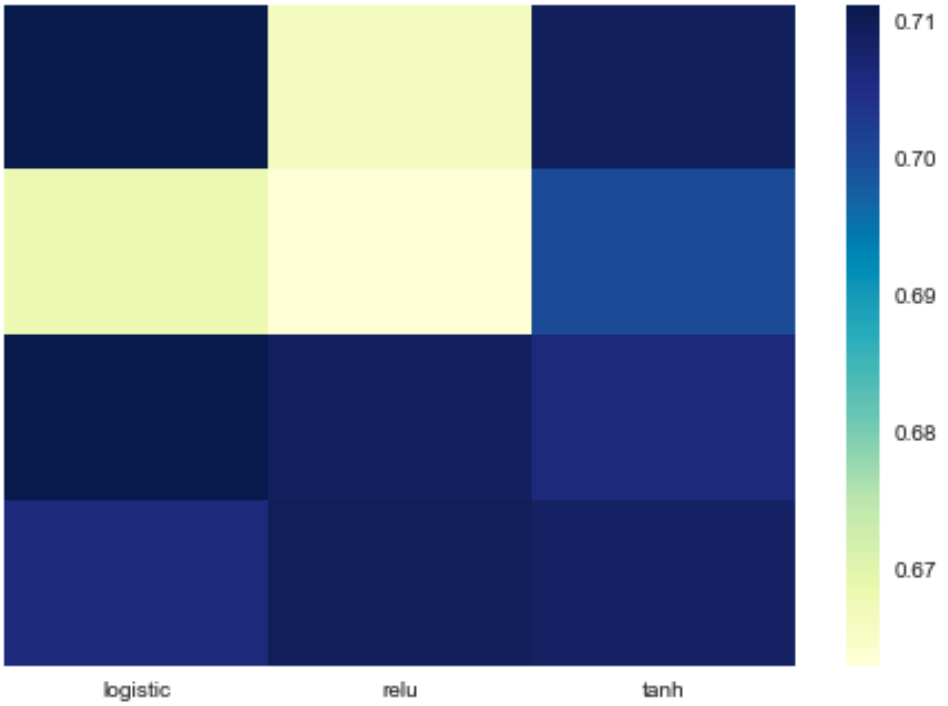
In Figure 04, I have illustrated the best model for L1 and L2 regularizations. The highest accuracy of 0.7150 was at around C=1 on L2 and a standard deviation of 0.0193. It seems the L2 model may have overfit with C values that are close to 1.0 due to the narrow range of accuracy results. In comparison to Part One, this current result underperforms but can also imply underfitting due to improper normalization of inputs. The model had error rate of 0.1683 and an AUROC of 0.8317 calculated using Leaderboard, both performing worse than in Part One.

2.3 Multilayer Perceptron Model on Training data with Word Embeddings:

In this section, I ran Multilayer Perceptron (MLP) model via sklearn. Then, I used two hyperparameters, the activation functions and number of hidden layers with composition from neurons. The three activation functions I used were similar to Part One: 'Logistic', 'reLU', and 'Tanh'. On hidden layers, I chose range of 1 to 3 similar to Part One to get most optimized

results from hidden layers. For neurons per hidden layer, I experimented with few and eventually landed on 'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,), (256,)]. With the help of GridSearch and 5 folds for best time efficiency, I cross-validated the values for the hyperparameters via clf = GridSearchCV(logreg, hyperparams, cv=5, verbose=0).

Figure 05: MLP Heatmap, x-axis as hidden layer and y-axis as activation function



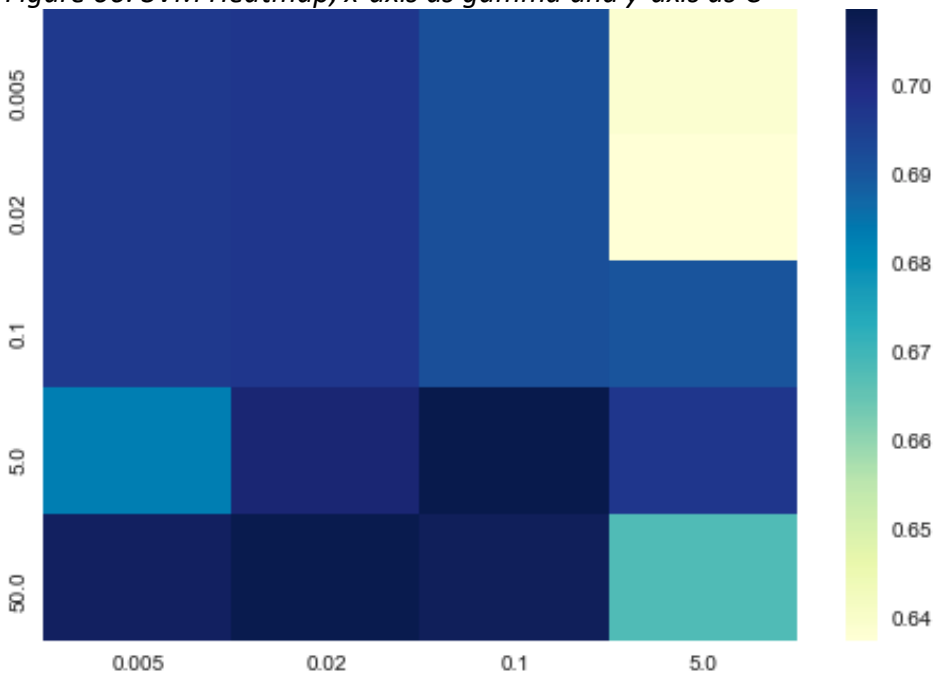
BEST SCORE: 0.7112
STANDARD DEVIATION FOR BEST SCORE: 0.0229
LEADERBOARD ERROR RATE: 0.3000
LEADERBOARD AUROC: 0. 7752

In Figure 05, I have plotted all three activation functions with the above mentioned hyperparameters. The Logistic Activation Function showed the best results, similar to Part One with Bag of Words, with accuracy of 0.7112 and standard deviation 0.0229. However, Word Embeddings perform poorly in Logistic Activation Function, compared to Part One if we consider only the accuracy range. This means that I could have underfit the data in this part of Logistic Activation Function in comparison to Part One, resulting in poor performance of training and testing data. The reLU and Tanh Activation Functions had worse results per the figure heatmap. My initial thoughts were that there is a possibility of overfitting in Logistics Activation Function and Tanh Activation Function has underfitting results proven by Leaderboard scoring. The model had error rate of 0.3000 and an AUROC of 0.7752 calculated using Leaderboard.

2.4 Support Vector Machines on Training data with Word Embeddings:

In this section, I ran Support Vector Machines (SVM) model via sklearn. I used the inverse penalty value C and gamma values for hyperparameters. On the inverse penalty value C, I decided to utilize five magnitudes *[0.005, 0.02, 0.1, 5, 50]* to understand which inverse penalties, lower and higher, serves me better. Since the SVM model is using non-linear kernel, I chose gamma values to be *'gamma':[0.005, 0.02, 0.1, 5]*, which helped me identify if the higher gramma fits the data compared to lower gamma. With the help of GridSearch and 5 folds for best time efficiency, I cross-validated the values for the hyperparameter via clf = GridSearchCV(logreg, hyperparams, cv=5, verbose=0)

Figure 06: SVM Heatmap, x-axis as gamma and y-axis as C



BEST SCORE: 0.7088
STANDARD DEVIATION FOR BEST SCORE: 0.0229
LEADERBOARD ERROR RATE: 0.2983
LEADERBOARD AUROC: 0.7810

In Figure 06, the plot shows greater variety of results in performance compared to for SVM to MLP in Part One. It has the worst results compared to all the above models with best scored 0.7088 for accuracy and a standard deviation of 0.0229. The gamma values of 0.005 and 5 had lower accuracy on all values of C, resulting in underfitting on the training data. However, the best results are at gamma close to 1 and higher C value that helps fit the model better to the training set. The model had error rate of 0.2983 and an AUROC of 0.7810 calculated using Leaderboard.

2.5 Best Results on Training data with Word Embeddings:

From the above discussed models, Logistic Regression has the best results out of all three (see a summary of results in Table 4). The other two performed poorly with SVM among the worst out of the two. This is a similar conclusion to Part One, where Logistic Regression performed better than the other two models, KNN and MLP. However, accuracy range is smaller than Part One. This also means that in Part One, the three models had overfitting due to larger range in accuracy. Since in my processing step the data was calculated by averaging all the words within the reviews to have a vector of reviews, this could have caused the lack of optimization in terms of accuracy, along with the TF-IDF weighting. Defined by the leaderboard and its poor results, the TD-IDF weights could have overfit on testing data due to further adjustment from word-embedding algorithms. Next time, I should use an easy method for vectorization, i.e. using averages to get normalization to adjust for size of each feature but further TD-IDF might be too much here.

Table 4: Results across all three models on Training data

Classifier	Score	Std. Dev	Error Rate	AUROC
Logistic Regression	0.71500	0.01929	0.29330	0.77870
MLP	0.71125	0.02300	0.30000	0.77520
SVM	0.70875	0.02291	0.29833	0.78101

To predict the positive review through my Logistic Regressions model, Amazon reviews performed the best when it comes to accuracy where the True Positive Rate was the lowest 0.6925, compared to the other two reviewer types (Yelp and IMDB). At the same time, the model produced a lowest False Positive Rate for Yelp reviews at 0.2725, compared to other two (Amazon and IMDB). The True Negative Rate is also the lowest for Yelp, with a rate of 0.7050. In conclusion, Amazon reviews performed better with an accuracy of 0.7438, compared to other two; with Yelp performing the second best. I believe this was largely due to easily determination of positive vs. negative reviews on Amazon website. Amazon is more popular and has the best reaction to customers' reviews compared to the other two websites, which could be due to a better data set.

Table 05: Calculated Accuracy, TPR, TNR, FPR, FNR

Websites	Accuracy	True Positive	True Negative	False Positive	False Negative
Amazon	0.7438	0.6925	0.7950	0.3075	0.2050
Yelp	0.7163	0.7275	0.7050	0.2725	0.2950
IMDB	0.7163	0.6950	0.7375	0.3050	0.2625

2.6 Applying Best Model to Leaderboard:

Similar to my results on the three discussed classifiers performed on the training data and conclusion of Part One, Logistic Regression model gave the best results in Word Embedded Leaderboard. The Logistic Regression had the lowest error rate at 0.2933 among the three main models. While AUROC was 0.7787 for Logistic Regression, SVM had the highest AUROC of 0.7810. In Table 06, Leaderboard results best represent the scoring since it matches up with how the training and test data had performed on all three classifier models.

I had anticipated that using word embeddings in training would increase the accuracy. However, word embeddings had the opposite effect in that the accuracy ended up being the worst for all three classifier models in Part Two. Furthermore, I had predicted that the 50 not so abundant feature vectors in Part Two would produce better results in terms of higher accuracy and lower errors, compared to 3417-dimensional feature vectors in Part One. However, the vectorization of features reduced reviews sentiments with smaller inputs.

In conclusion, the training and test datasets had the best performance using Logistic Regression. I observed that my initial steps of data processing may have underfit the data in Part Two since the results worsened compared to Part One. Instead of using average function on 50 dimensions feature vector, I could have experimented with sum, concatenate or other functions to better process the data.

Table 06: Results across all three models and Leaderboard on Training data

Classifier	Score	Std. Dev	Error Rate	AUROC
Logistic Regression	0.71500	0.01929	0.29330	0.77870
MLP	0.71125	0.02300	0.30000	0.77520
SVM	0.70875	0.02291	0.29833	0.78101