# **CAP 4770 Fall 2019 - Project Report**

# **Nicolas Caceda, Nathalie Crespo, Damian Morgan – DM 6**

## Data Description

The data set used for this project was a data set of Amazon Reviews on food, known as the Amazon Fine Foods Data. The data set contains the columns ProductId, UserId, HelpfulNessNumerator, HelpfulnessDenominator, Rating, Time, Summary, and Text. The majority of the calculations in Part A of the project were done with the reduced version of the data set. In Part B, an equal number of calculations were done between the reduced and full. For the purpose of this project (both Parts A and B), the only two columns used will be ‘Rating’ and ‘Text’. The ratings will act as the classes. The Amazon Review data set is not a very balanced data set. The majority of the reviews in the reduced data set are positive with relatively few of them being negative and even less being neutral. This imbalance becomes more evident as the full data set is analyzed. Figure 1 shows the count and respective ratios of each rating in both the reduced and full data sets. The only data not included from the data sets were the stop words, as they were filtered out. Additionally, any empty reviews and numbers were removed before the count of the classes were taken.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Total Reviews** | **Positive Reviews** | **Neutral Reviews** | **Negative Reviews** |
| **Reduced Data set** | 14,906 | 5,417  (~36.3%) | 2,790  (~18.7%) | 6,699  (~45.0%) |
| **Full Data set** | 500,000 | 82,037  (~16.4%) | 42,640  (~8.5%) | 375,323  (~75.1%) |

**Figure 1: Statistics for the Full and Reduced Data sets**

For our Bag of Words Experiments, all the positive, neutral, and negative reviews were used for both the reduced and full data sets. However, for our Word Embeddings and NN Experiments, we removed the neutral reviews from both the reduced and full data sets. It provided more accurate results and we found it was easier to use the neural networks with two classes instead of one.

## Bag of Words Experiments

For this section of the project, both the full and reduced data set were used with 5 different classifiers, Naïve Bayes, kNN (using cosine distance), Decision Trees, Random Forest, and SVM (Support Vector Machine). Accuracy, precision and recall were calculated for all classifiers. Our experiments are described in the paragraphs below.

To start, the reduced data set was first read using a Pandas csv reader while specifying the use of columns 5 and 8. These columns are for the ratings and the text respectively and were used to create a data frame. In order to properly be able to classify the data, the data was converted into a TF-IDF matrix using scikit-learn’s TfidfVectorizer. It is with the vectorizer that the stop words were taken out, as stop words can be passed into the vectorizer as a parameter to be filtered out.

Before the classifiers could be used, the data needed to be split. To split the testing and training data, the test\_train\_split() function was used. It was specified in the function that 80% of the data would be for training and 20% testing.

For the Naive Bayes classifier, an instance of MultinominalNB() was created. Once the data was fit, accuracy was computed by using the accuracy\_score() function and passing in the test labels and the predicted labels. Precision was calculated using the precision\_score() function the same parameters were passed in from accuracy. However, the Macro, Micro, and Weighted precision scores were calculated for each classifier as we wanted to account for the class imbalance that is in both the reduced and full data set. Same rules apply for recall except the function was recall\_score(). 10-Fold Cross Validation was done a similar way for each classifier. The cross\_val\_score() function was called and the classification model, the X and y test data, and cv=10 were passed in each time. This would come up with 10 accuracies and average them to produce a single score.

Our parameters for kNN and Random Forest were the same. However, experimentation was done with kNN. We predicted accuracies for multiple values of K, K 1 to 10. Precision, recall, 10 CV, and accuracy score were computed the same way as detailed above. For the decision tree classifiers, we did some experimentation with Gini and Entropy. We ran multiple models of SVM and found that SVM RBF had the highest accuracy followed by SVM linear. SVM polynomial was too inaccurate for the amount of time that it took. SVM RBF produced the highest accuracy of all the classifiers in the reduced set.

Once all the classifiers were computed for the reduced data set, we moved onto making the classifiers work with full data set. To start, Naïve Bayes was performed on the full data set using a TfidfVectorizer using partial fit. The full data was loaded into ‘chunks’ and fitted using each chunk. Each chunk had a test portion that was saved into two lists for data chunks and label chunks. The test chunks were used to create a list of chunks of data that contained predictions. Accuracy, precision, and recall where then computed using these lists and scores were returned for the full data set using the Naïve Bayes classifier.

Afterwards, another method for analyzing the full data was used, the HashingVectorizer. In this step, another Pandas data frame had to be made specifically for the HashingVectorizer, this vectorizer also filtered out stop words. Once the data frame was made, the vectorizer was created and the data was split like it was with the TfidfVectorizer. The classifiers were run again like earlier, but this time the scores returned were for the 500,000 records in the full data set. For all the classifiers, using the full data set resulted in much higher accuracy, precision, recall, and 10CV scores. It is also worth mentioning that the full data has more of a class imbalance than the reduced, so the weight of these numbers may be skewed so the weighted precision/recall score is of high importance since it accounts for class imbalance. Naive Bayes also had a relatively high accuracy compared to kNN/Decision Trees even with its short computation time. Most of the issues we encountered using the full data were with SVM and Random Forest using the HashingVectorizer. The code for those algorithms compile, but the computation time is extremely long. A more optimized approach of implementing those classifiers with large data sets would be ideal in the future.

All the results for both the full and reduced data sets are in the Results section.

## Word Embeddings and NN Experiments

For this section of the project, both the full and reduced data set were used in multiple Keras. A CNN and LSTM model were made, both had 3 variations for a total of 6 NN models. The full and reduced data set were run on all 6, and accuracy was recorded on every model. On some models, precision, recall, and F1 score were also recorded.

To start, a Pandas data frames were made for both the full and reduced data and the ‘Rating’ and ‘Text’ columns were loaded in. After the frames were created, we removed the neutral ratings from the data frames because it would provide more accurate results due to removing the large class imbalance described in section A. Additionally, it was proving difficult to get the neural networks running with three classes. Also stop words, numbers, special characters, and empty reviews were filtered out of the data frame to provide accurate results.

Once the data frames were made. The data was prepared for tokenization and word embedding. A tokenizer was made that tokenized each row in the respective data frame and padded the data. Then the get\_dummies() function was called on the rating column which converted the categorical data into indicator variables. This allowed it to be tokenized/embedded so it could begin to be passed into the neural network models. However, before it was passed in, the data was split 80/20% with the train\_test\_split() function. The split was made to be consistent with the BOW experiments so we could properly compare the two.

An LSTM model, known as LSTM1, was then constructed that contained 6 layers. It used the relu activation for the convolution layer and the softmax activation for the dense layer. Additionally, it used the Adam optimizer when the model was compiled. The LSTM1 model was then trained with a 30-batch size and 2 epochs, accuracy, loss, precision, recall, and F1 score were all recorded for this model. After LSTM1’s metrics were calculated, experimentation was performed with the number of Epochs. The plan was to find how many epochs an LSTM model could take before it began to overfit. After testing LSTM1 with 20 epochs, the loss versus epoch number was graphed and the best number of epochs was determined. Based on the tutorial provided by Real Python, the ‘best’ number of epochs is the number of epochs before the model begins to overfit. Once the number of epochs were determined to be 6 via the graph, LSTM1 was trained again with a higher number of epochs to see how accuracy increased. Shown in the results section.

Two more LSTM models were made (LSTM2 and LSTM3). These models had changes made to their layers to see how they affected the recorded metrics. For LSTM2 the one-dimensional convolution layer had a sigmoid activation, while the dense layer had a softplus activation. Additionally, the optimizer was changed from Adam to SGD. For LSTM3, the one-dimensional convolution layer was changed to a linear activation and the dense layer was changed to a softsign activation. The optimizer was changed to Adadelta. Both models were trained with 6 epochs and the same results were recorded.

After LSTM was properly experimented with, the first CNN model was made (CNN1). The CNN models had 9 layers. To keep the results fair between LSTM and CNN, CNN1 was given the same activation function and optimizer as LSTM1 was. We wanted to see how the results changed purely based on the higher number of layers in CNN.

Like LSTM1, after CNN1 had it’s metric recorded (same as LSTM1), CNN1’s epochs were experimented with. A test was run with 20 epochs and then the loss versus epoch number was plotted. The ‘best’ number of epochs for CNN was 5, which was lower than LSTM’s 6. Afterwards, CNN1 was tested, CNN2 and CNN3 were made. They had the same activations and optimizer as LSTM 2 and 3. Results were recorded and the differences between the number of layers between the two models (LSTM and CNN) are displayed in section D and discussed in section E. All second and third models (CNN2 & 3 LSTM2 & 3) had different batch sizes compared the first models.

## Results

Numerical results of all experiments are shown here. Discussion about the results are in section E.

**Table 1: BOW Results for the Reduced Data, all with the same 80-20 train-test split.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Classifier** | **Accuracy** | **Precision** | **Recall** | **10-Fold CV Accuracy** |
| Naive Bayes | 0.649 | Macro: 0.453  Micro: 0.649  Weighted: 0.544 | Macro:0.520  Micro: 0.649  Weighted: 0.649 | 0.56 (+/- 0.04) |
| kNN | K = 3: 0.529  K = 5: 0.540  K = 9: 0.558 | For 3NN:  Macro: 0.478  Micro: 0.529  Weighted: 0.504 | For 3NN:  Macro: 0.437  Micro: 0.529  Weighted: 0.529 | For 3NN:  0.49(+/-0.08)  For K 1-10NN:  0.52(+/-0.07) |
| Decision Trees | Gini/MaxDepth=0  0.568  Entropy/MaxDepth=3  0.500  Gini/MaxDepth=10  0.562 | Gini/MaxDepth=0  0.510  Entropy/MaxDepth=3  0.363  Gini/MaxDepth= 10  0.462 | Gini/MaxDepth=0  0.510  Entropy/MaxDepth=3  0.403  Gini/MaxDepth= 10  0.468 | Gini/MaxDepth=0  0.52(+/-0.04)  Entropy/MaxDepth=3  0.48(+/-0.02)  Gini/MaxDepth=10  0.53(+/-0.04) |
| Random Forest | 0.696 | Macro: 0.754  Micro: 0.696  Weighted: 0.727 | Macro: 0.579  Micro: 0.696  Weighted: 0.696 | 0.64 (+/-0.05) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Classifier** | **Accuracy** | **Precision** | **Recall** | **10-Fold CV Accuracy** |
| SVM | SVM Linear  0.703  SVM RBF  0.710  SVM Polynomial  0.571  SVM Sigmoid  0.698 | SVM Linear  Macro: 0.643  SVM RBF  Macro: 0.685  SVM Polynomial  Macro: 0.771  SVM Sigmoid  Macro: 0.636 | SVM Linear  Macro: 0.611  SVM RBF  Macro: 0.601  SVM Polynomial  Macro: 0.452  SVM Sigmoid  Macro: 0.601 | SVM Linear  0.69(+/-0.02)  SVM RBF  0.70(+/-0.02)  SVM Polynomial  0.55(+/-0.01)  SVM Sigmoid  0.68(+/-0.01) |

**Table 1 Continued:**

**Table 2: BOW Results for Full Data (Tfidf = TfidfVectorizer. HV = HashingVectorizer)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Classifier** | **Accuracy** | **Precision** | **Recall** | **10-Fold CV Accuracy** |
| Naïve Bayes | Tfidf:  0.79558  HV:  0.75185 | Tfidf:  0.76957  HV:  Macro: 0.55061  Micro: 0.75185  Weighted: 0.71159 | Tfidf:  0.43114  HV:  Macro: 0.33513  Micro: 0.7519  Weighted:0.7519 | Tfidf:  N/A  HV:  0.75(+/-0.00) |
| kNN | K = 3: 0.81724 | For 3NN:  Macro: 0.77354  Micro: 0.81724  Weighted: 0.80724 | For 3NN:  Macro: 0.55559  Micro: 0.81724  Weighted: 0.81724 | For 3NN:  0.76(+/-0.00) |
| Decision Trees | Gini/MaxDepth=0  0.84167  Entropy/MaxDepth=3  0.75715  Gini/MaxDepth=10  0.77198 | Gini/MaxDepth=0  Macro: 0.7107  Micro: 0.84167  Weighted: 0.83812  Entropy/MaxDepth=3  Macro: 0.43295  Micro: 0.75715  Weighted: 0.66037  Gini/MaxDepth= 10  Macro: 0.69319  Micro: 0.77198  Weighted: 0.74832 | Gini/MaxDepth=0  Macro: 0.694245  Micro: 0.84167  Weighted: 0.84167  Entropy/MaxDepth=3  Macro: 0.35555  Micro: 0.75715  Weighted: 0.75715  Gini/MaxDepth= 10  Macro: 0.38841  Micro: 0.77198  Weighted: 0.77198 | Gini/MaxDepth=0  0.79(+/-0.01)  Entropy/MaxDepth=3  0.76(+/-0.00)  Gini/MaxDepth=10  0.77(+/-.00) |

**Table 2 Continued: (Note: SVM Polynomial was not computed with the full data due to the over 24-hour compute time)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Classifier** | **Accuracy** | **Precision** | **Recall** | **10-Fold CV Accuracy** |
| Random Forest | 0.865910 | Macro: 0.939913  Micro: 0.86591  Weighted: 0.88315 | Macro: 0.63069  Micro: 0.86591  Weighted: 0.865 | 0.82(+/-0.00) |
| SVM | SVM Linear  0.85717  SVM RBF  0.710  SVM Sigmoid  0.77851 | SVM Linear  Macro: 0.7579  Micro: 0.85717  Weighted: 0.84015  SVM RBF  Macro: 0.877077  Micro: 0.89944  Weighted: 0.89744  SVM Sigmoid  Macro: 0.5574  Micro: 0.77851  Weighted: 0.75545 | SVM Linear  Macro: 0.60726  Micro: 0.8572  Weighted: 0.8572  SVM RBF  Macro: 0.71797  Micro: 0.8994  Weighted: 0.8994  SVM Sigmoid  Macro: 0.53371  Micro: 0.7785  Weighted: 0.7785 | SVM Linear  0.84(+/-0.00)  SVM RBF  0.86(+/-0.00)  SVM Sigmoid  0.79(+/-0.01) |

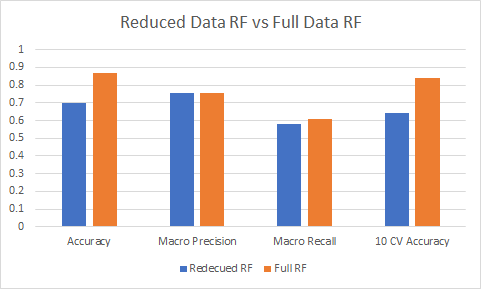
**Table 3: NN Results for Reduced Data. All experiments are with the same 80-20 train-test split**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Classifier** | **Best Performance Achieved Using** | **Accuracy** | **Precision** | **Recall** | **F1 Score** |
| LSTM1 | Conv1D = relu  Dense = softmax  Opt = adam  Batch = 30  Epoch = 6 | 0.83 | Negative: 0.81  Positive: 0.85 | Negative: 0.82  Positive: 0.84 | Negative: 0.81  Positive: 0.84 |
| LSTM2 | Conv1d = sigmoid  Dense = softplus  Opt = SGD  Batch = 32  Epoch = 6 | 0.55 | Negative: 0.51  Positive: 0.55 | Negative: 0.00  Positive: 1.00 | Negative: 0.00  Positive: 0.71 |
| LSTM3 | Conv1d = linear  Dense = softsign  Opt = Adadelta  Batch = 32  Epoch = 6 | 0.46 | Not Recorded  (broke NN) | Not Recorded  (broke NN) | Not Recorded  (broke NN) |
| **Classifier** | **Best Performance Achieved Using** | **Accuracy** | **Precision** | **Recall** | **F1 Score** |
| CNN1 | Conv1D = relu  Dense = softmax  Opt = adam  Batch = 30  Epoch = 5 | 0.84 | Negative: 0.80  Positive: 0.83 | Negative: 0.79  Positive: 0.84 | Negative: 0.80  Positive: 0.83 |
| CNN2 | Conv1d = sigmoid  Dense = softplus  Opt = SGD  Batch = 32  Epoch = 5 | 0.76 | Negative: 0.51  Positive: 0.55 | Negative: 0.00  Positive: 1.00 | Negative: 0.00  Positive: 0.83 |
| CNN3 | Conv1d = linear  Dense = softsign  Opt = Adadelta  Batch = 32  Epoch = 5 | 0.44 | Not Recorded  (broke NN) | Not Recorded  (broke NN) | Not Recorded  (broke NN) |

**Table 4: NN Results for Full Data**

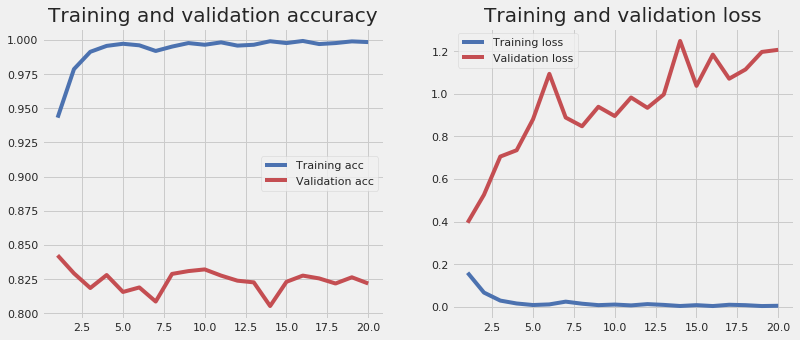
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Classifier** | **Best Performance Achieved Using** | **Accuracy** | **Precision** | **Recall** | **F1 Score** |
| LSTM1 | Conv1D = relu  Dense = softmax  Opt = adam  Batch = 30  Epoch = 2 | 0.96 | Negative: 0.90  Positive: 0.97 | Negative: 0.85  Positive: 0.98 | Negative: 0.87  Positive: 0.97 |
| LSTM2 | Conv1d = sigmoid  Dense = softplus  Opt = SGD  Batch = 32  Epoch = 2 | 0.82 | Not Recorded  (broke NN) | Not Recorded  (broke NN) | Not Recorded  (broke NN) |
| LSTM3 | Conv1d = linear  Dense = softsign  Opt = Adadelta  Batch = 32  Epoch = 2 | 0.18 | Not Recorded  (broke NN) | Not Recorded  (broke NN) | Not Recorded  (broke NN) |
| CNN1 | Conv1D = relu  Dense = softmax  Opt = adam  Batch = 30  Epoch = 3 | 0.97 | Negative: 0.92  Positive: 0.96 | Negative: 0.83  Positive: 0.98 | Negative: 0.87  Positive: 0.97 |
| CNN2 | Conv1d = sigmoid  Dense = softplus  Opt = SGD  Batch = 32  Epoch = 3 | 0.82 | Not Recorded  (broke NN) | Not Recorded  (broke NN) | Not Recorded  (broke NN) |
| **Classifier** | **Best Performance Achieved Using** | **Accuracy** | **Precision** | **Recall** | **F1 Score** |
| CNN3 | Conv1d = linear  Dense = softsign  Opt = Adadelta  Batch = 32  Epoch = 3 | 0.15 | Not Recorded  (broke NN) | Not Recorded  (broke NN) | Not Recorded  (broke NN) |

**Plot for BOW:**

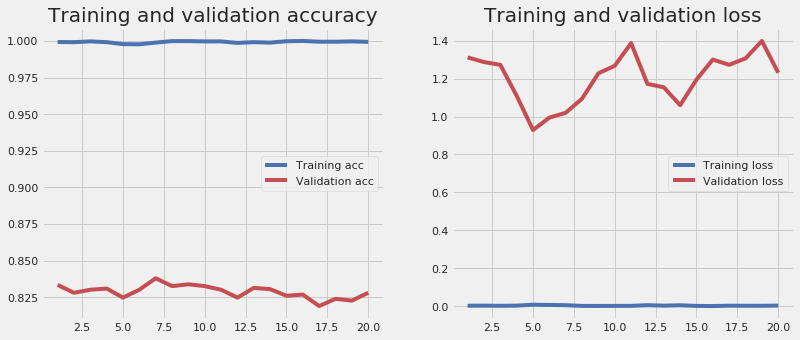
****

**Figure 2: Reduced Random Forest vs Full Random Forest**

**Plots for NN:**



**Figure 3: Loss Versus Epoch Number for LSTM1**

****

**Figure 4: Loss Versus Epoch Number for CNN1**

## Discussion

During the experiments with BOW, several observations were made that we found to be interesting. When we were experimenting with kNN, we predicted accuracies for multiple values of K, K 1 to 10. We noticed that accuracy went up as K increased but if it goes too high it will over classify, similar to decision trees. Despite that, we did notice that random forest produced a higher accuracy than Naive Bayes and kNN with the reduced data. When we were experimenting with decision trees classifiers with the reduced data, we noticed that Gini consistently produced much higher numbers in both accuracy, precision, and 10CV than Entropy. We also noticed that the higher the max depth is, the better accuracy you get. But if you make the depth too high, it will begin to overfit the tree and become too specific.

At one point we started experiencing issues with the amount of computing time required for the classifiers when we started working with SVM. Naive Bayes (in both instances) was the fastest algorithm used. Even with the full 500k records, it never took longer than a minute to finish computing its scores. However, each SVM method took about 2-4 minutes to return the accuracy and around 6-10 minutes when computing the respective 10-fold CV with the reduced data. When we worked with the full data, the computation times drastically increased. The computation times for SVM and Random Forest was one of the things we struggled with the most. However, the computation time makes sense as some SVM methods have relatively high space-time complexities. If we had more time, we would have investigated more optimized approaches of implementing those classifiers with large data sets.

During the experiments with the neural networks, we ran into many issues embedding the text and calculating certain metrics. In Table 3 and 4, some NN models do not have precision, recall, and F1 score calculated. We would get wrong dimension errors when attempting to compute these metrics on some models, but accuracy always computed fine. We also noticed that the Relu activation function always provided the best accuracy in all models. This is most likely due to the fact that Relu’s activation is sparse compared to others, making it more efficient. Additionally, the we discovered that the non-Adam optimizers have a higher learning rate. Our results with the non-Adam optimizers make sense since a high learning rate can cause the model to quickly converge to suboptimal positions. Finally, another interesting observation can be found when comparing the accuracy of CNN to LSTM models. Even when the activations and optimizer was sub-optimal, CNN always had a higher accuracy compared to LSTM, even if it was just slightly higher. This is most likely due to the increase in layers from LSTM to CNN. While this results in a higher accuracy, it also results in a decrease in train loss which is a sign of overfitting. So based on our experiments, it seems that the more layers you add, the more likely you will overfit the model.

## Extra credit

No Extra Credit was attempted.

## Contributions

**Nicolas Caceda:** For Part A, Nicolas got Partial Fit to work with Naïve Bayes and was able to perform analysis on the full data using the Hashing Vectorizer. For Part B, Nicolas researched multiple Keras models and experimented with batch size

**Nathalie Crespo:** For Part A, Nathalie implemented the random forest and SVM classifiers and computed their respective metrics. Additionally, for both Parts A and B Nathalie contributed heavily to the report.

**Damian Morgan:** For Part A, Damian implemented the Naïve Bayes, kNN, and Decision Trees classifiers and computed the respective metrics. For Part B, Damian wrote the NN code in the Colab Notebook and helped write the report.

## References

[1] Nikolai, Janakiev. *Practical Text Classification With Python and Keras* [Online]. Available: <https://realpython.com/python-keras-text-classification/>

[2] Keras Documentation. *Activations and Core Layers* [Online]. Available: <https://keras.io/activations/>

[3] Keras Documentation. *1D CNN for Text Classification* [Online]. Available: <https://keras.io/examples/imdb_cnn/>

[4] *Tokenize Each Row in a Data Frame* [Online]. Available: <https://stackoverflow.com/questions/55578939/tokenize-each-row-in-a-dataframe-for-loop-not-working>

[5] Chris Albon. *Validating Neural Networks.* [Online]. Available: <https://chrisalbon.com/deep_learning/keras/k-fold_cross-validating_neural_networks/>

[6] *Updating Feature Names into SciKit TFIDF Vectorizer.* [Online]. Available: <https://stackoverflow.com/questions/25154231/updating-the-feature-names-into-scikit-tfidfvectorizer>

[7] Nils Schluter. *Don’t Overfit! How to prevent Overfitting in your Deep Learning Models.* [Online]. Available: <https://towardsdatascience.com/dont-overfit-how-to-prevent-overfitting-in-your-deep-learning-models-63274e552323>

[8] Scikit Learn. *API Reference.* [Online]. Available: <https://scikit-learn.org/stable/modules/classes.html>

[9] Pandas. *API Reference.* [Online]. Available: <https://pandas.pydata.org/pandas-docs/stable/reference/index.html>

[10] Krishni. *K-Fold Cross Validation.* [Online]. Available: <https://medium.com/datadriveninvestor/k-fold-cross-validation-6b8518070833>

[11] Jason Brownlee. *Understand the Impact of Learning Rate on Neural Network Performance.* [Online]. Available: <https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/>