Wine Quality Classification Using Machine Learning

Seif Elkhashab
Department of Electrical and
Systems Engineering
Washington University in St. Louis
St. Louis, Missouri
e.seif@wustl.edu

Ben Ko
Department of Electrical and
Systems Engineering
Washington University in St. Louis
St. Louis, Missouri
ben.k@wustl.edu

Ben Watkins
Department of Electrical and
Systems Engineering
Washington University in St. Louis
St. Louis, Missouri
ben.m.watkins@wustl.edu

Introduction

While there are many definitions of machine learning, in a nutshell, machine learning is a tool for turning data into knowledge. People use machine learning to learn the rules governing the data and use them to make predictions or decisions without explicitly programming to execute the task. In this Case Study, we will analyze and dissect a data set with the goal of correctly classifying wines into their correct quality classification.

Wine Data Set:

Our group would like to use this powerful tool to make our fair share of predictions. The dataset used in the project is quality of red wine from UC Irvine's Wine Quality Dataset. With 1599 instances and 11 input variables, we believe that the dataset is sufficient for our use. The goal of the project is to determine the quality of the wine using input variables and whether these variables are relevant in determining the quality of the wine in the first place.

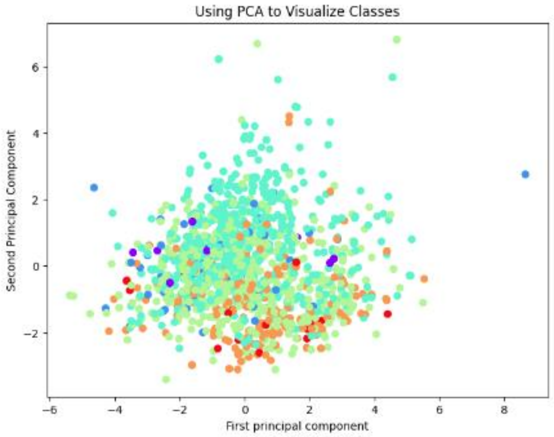


Figure 1 Shows the data visualized.

Our group used four machine learning methods: Random Forest Classifier, Support Vector Machine, Logistic Regression, and Artificial Neural Network. We provide a brief background on each method below.

Algorithms Used:

Random Forest Classifier: is an ensemble learning method that consists of multiple decision trees. The decision trees are trained on a slightly different set of observations and the nodes in each tree are split considering a limited number of features. The final predictions of the random forest classifier are made by averaging the predictions of each individual tree.

Support Vector Machines: are a popular type of supervised learning algorithm used in machine learning for classification and regression tasks. SVM works by finding the best possible hyperplane that separates different classes of data. A hyperplane is a decision boundary that helps to classify the input data into different classes. In SVM, the hyperplane that best separates the two classes is the one that maximizes the margin between the classes. The margin is the distance between the hyperplane and the closest data points from both classes. SVM can also use a kernel trick to transform the input data into a higher-dimensional space, where it may become easier to separate the data into classes. SVM has many applications in the fields of image recognition, bioinformatics, text classification, and more.

Artificial neural network is a computing system vaguely inspired by the biological neural networks of animal brains. It uses multiple layers of perceptrons which can be categorized into input layer, hidden layers, and output layer. Input layer nodes transmit input values to the hidden layer nodes without performing any computations. Each hidden layer node then computes the weighted sum of its inputs to form a scalar net activation. Then they output the nonlinear function of the net activation. After a single or multiple hidden layers, the output node computes the net activation based on hidden node outputs. Each output node emits the nonlinear function of the net activation.

Multi-Class Logistic Regression: is a popular statistical method that is used to analyze and model relationships between multiple independent variables and a categorical dependent variable. It is a technique used in machine learning, data science, and predictive analytics to build models that classify data into multiple classes. While we did use this model, we did not write an analysis on it as it seemed redundant.

Using these methods and tuning the hyperparameters of each model, we were able to predict the quality of wine with reasonable accuracy.

Methods

All the methods used will be expressed in detail as well as the steps taken to classify the data.

Exploratory Data Analysis:

Before starting to manipulate the data, it’s often helpful to graph the data to see if there are strong between certain classes in the data, this can be seen by the correlation matrix in Figure 2.

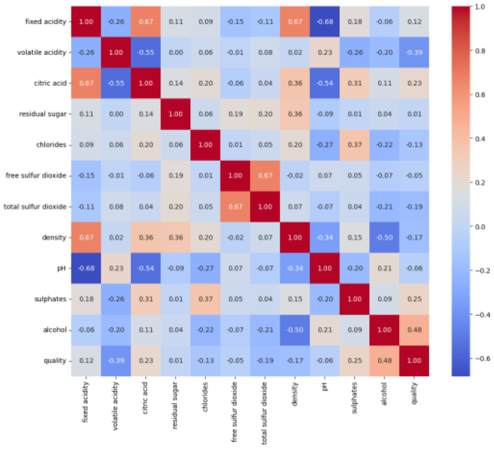


Figure 2 Shows the correlation matrix of all classes.

We also performed Principal Component Analysis, or PCA, to reduce the dimensionality of the dataset and visualize them in a plot for better understanding of how our data is structured. Figure 3 shows the data visualization using PCA.

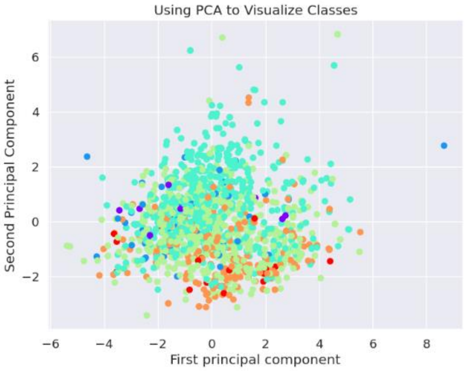


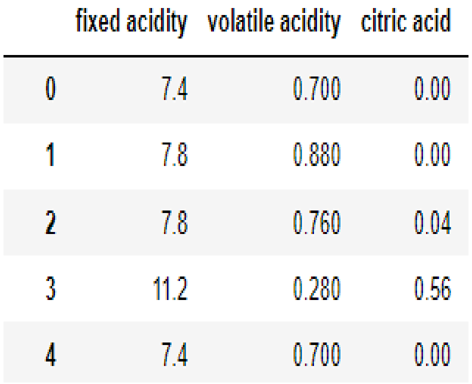
Figure 3 Visualization using PCA.

Data Pre-Processing:

Before the data set can be used, it must be inspected and deemed fit to use with the desired machine learning algorithms.

Data Pre-Processing is often very useful in increasing the accuracy and reliability of machine learning models. The first steps we took in cleaning our data were to check for and remove any null values, separate the input variables from the output variable, and rescale the data to more optimal ranges when training machine learning models. We also considered the significance of features with the least predictive capabilities.

An effective way to trim down the number of features we are working with, which can be beneficial for both computation times and the generalization of models like the ANN and SVM, is to select the k best features. There are many ways to score the k best features, but a popular method is comparing the ratio of explained variance to unexplained variance, or, in other words, the ANOVA F-value. We used this technique through sklearn to remove the four features with the lowest scores from our dataset: density, pH, and chlorides.



	fixed acidity	volatile acidity	citric acid
0	7.4	0.700	0.00
1	7.8	0.880	0.00
2	7.8	0.760	0.04
3	11.2	0.280	0.56
4	7.4	0.700	0.00

Figure 4 Shows the first three features of the data set.

Class Imbalance:

Class imbalance refers to when some classes in the data set have much lower representation, as seen in Figure 2, classes 3, 4, 7, & 8 have much lower instances than those of 5 & 6. This will cause the algorithms to be very biased towards classes 5 & 6 and ignore the other classes for the most part. It's also important to note that the data set gives the wine a quality score between 0 and 10 but the data set has no representation for qualities 0, 1, 2, 9 & 10, which is problematic.

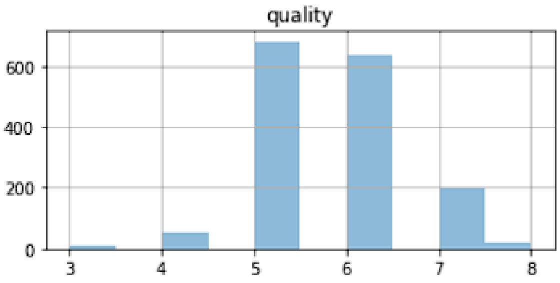


Figure 5 Shows how unbalanced the data is.

Solutions To Class Imbalance:

There are many ways to deal with Class Imbalance as it's a common problem in Machine Learning and data processing. Several strategies were explored below.

Oversampling Minority Class:

By copying the small amount of data, we have for the minority class and duplicating them, they are more represented, but the downside is of too many copies are made, the model may be overfit to those minority instances.

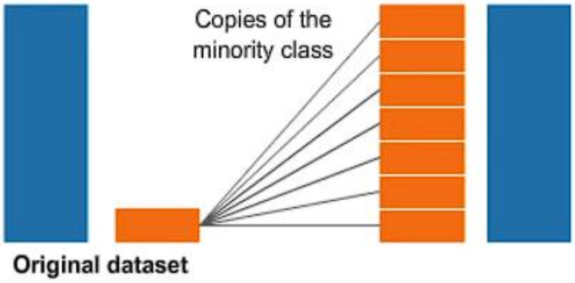


Figure 6 Diagram showing how oversampling works.

Under Sampling Majority Class:

By leaving out some of the instances of the major classes, the classes become more balanced, but the downfall is useful data is lost.

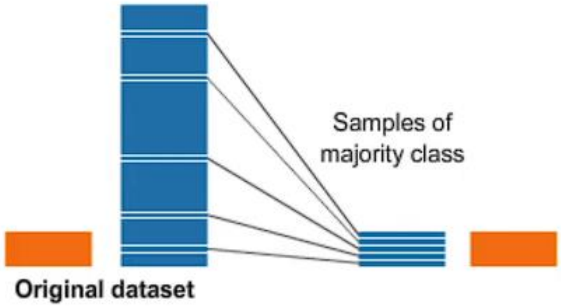


Figure 7 Diagram showing how under sampling works.

Synthetic Minority Oversampling (SMOTE):

Instead of just duplicating the minority class to increase the number of instances, synthetic instances can be used by using the nearest neighbors approach. The downside is that it is assumed that the correlation between the different features is relatively linear.

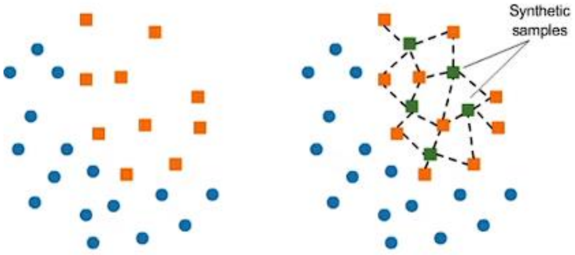
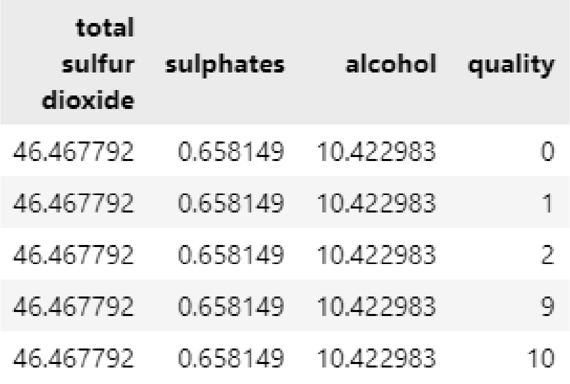


Figure 8 Graphic showing synthetic instances.

Data Imputation:

Although it is a common practice to identify missing values in a dataset and replace them with a numeric value, what we are trying is creating new data for classes with no other instances available. Using the mean value of each input variable, we created one instance for each of the missing classes.



total sulfur dioxide	sulphates	alcohol	quality
46.467792	0.658149	10.422983	0
46.467792	0.658149	10.422983	1
46.467792	0.658149	10.422983	2
46.467792	0.658149	10.422983	9
46.467792	0.658149	10.422983	10

Figure 9 Table shows the instances added for classes 0,1,2,9, & 10

Hyperparameter Tuning:

With all the algorithms used in this case study, several hyperparameters need to be chosen for the algorithm to run on. Instead of choosing the parameters though trial and error, grid search was used for an exhaustive sampling of the hyperparameter space to find the optimal results. We usually use accuracy as a measure of a model's predictive score.

Random Forest Classifier (RFC):

Hyper-parameter tuning for the random forest classifier algorithm was done in tandem with training the random forest classifier with the following datasets: the trimmed version, the original dataset, and the SMOTE dataset. While we could have relied on sklearn’s built in oop-scoring, which generates out-of-bag samples to test with, for this machine learning model we chose to create a validation set before creating three versions of the hyper-parameter tuning set: two modified hyper-parameter training sets and the original (with all 11 features). Get picture of one of the decision trees:

This allowed for more consistent testing methods, as we were able to validate the scores of the trained models using the same validation set. We manually coded in that the final model would use the original data set after seeing the results of the testing on the validation set, which used the entire training data set, including the validation set to get as many minority classifications as possible in the training set. Here are some graphs displaying the trend of hyper-parameters around the local maxima for score of the random forest classifier trained on the original training set.

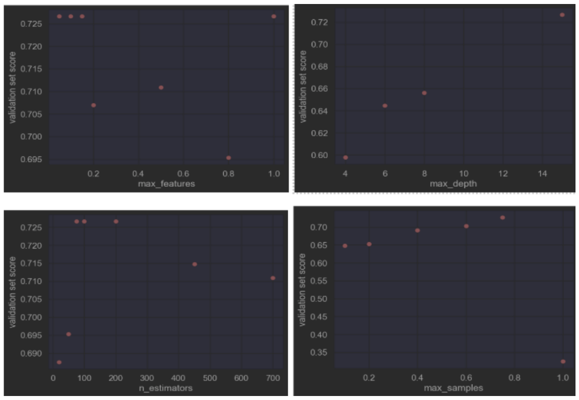
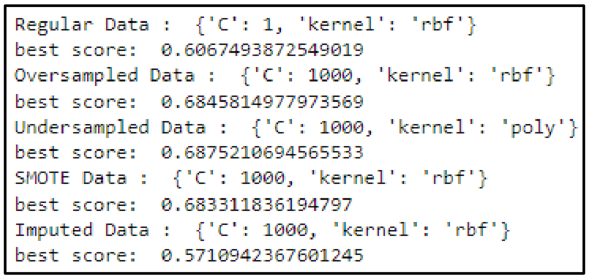


Figure 10 Shows the optimal parameters for RFC.

Support Vector Machines (SVM):

It’s important to note that for each data set, the hyperparameters are not necessarily the same, and that for the majority of data set types, c is at 1000 meaning that the algorithm will choose a smaller-margin hyperplane if that will cause more of the training instances to be classified correctly. This can clearly be seen in Figure 10.

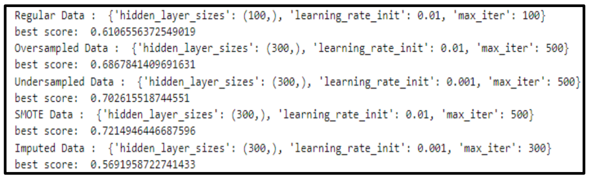


```
Regular Data :  {'C': 1, 'kernel': 'rbf'}
best score:  0.6067493872549019
Oversampled Data :  {'C': 1000, 'kernel': 'rbf'}
best score:  0.6845814977973569
Undersampled Data :  {'C': 1000, 'kernel': 'poly'}
best score:  0.6875210694565533
SMOTE Data :  {'C': 1000, 'kernel': 'rbf'}
best score:  0.683311836194797
Imputed Data :  {'C': 1000, 'kernel': 'rbf'}
best score:  0.5710942367601245
```

Figure 11 Shows the optimal parameters for SVM.

Artificial Neural Networks (ANN):

ANNs were implemented using the scikit-learn Package in python. While the Sklearn Package computes the mathematics behind the ANN algorithm, the user must still choose the hyperparameters for the algorithm, these hyperparameters are the activation function to be used, the solver, the learning rate, and the maximum number of iterations. As can be seen in Figure 11, as the data was transformed from the original data set, the hidden layer sizes tripled and the max_iter also significantly grew. This shows that changing the dataset muddied the relationship a bit to the point that now it takes significantly more resources for the algorithm to run but the accuracy was over all better.



```
Regular Data :  {'hidden_layer_sizes': (100,), 'learning_rate_init': 0.01, 'max_iter': 100}
best score:  0.6106556372549019
Oversampled Data :  {'hidden_layer_sizes': (300,), 'learning_rate_init': 0.01, 'max_iter': 500}
best score:  0.6867841409691631
Undersampled Data :  {'hidden_layer_sizes': (300,), 'learning_rate_init': 0.001, 'max_iter': 500}
best score:  0.702615518744551
SMOTE Data :  {'hidden_layer_sizes': (300,), 'learning_rate_init': 0.01, 'max_iter': 500}
best score:  0.7214946446687596
Imputed Data :  {'hidden_layer_sizes': (300,), 'learning_rate_init': 0.001, 'max_iter': 300}
best score:  0.5691958722741433
```

Figure 12 Shows the optimal parameters for ANN.

Results & Analysis:

After running the methods explained above, the results are analyzed, and conclusions are made about the data.

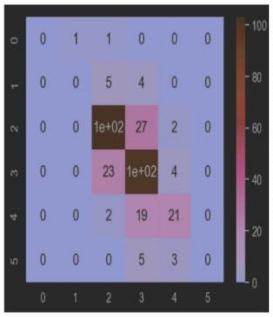
Random Forest Classifier (RFC):

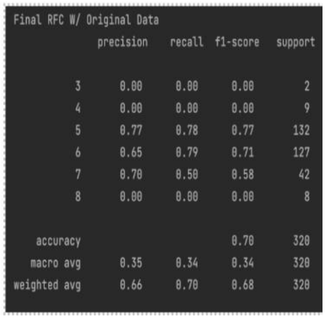
I found that the scores of the model trained on the original training set scored better on the validation set overall. This did not surprise me because of the nature of the hyper-parameter tuning.

The random forest classifier models allow for tuning of the power/generalization capabilities, and I thought that the grid-search would find the most optimal amount of generalization on its own. Also, with random forest classifier you can tune the depth of the trees, which would practically have the same effect as removing the features with the lowest predictive power.

It makes sense that the random forest classifier trained on the original data set is able to achieve the highest accuracy score out of the three, because the red wine data set is far from linearly separable. We managed to get an accuracy score of .7 on the final test set, but this could be an anomaly, and an accuracy of at least .65 is more reliable for our trained model. See the appendix for more details on the score.

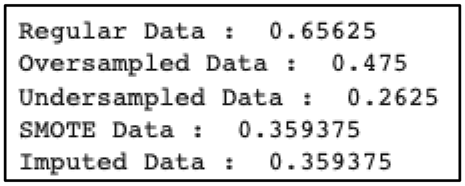
1





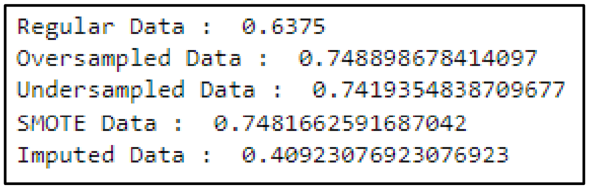
```
Final RFC W/ Original Data
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       0.00      0.00      0.00         9
           5       0.77      0.78      0.77       132
           6       0.65      0.79      0.71       127
           7       0.70      0.50      0.58        42
           8       0.00      0.00      0.00         8

    accuracy                           0.70       320
   macro avg       0.35      0.34      0.34       320
weighted avg       0.66      0.70      0.68       320
```

Figure 13 Final RFC: Final RFC Results



```
Regular Data :  0.65625
Oversampled Data :  0.475
Undersampled Data :  0.2625
SMOTE Data :  0.359375
Imputed Data :  0.359375
```

Figure 14 RFC baseline results tested on regular data.



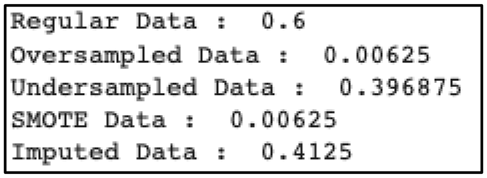
```
Regular Data :  0.6375
Oversampled Data :  0.748898678414097
Undersampled Data :  0.7419354838709677
SMOTE Data :  0.7481662591687042
Imputed Data :  0.40923076923076923
```

Figure 15 RFC results after hyperparameter tuning.

Support Vector Machines (SVM):

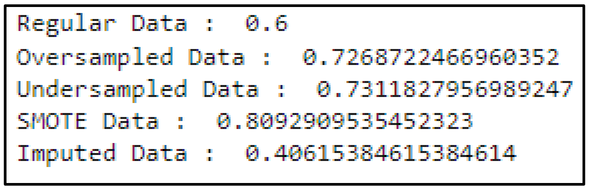
We managed to achieve a respectable accuracy of around .6 with the support vector machines model. The kernel “rbf” was by far the best kernel method when testing on the validation set. This is likely because it is more powerful than the other methods, and our data seems to require a powerful model.

SVM tested best on the validation set with the oversampled data, but this was very close, and overall, it seems like the less we touched the data set, the better it did on the validation set. This is likely because of how the GridSearchCV function evaluates its training scores, which leads to each modified training set overfitting their own data set, rather than being able to generalize on to the original validation set.



```
Regular Data :  0.6
Oversampled Data :  0.00625
Undersampled Data :  0.396875
SMOTE Data :  0.00625
Imputed Data :  0.4125
```

Figure 16 SVM baseline results.

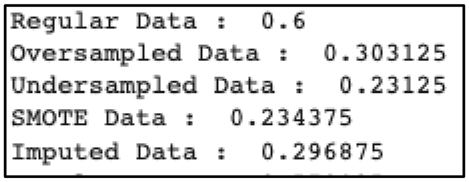


```
Regular Data :  0.6
Oversampled Data :  0.7268722466960352
Undersampled Data :  0.7311827956989247
SMOTE Data :  0.8092909535452323
Imputed Data :  0.40615384615384614
```

Figure 17 SVM results after hyperparameter tuning.

Artificial Neural Networks (ANN):

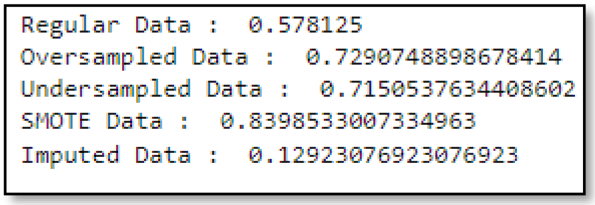
The ANN algorithm was run with both the original data set as well as the processed datasets to determine what gives the best results. Using the optimal hyperparameters calculated in the above section, the regular data performed the best as can be seen in Figure 19.



```
Regular Data :  0.6
Oversampled Data :  0.303125
Undersampled Data :  0.23125
SMOTE Data :  0.234375
Imputed Data :  0.296875
```

Figure 18 ANN baseline results.

If, however, the test data was processed in the same way that the training data was, then it can be seen as shown in Figure 20, that there is very significant model performance improvement compared to the regular data. The SMOTE data performed the best with an accuracy of about 84%, which is very good.



```
Regular Data :  0.578125
Oversampled Data :  0.7290748898678414
Undersampled Data :  0.7150537634408602
SMOTE Data :  0.8398533007334963
Imputed Data :  0.12923076923076923
```

Figure 19 ANN results after hyperparameter tuning.

General Analysis:

In general, it seems that the best model was the random forest classifier trained on the original data set.

The first likely cause is that the score used to tune the hyper-parameters was much more representative of the actual final testing data. Not because the data set is the least altered, but because with the manual grid search, which we implemented only for random forest classifier, we were able to test and score each set of hyperparameters on a validation set much more representative of the test set than the scores generated by the GridSearchCV function.

A second, less interesting cause, is that the random forest classifier model is just better with these types of data sets. As we saw, the red wine data set is far from linearly separable, and its minority classes are very small. A random forest classifier model could have an advantage if its training set represented the test set well (which it does) when there are minority classes as small as these. Its training algorithm does not rely on the gradient search method, which could be why it is better at handling minority cases.

Conclusion:

The red wine data set is very difficult to accurately classify, but with powerful machine learning models, we were able to get some respectable results. While the random forest classifier model had the best training results, that is only if the actual population follows the data set. If, for example, our data set is misrepresentative of the actual population, then perhaps the models trained on some of our modified data sets, like SMOTE, would be better at predicting results from outside our data set. But, with the information and data we have, modifying the data set did not significantly benefit the accuracy of our results.

Contributions:

Seif Elkhashab:

- Programmed Neural Networks.
- Helped Program SMOTE.
- Wrote ANN in report.
- Wrote Background on Algorithms.
- Wrote Class imbalance section.

Ben Ko:

- Programmed Hyper tuning.
- Programmed Data Preprocessing.
- Programmed MLR and SVM.
- Programmed Data Transformations.
- Wrote the introduction.

Ben Watkins:

- Worked on Manual Grid search for RFC and RFC coding.
- Helped with editing and writing reports, especially RFC related sections.

Appendix: