Purpose: explore a variety of classifiers, applied to some real-world data-sets for natural language processing
To do this, we use datasets consisting of reviews from yelp.com, amazon.com and imdb.com, where each review has a classification based on sentiment.

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

1.1: Preprocessing and Bag-Of-Words:

After the data was read in using pandas dataframes, I cleaned the reviews before starting to initialize my bag of words. Before cleaning, the training set had 5232 different words. To clean each review, I took each review and got rid of punctuation, replacing each instance of punctuation with a space. I also made every letter lowercase. This will hopefully help improve the performance of the bag-of-words features. I then got rid of stop words, which are common words that don't influence the sentiment of the line. After cleaning, the size of the vocabulary shrunk to 3417 words.

After cleaning the reviews, I created my bag of words. I initially did this in the default manner, counting the number of instances of every single word in order to create the vocabulary vector. In part one of this report, models that use this method will be under sub-section A.

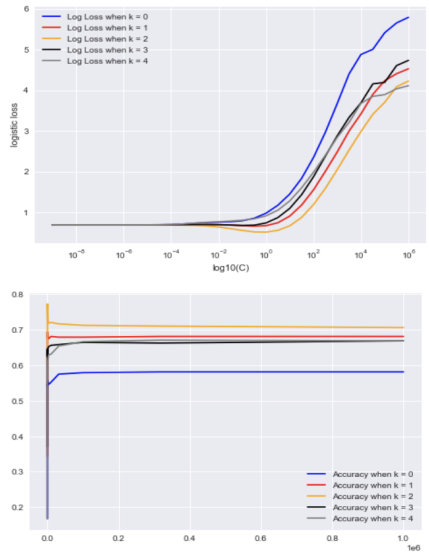
After attempting this straightforward method, I wasn't satisfied with my models' accuracy so I tried the 'term frequency inverse document frequency' method described in class to construct the bag of words. This vectorization method is used to determine how important every word is to a set of words. This is done by taking the log of the number of lines divided by the number of lines with a given word. In part one of this report, models that use this method will be under sub-section B.

In the sections below, I say that I "use 5-fold cross validation". This was done because of the lack of testing output data and to test model performance on various testing and training data. My method for doing this cross validation was the same method described in class and in past assignments; I split both the input and output data into 5 folds, where 4 of the 5 are used as training data and the remaining 1 is used for testing. Through cross validation, each of the folds is used as a testing set once where the rest are used for training. For this project, I utilized the sklearn library's Kfold split functions to perform this cross validation. On the graphs below, each line represents one variation of the testing and training data where k represents the fold of the testing fold. The listed standard deviations are of model accuracy across the variation in hyperparameters as well as through cross-validation.

1.2: Basic Logistic Regression Model:

1.2.A: Similar to past assignments, I trained my basic logistic regression model around penalty strength. Penalty strength, the parameter C, is used to increase the accuracy of models on new data by decreasing the coefficient weights at a certain rate. As we've done in the past, I utilized the numpy logspace to test 31 different penalty strength values and recorded the accuracy and log-loss of each model. All of this was done using 5-fold cross validation. The results are as follows:

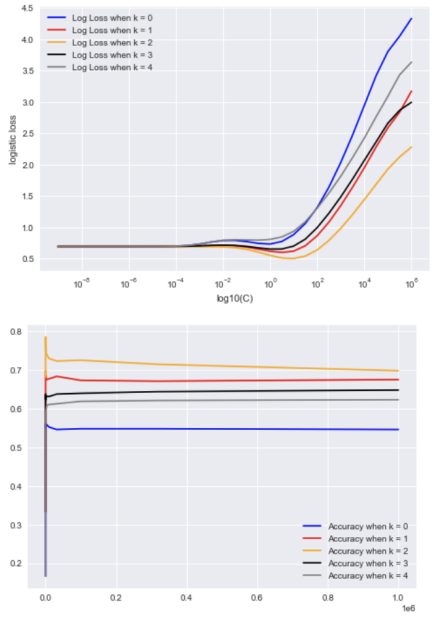
```
Standard deviation of models: 0.15007577852859932
```




```
Best C-value for LR: 0.0000000
Log-loss at best C-value: 0.5148
Accuracy of model using best C-value: 0.7729167
```

1.2.B: For my TFIDF bag-of-words I followed the same steps described above to find the best penalty strength and max iterations for my model. The results are as follows:

```
Standard deviation of models: 0.14632646060890242
```




```
Best C-value for LR: 0.0000000
Log-loss at best C-value: 0.5008
Accuracy of model using best C-value: 0.7854167
```

Discussion of Part 1.2:

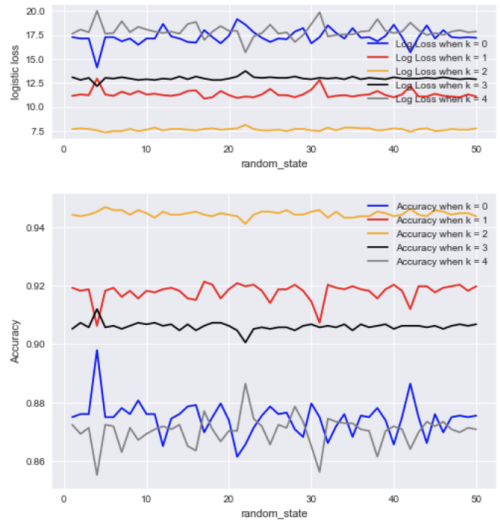
From these results, we can see that the straightforward bag-of-words and the TFIDF bag-of-words have similar performance, with the basic bag-of-words having slightly higher accuracy and lower log-loss. The two also have similar, high standard deviations across their models, telling us that variation in accuracy alone isn't enough to distinguish which is better. The high standard deviation tells us that this model's accuracy isn't very stable. However, the accuracy of these models isn't enough to say that overfitting is occurring. Moreover, the parallel trends in log-loss and accuracy demonstrate that the variation in accuracy isn't due to the cross-validation.

1.3: Neural Network Model:

1.3.A: Similar to assignment 4, I trained 2 different MLP models using 2 activation functions, 'logistic' and 'relu'. These activation functions serve the purpose of deciding the output of every neuron in the neural network. As we did in assignment 4, I changed the random_state parameter by increments of one, recording the log-loss and accuracy of each. The random_state parameter is used to generate the initial weights and bias. This was done for both activation functions and using 5-fold cross validation. Below, you can see how the log-loss and accuracy of the models changes based on the random_state parameter.

Logistic:

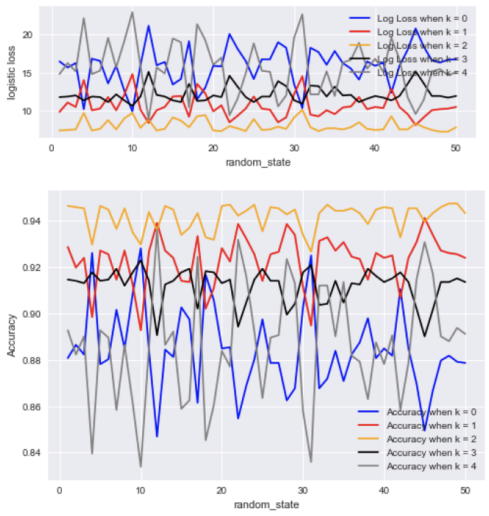
```
Standard deviation of models: 0.0032232432106539684
```




```
Best random_state for MLP: 3.0000000
Log-loss at best random_state: 7.3396
Accuracy under the best random_state: 0.9469
```

ReLu:

```
Standard deviation of models: 0.013415744861766055
```

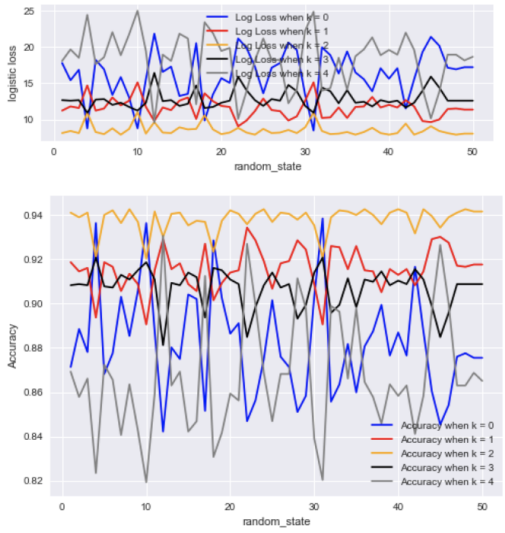



```
Best random_state for MLP: 3.0000000
Log-loss at best random_state: 7.2676
Accuracy under the best random_state: 0.9474
```

1.3.B: For my TFIDF bag-of-words, I followed the same procedure described above in order to find the best value for the random_state parameter. The results are as follows:

Logistic:

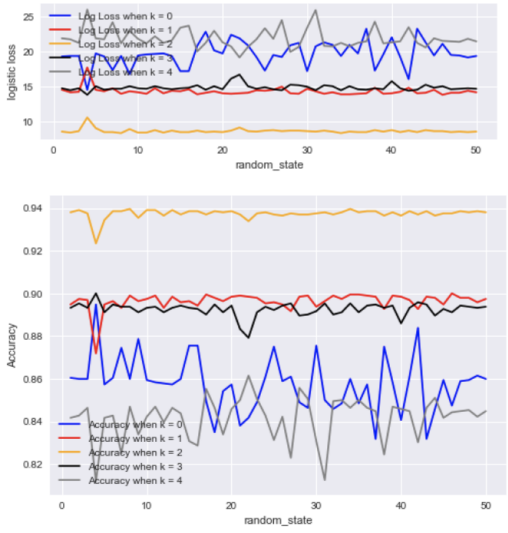
```
Standard deviation of models: 0.014944164030228363
```




```
Best random_state for MLP: 3.0000000
Log-loss at best random_state: 7.9152
Accuracy under the best random_state: 0.9427
```

ReLu:

```
Standard deviation of models: 0.006548116236843003
```




```
Best random_state for MLP: 3.0000000
Log-loss at best random_state: 8.3469
Accuracy under the best random_state: 0.9396
```

Discussion of Part 1.3:

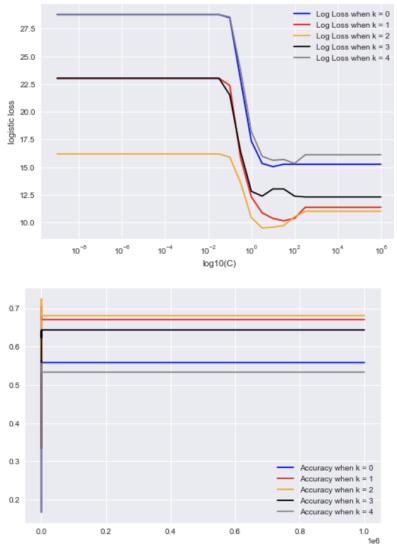
From these results, we can see that the straightforward bag-of-words had better overall performance than the TFIDF bag-of-words. This is due to the fact that the MLP model with the straightforward bag-of-words had higher accuracy and lower log-loss, with similar variation in accuracy. There is a small difference in the performance of the two activation functions with both bags-of-words, but it isn't clear which activation function provides the better performance. For the straightforward bag-of-words, the models that used the logistic activation have lower accuracy and higher log-loss. However, for the TFIDF bag-of-words, the models that used the relu activation function had higher accuracy but also higher log-loss. The wide variation in the accuracy of the models themselves is likely a result of the random_state value having seemingly random variation, but testing with cross-validation for the strongest value of random_state is still valuable in driving towards the most accurate model. It was also interesting that every variation of the MLP model resulted in the same optimal random state, 3.0. This optimal model provides an accuracy of 0.9474 and log-loss of 7.2676.

The high accuracy is a sign of overfitting to the training data. It is unlikely that the optimal neural network model described above will perform as well on new data, such as in the leaderboard.

1.4: Support Vector Machine Model:

1.4.A: To build my SVM model, I varied the parameter C, penalty strength. As stated above, C is used to increase the accuracy of models on new data by decreasing the coefficient weights at a certain rate. I did this in a similar fashion to section 1.2, initializing an array of 31 evenly spaced C-values between -9 and 6. I then incremented through this array, building an SVM model for each penalty strength and recording the log-loss and accuracy of the model. This was done using 5-fold cross validation in order to have a testing set of outputs. The results are as follows:

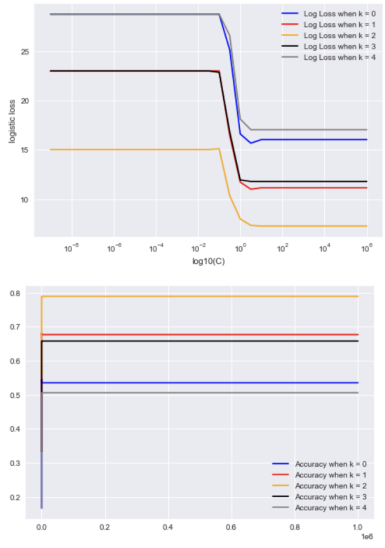
```
Standard deviation of models: 0.15516914681316804
```




```
Best C-value for LR: 0.0000000
Log-loss at best C-value: 9.4982
Accuracy of model using best C-value: 0.7250000
```

1.4.B: To build my SVM model around the TFIDF bag of words, I tested the penalty strength parameter in a similar fashion to sections 1.2 and 1.4.A. After initializing the array of 31 evenly spaced values between -9 and 6, I created a model using each, recording log-loss and accuracy. This was also done using 5-fold cross validation.

```
Standard deviation of models: 0.15851537555280992
```




```
Best C-value for LR: 0.0000000
Log-loss at best C-value: 7.2676
Accuracy of model using best C-value: 0.7895833
```

Discussion of Part 1.4:
From these results we can see that SVM models that used the TFIDF bag-of-words perform significantly better, in terms of both log-loss and accuracy. It is clear that the optimal penalty strength, C, is 0.0. This optimal model provides an accuracy of 0.7895833 and log-loss of 7.2676. While the variance of the accuracy is high, and there are parallel trends between folds, so it can be deduced that the variation in accuracy is due to the variance in the C-values. This is unfortunate, as it leads us to believe that this model might not perform as well as it does here on new data.

Discussion of Part One:

The model with the highest accuracy is the neural network model that uses the straightforward bag-of-words and logistic activation function. For this model, the optimal random_state value was 3.0. This could be due to the backpropagation ability of neural networks, allowing models to shift weights based on results that come further down the line. This model shows unreasonably high accuracy, which is a strong sign of overfitting to the training data. This leads me to believe that this might not be the optimal model to submit to the leaderboard. There was no consistency in which form of bag-of-words performed better, but generally the models that used the TFIDF bag-of-words performed better overall and had smaller variance with cross-validation.
In terms of cost, building the SVM models took around ten minutes, as well as about 10% of my computer's battery. On the other hand, building the logistic regression and MLP models took less than a minute each. If I were building an application that built SVMs to classify data such as the data provided, I would take this into account. For the purpose of this project, however, the time taken to build the optimal SVM is not important.
Leaderboard Performance:

Unfortunately, my optimal neural network model with a random_state of 3.0 performed significantly worse compared to the performance on the testing and training data used to build the model, achieving an abysmal error rate of 0.50167 and an AUROC of 0.49833. However, upon submitting my optimal SVM model, I received an error rate of 0.19 and an AUROC of 0.81. This is in contrast to the error rate from testing and training data, which was about 0.21.

This could be expected, as this model did not demonstrate overfitting to the training data, thereby should perform better on new data than the optimal neural network model. This backs my previous claim that the optimal neural network model demonstrates overfitting to the training data. One thing I found incredibly strange is that when I submitted my other models to the leaderboard, as well as changed all of the parameters within these models multiple times, all of them have the same error rate and AUROC within a range of around .05. Again, I'm not exactly sure why this is.

Part Two: Classifying Review Sentiment with Word Embeddings

2.1: Preprocessing and Vectorization:

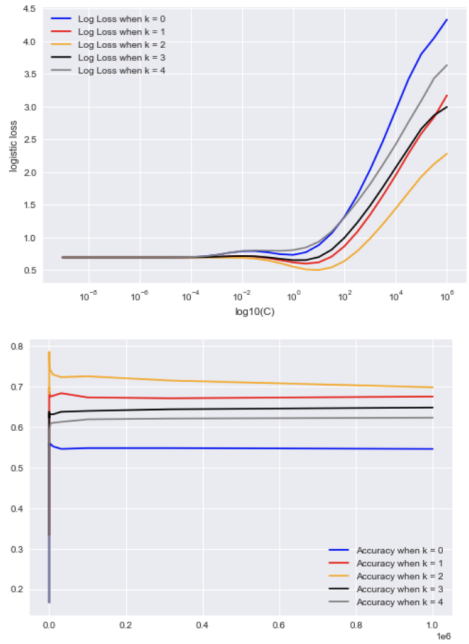
First, the new data was loaded into a pandas dataframe before being preprocessed, as described in the given file load_word_embeddings.py. This new data is in the form of word embedding vectors, which try to describe the meaning and context. To preprocess the old data, I used the same procedure described in part 1: converted all letters to lowercase, got rid of punctuation, then got rid of stop words. However, unlike Part One I standardized the new data by taking the average of the 50-dimension feature vector provided by GloVE word embeddings for each word. This is part of an effort to widen the scope of these vectors, as each word will have different meaning in different contexts: different review sources, different review subjects, etc. I did not ignore rare or common words.

Since the TFIDF bag-of-words performed better in Part One, I've decided to only use this form of vectorization for Part Two. Additionally, as described in Part 1.1, when I mention that I used 5-fold cross validation on a model it was done in the same exact manner as described before. The listed standard deviations are of the models across the variation in hyperparameters as well as through cross-validation. Each line in the graphs below still represents the performance of the model where k is the index of the testing fold.

2.2: Basic Logistic Regression Model:

Similar to part 1.2, I trained my basic logistic regression model around penalty strength. As stated above, C is used to increase the accuracy of models on new data by decreasing the coefficient weights at a certain rate. I utilized the numpy logspace to test the same 31 different penalty strength values as before and recorded each the accuracy and log-loss of each model. All of this was done using 5-fold cross validation. The results are as follows:

```
Standard deviation of models: 0.14632646060890242
```




```
Best C-value for LR: 0.0000000
Log-loss at best C-value: 0.5008
Accuracy of model using best C-value: 0.7854167
```

Discussion of Part 2.2:

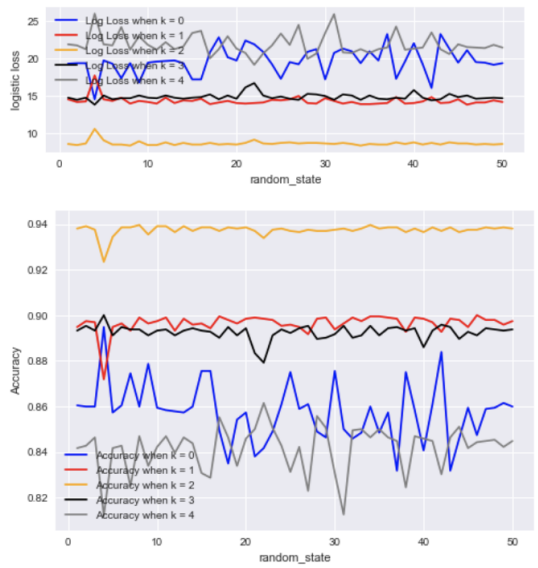
From these results we can see that for the data consisting of these new features, the optimal penalty strength when building a logistic regression classifier is 0.0, which achieves an accuracy of 0.785. However, through cross validation and varying the penalty strength we have a high standard deviation of 0.146. This isn't too concerning, as the parallel trends through cross-validation demonstrate that the variation comes from the variation in the penalty strength. Moreover, there is no evidence of overfitting as the maximum accuracy achieved isn't unreasonably high.

2.3: Neural Network Model:

Similar to Part 1.3, I trained 2 different MLP models using 2 activation functions, 'logistic' and 'relu'. By increments of one, I changed the random_state parameter, recording the log-loss and accuracy of each. This was done for both activation functions. Below, you can see how the log-loss and accuracy of the models changes based on the random_state parameter.

Logistic:

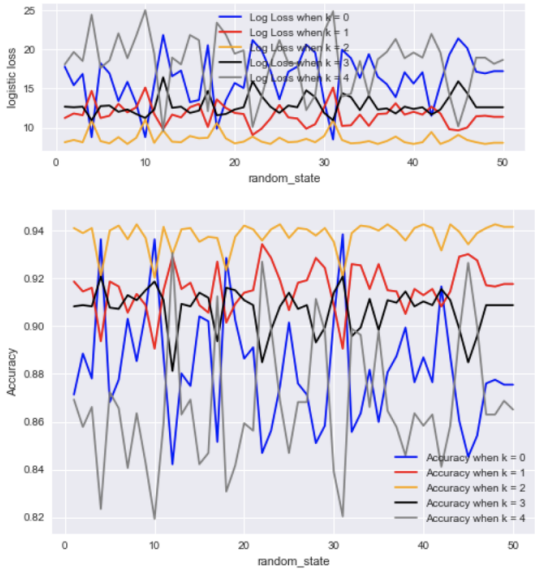
```
Standard deviation of models: 0.006548116236843003
```




```
Best random_state for MLP: 3.0000000
Log-loss at best random_state: 8.3469
Accuracy under the best random_state: 0.9396
```

Relu:

```
Standard deviation of models: 0.014944164030228363
```




```
Best random_state for MLP: 3.0000000
Log-loss at best random_state: 7.9152
Accuracy under the best random_state: 0.9427
```

Discussion of Part 2.3:

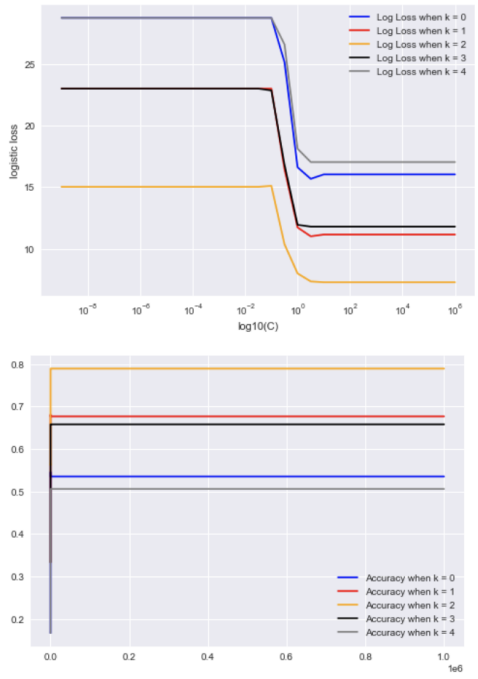
From these results we can see that for this newly augmented data, under a neural network model, the relu activation function achieves a lower log-loss and higher accuracy with an optimal penalty strength of 3.0. Under these parameters, the accuracy of the model is 0.9427 and the log-loss is 7.9152. The lack of parallel trends between folds is concerning, as we cannot deduce that the variation in accuracy is due to the variation in the random_state parameter. However, the variation in accuracy isn't high enough to override the fact that the optimal model with the Relu activation function achieves a higher accuracy and lower log-loss. Moreover, the optimal neural network models under both activation functions achieve higher accuracy but significantly higher log-loss than the optimal logistic regression model.

The high accuracy is a sign of overfitting to the training data. This is unfortunate, as this accuracy is significantly higher than the accuracy of the other models. It is unlikely that the optimal SVM model will perform as well on new data, such as in the leaderboard.

2.4: Support Vector Machine:

Similar to section 1.3, to build my SVM model I tested the parameter C, penalty strength. As stated above, C is used to increase the accuracy of models on new data by decreasing the coefficient weights at a certain rate. I did this in a similar fashion to section 1.2, initializing an array of 31 evenly spaced C-values between -9 and 6. I then increment through this array, building an SVM model for each penalty strength and recording the log-loss and accuracy of the model. This was done using 5-fold cross validation in order to have a testing set of outputs. The results are as follows:

```
Standard deviation of models: 0.15851537555280992
```




```
Best C-value for SVM: 0.0000000
Log-loss at best C-value: 7.2676
Accuracy of model using best C-value: 0.7895833
```

Discussion of Part 2.4:

These results tell us that the optimal penalty strength, C, is 0.0 for building an SVM model using the newly augmented TFIDF bag-of-words. This optimal model provides an accuracy of .78958 and log-loss of 7.2676. This is in line with the outcome of section 1.4, as we get the same optimal penalty strength. Moreover, the accuracy under the optimal penalty strength isn't high enough to signify overfitting. While the variation in accuracy is high, the parallel trends between the folds tells us that the variation is due to the changing penalty strength across the models. This is not concerning, as we only care about the optimal value not how our model performs across all tested values of C.

Discussion of Part Two:

The model with the highest accuracy with this new augmented data is the neural network model that uses the relu activation function. This model had an accuracy of 0.9427 and a log-loss of 7.9152. For this model, the optimal random_state value was 3.0. Oddly, this is almost the same exact result as Part One. This could be due to the backpropagation ability of neural networks, allowing models to shift weights based on results that come further down the line.

Similar to Part One, this optimal model shows signs of overfitting due to the unreasonably high accuracy on training and testing data. Again, this leads me to believe that this model will not perform as strongly on new data as my other models.

In terms of cost, building the neural network models took around six minutes, as well as about 20% of my computer's battery. On the other hand, building the logistic regression and SVM models took around two minutes each. If I were building an application that built neural network models to classify data such as the data provided, I would take this into account. However, for the purpose of this project, the time taken to build the optimal neural network is not important.

Leaderboard Prediction:

Although the optimal neural network model had the highest accuracy by a wide margin, the unreasonably high accuracy leads me to believe that this model is overfit to the training data and will not perform well on the leaderboard. Upon uploading this model's predictions to the leaderboard my hunch was proven right, as this model achieved an error rate of 0.23 and an AUROC of 0.77. While this put me in a respectable tenth place, this is in line with the performance of my other models -- significantly worse performance than the model's performance on cross-validation. This is textbook proof of overfitting.

After seeing this, I uploaded my second best model's predictions. My second best model was my SVM with a random state of 0. This model didn't have high enough accuracy on testing data to signify overfitting, so I thought this model would perform significantly better than the optimal neural network.

RANK	SUBMISSION NAME	ERROR_RATE	AUROC
1	Charlie Day	0.15833	0.84167

I was proven correct, as this model placed first overall with an error rate of 0.15833 and an AUROC of 0.84167. This is slightly better performance than it had on cross-validation as described above. I am not surprised by this, as the lack of evidence of overfitting shows that this model would perform well on new data.