417 Introduction to Machine Learning Final Project

Mitchell Black
Tricia Brown

I. Introduction

Machine learning classification algorithms use prior examples to make predictions about the category of future data points. Model building requires a training dataset with numerous examples of inputs and outputs from which to learn. The goal of this project is to use two different classification methods to accurately label points in the wine quality data set from the UCI Machine Learning Repository. This data set contains information about both red and white wine samples. The inputs are objective tests and the output is based on sensory data, taken from the median of at least three evaluations made by wine experts. The wine quality is graded on a scale from 0 to 10 with 10 being very excellent. There are 11 attributes plus an output quality score:

1 - fixed acidity
2 - volatile acidity
3 - citric acid
4 - residual sugar
5 - chlorides
6 - free sulfur dioxide
7 - total sulfur dioxide
8 - density
9 - pH
10 - sulphates
11 - alcohol
Output variable:
12 - quality (score between 0 and 10)

The methods used are the Random Forest Classifier and K Nearest Neighbor Method. The Random Forest method uses multiple decision trees to settle on an average value while the K Nearest Neighbor Method classifies values based on the mean of the K closest points to the given input.

II. Methods

A. Random Forest

The Random Forest Classifier method is an ensemble model that consists of many decision trees. It uses bootstrap aggregation to create multiple new training data sets from the original as well as using a random subset of features considered when splitting nodes in order to enhance the randomness of the trees. This method combines hundreds or thousands of decision trees and trains each one on a slightly different set of observations. The final predictions of the random forest are made by averaging the predictions of each tree. The motivation for taking the sample mean can be demonstrated by the following equations:

$$E\left[\frac{1}{N}\sum_{i=1}^{N} Y_i\right] = \frac{1}{N}\sum_{i=1}^{N} E[Y_i] = \frac{1}{N} N\mu = \mu$$

$$Var\left(\frac{1}{N}\sum_{i=1}^{N} Y_i\right) = \frac{1}{N^2}\sum_{i=1}^{N} Var(Y_i) = \frac{1}{N^2} N\sigma^2 = \frac{\sigma^2}{N}$$

In summary, the average sample mean has the same expected value but the variance is reduced compared to each of the individual variance instances.
In our model, we used 70% of the data as the training set and used GridSearchCV to tune the hyperparameter of number of estimators and maximum depth.This resulted in an optimal max depth of 31 and number of estimators 91. We split our data set into a binary classification (-1 for bad wine, +1 for good wine) with a decision boundary at 5.5 since the wine quality values ranged from 3-8.

B. K Nearest Neighbor Method

The driving idea behind K Nearest Neighbors is that similar data points will be close to one other. KNN uses some measurements of distance to expand a sphere outward into feature space from the given input point until there are some number K points closest to the input point contained in the volume of the sphere. To classify a new point, we identify the K nearest points from the training data set and then assign the new point to the class having the most number of representatives amongst the K nearest neighbor set.

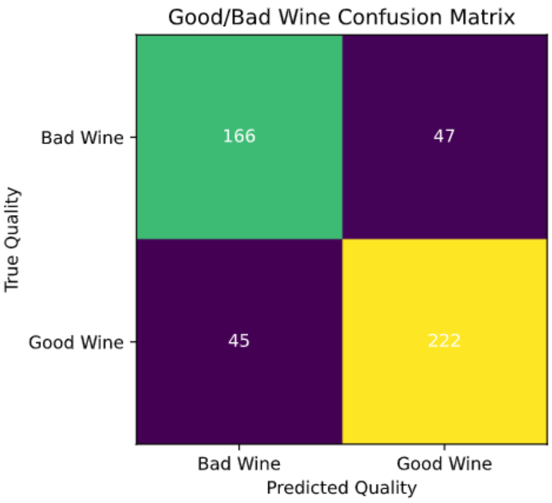
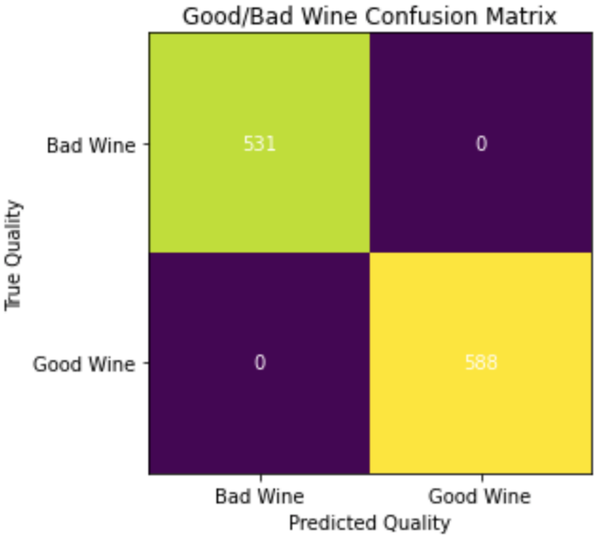
We performed the K Nearest Neighbors classification with binary classification (-1 for bad wine, +1 for good wine), as well as for the default dataset (wine qualities 3-8). Our distance metric is Euclidean distance (in appendix). The potential K values we used were 70 distinct integers spaced between 3 and 100 (generated via numpy.linspace).

Using hyperparameter tuning via sklearn.model_selection.GridSearchCV, in the binary classification experiment, we determined that the optimal number of neighbors is 19. In the default classification experiment, the optimal number of neighbors is 22.

III. Results and Analysis

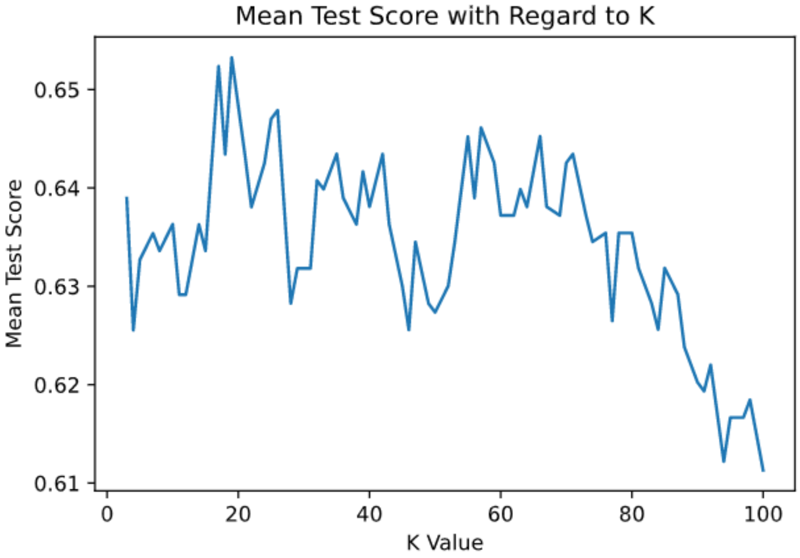
A. Random Forest

Our Random Forest model produced a training score of 1.0 and a test score of 0.8083.

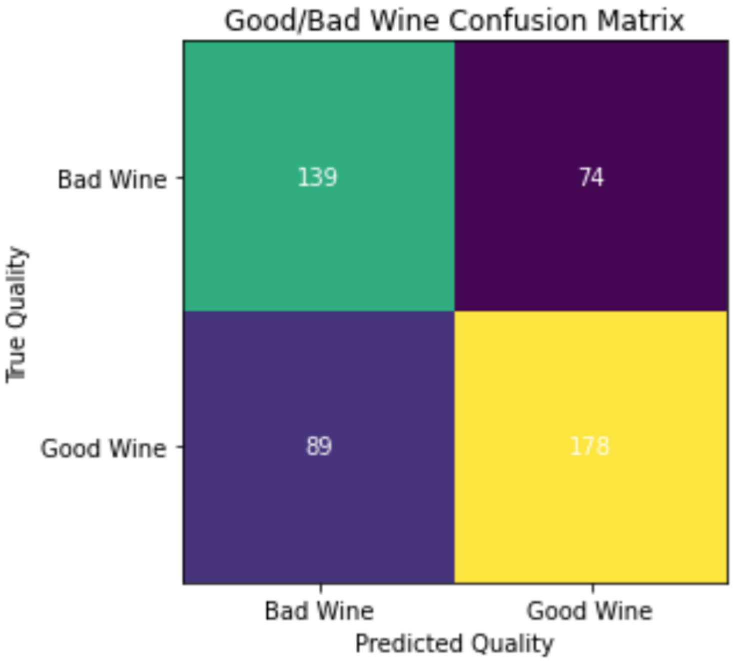
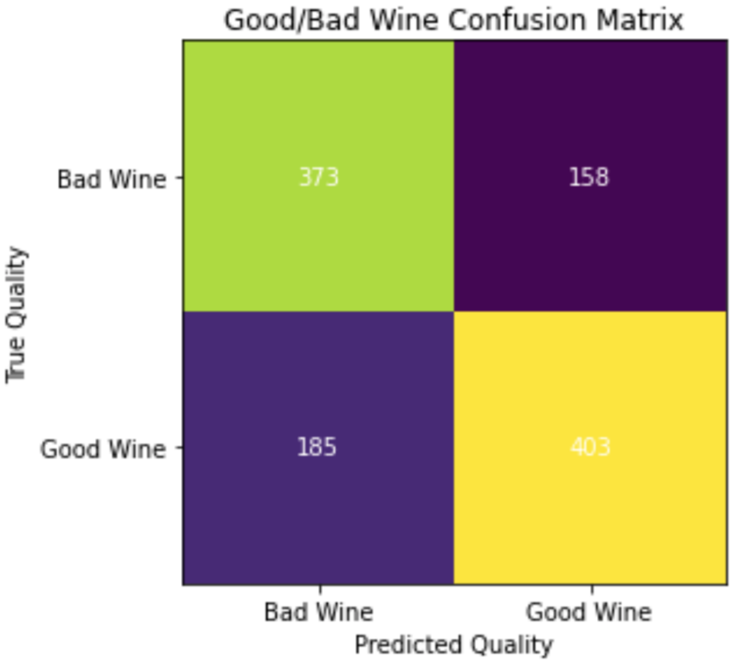
Testing Confusion Matrix	
Training Confusion Matrix	

B. K Nearest Neighbor

The K Nearest Neighbor binary classifier produced a training score of 0.6935 with a testing score of 0.6604 . These scores were produced from using GridSearchCV for hyperparameter tuning. The following plot of Accuracy Scores vs. K was used to determine the optimal value of K to be used:

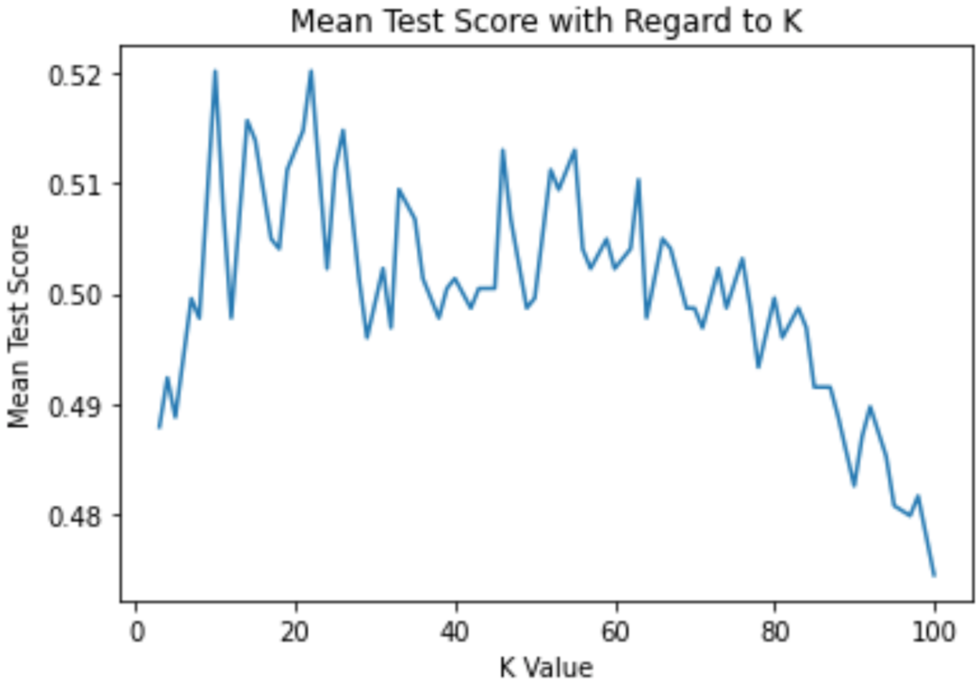


The KNN binary classification produced the following confusion matrices:

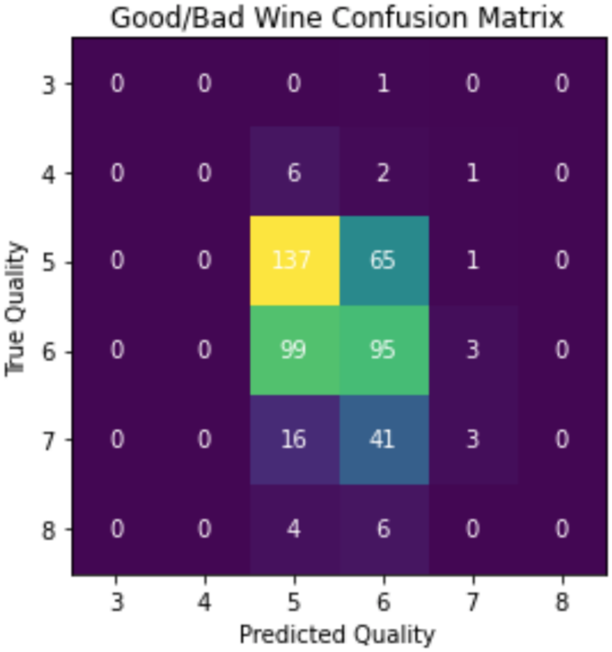
Testing Confusion Matrix	
Training Confusion Matrix	

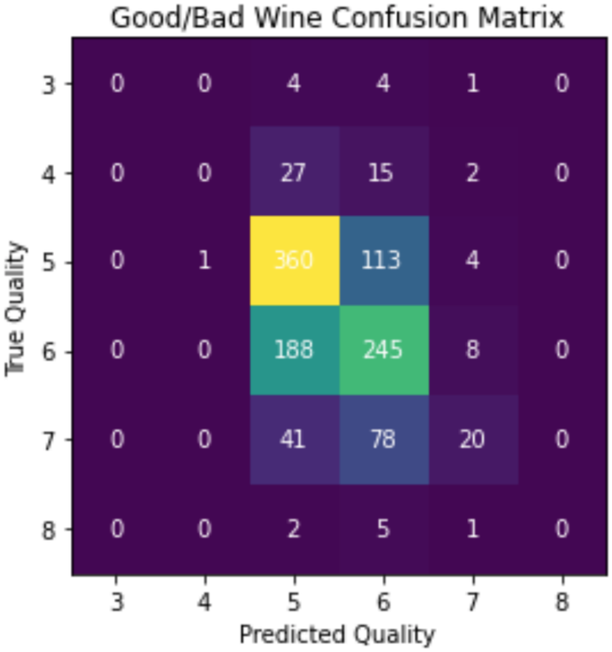
KNN Non-Binary Classification

The K Nearest Neighbor non-binary classifier produced a training score of 0.5585 with a testing score of 0.4895 . These scores were produced from using GridSearchCV for hyperparameter tuning. The following plot of Accuracy Scores vs. K was used to determine the optimal value of K to be used:



The KNN non-binary classification produced the following confusion matrices:

<table>
<tr>
<td>Testing Confusion Matrix</td>
<td>


</td>
</tr>
<tr>
<td>Training Confusion Matrix</td>
<td>


</td>
</tr>
</table>

IV. Conclusions

The Random Forest produced more accurate classification than the Nearest Neighbor method. This was to be expected because K Nearest Neighbor suffers from the curse of dimensionality. KNN makes the assumption that similar data points are close together and largely depends on having a dense data set. Higher dimensional data sets have a significantly larger amount of space between points, and the amount of data must grow exponentially to maintain the same density that is needed. With a fixed amount of data, and KNN requiring points to be close on every axis, the accuracy suffers as a result.

Mitchell wrote the majority of the code and Tricia wrote the report with editing from each team member on both components.

V. Appendix

Please see attached Jupyter Notebook file for runnable python code as specified from Piazza post.

Euclidean Distance:

$$d(\boldsymbol{x}, \boldsymbol{y}) = \left(\sum_{i=1}^{m} (x_i - y_i)^2\right)^{\frac{1}{2}}$$