Nathaniel Hoffelmeyer, Kirby Hood
IST 565 Data Mining

Final Project: Let's Wine About the Quality

Introduction

Anthony Bourdain once said, "Meals make the society, hold the fabric together in lots of ways that were charming and interesting and intoxicating to me" Hoffower, Gal. (2018). Along with meals served in many different regions of the world, wine accompanies and compliments the dishes that are served. People of different cultures come together to celebrate birthdays, holidays, religious beliefs, promotions, and sometime sorrow, but one thing is for certain that wine is more than likely always present. Cultures have produced wine since the biblical times and still remains popular today. Wine has the ability to bring people together. Whether friend or foe, share a meal and a "quality" bottle of wine.

Choosing a bottle of wine can be based on many factors including expert's quality rating, aromas, price, region, body, and chemical attributes. For the everyday wine connoisseur, "quality" is very important.

Data Acquisition

After researching and viewing many different data sets and the possibilities of different data mining techniques that could be used, the Wine Quality data set was chosen. This set was retrieved from the UCI repository archives at University of California Irvine. This data set was also chosen because it was one of the most complete sets that was explored and required very little transformation and cleaning.

The wine data was created by using data from the production of wine in the region of Vinho Verde. Vinho Verde is in the northwestern region of Portugal, and the wine is only made from grapes that are indigenous to the region. The Vinho Verde region, "is one of the largest and oldest wine regions in the world. It is home to thousands of producers, generates a wealth of economic activity and jobs and strongly contributes to the development of Minho and the country" Vinho Verde History (2018).

This wine region produces not only white and red wines, but also sparkling wines, Brandy Wine, and red or rose vinegar. The wine quality data set contains chemical attributes that are important in determining the quality of wine scientifically. The set also contains a sensory attribute that was determined by wine experts with a minimum of three evaluations were made. The "quality" attribute is a scale of 0 to 10, zero meaning very bad and 10 being outstanding.

Data Mining Opportunity and Experiment Design

Wine quality is highly considered when consumers are purchasing a bottle of wine as well as when the wine is being produced. Using the wine quality data set that was created originally by Vinho Verde, different models and algorithms will be applied, to output predictions of "quality" based on the chemical attributes.

The expert "quality" ratings currently in the wine data set are sensory driven. Different data mining techniques will be used to determine possible association rules to determine if the chemical attributes can accurately predict the expert's "quality" rating. Classification to Clusters technique can also be applied to help determine different drivers of the "quality" ratings. Naïve Bayes, Decision Tress, and Random Forests can also be applied to best predict what chemical attributes contribute to the expert's quality scores.

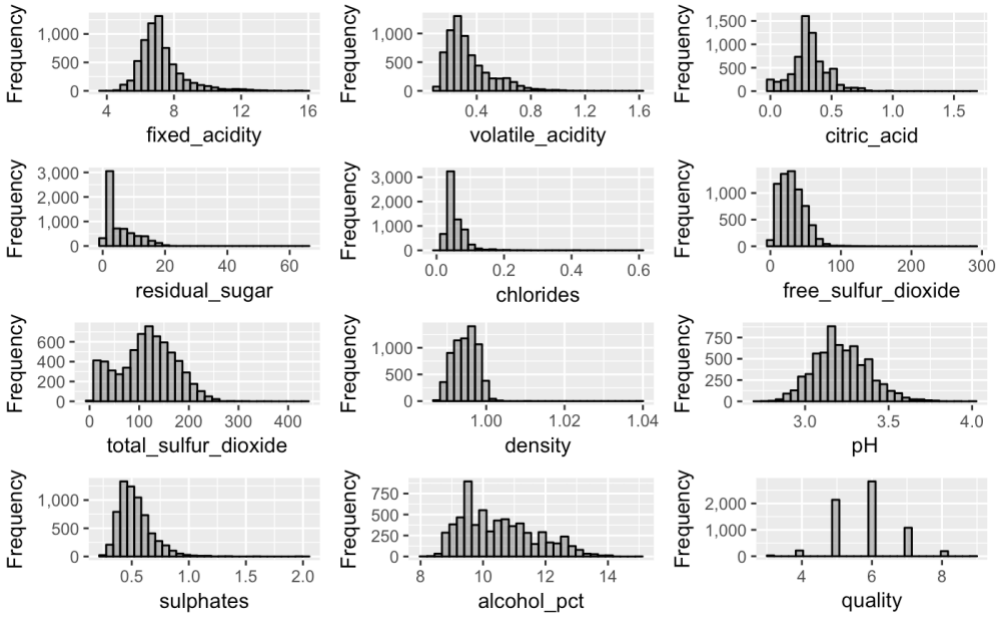
Data Exploration

When examining the chemical attributes of the wine data set, there are attributes that deal with the sugar content and acidity of the wines, as well as the sulfite levels and pH. It is possible as well that all of these can indicate the level of alcohol and quality of wines. All of the attributes are numbers.

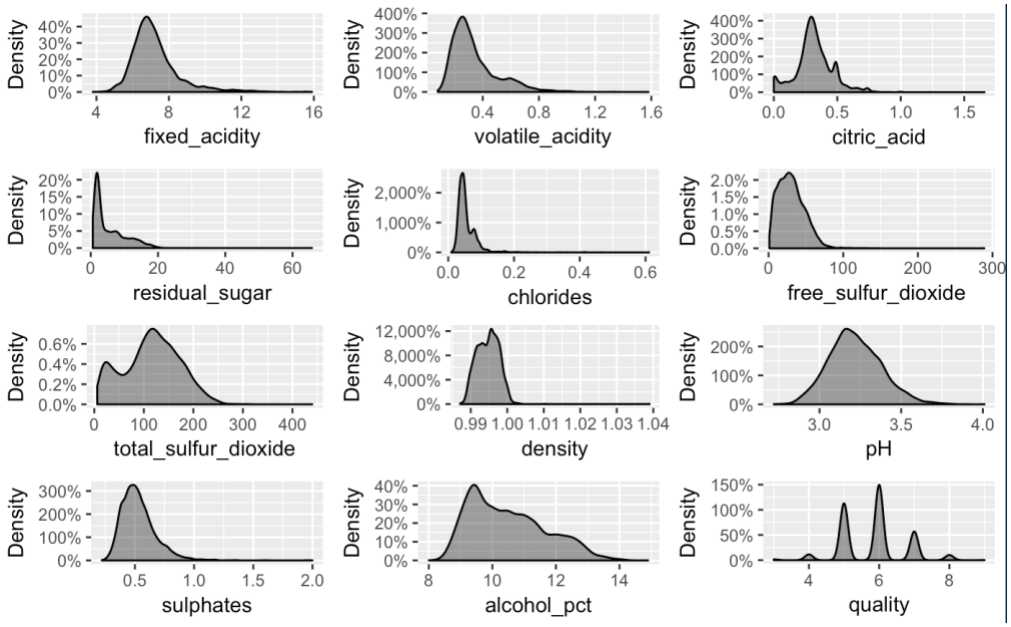
Intuitively it is known that many people do not like wines with high tannins (acidity). Wines that are high in tannins is what can give people headaches, but other people seek out wines with high tannins due to taste and quality.

The distribution of the data and the variables is very important to see what the data we are dealing with looks like. From the histograms created we are able to tell that some of the variables are skewed, and do not represent a normal distribution. The only variable with a normal distribution is the "quality" attribute. This could indicate that there are outliers within the data set and the range of the variables are also very large.

Frequency Distribution of Attributes:



Density Distribution of Attributes:



Calling the summary function on the entire data set will allow the range in variables to be seen.

Summary of wine quality data set:

Variable	color	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulfites	alcohol_pct	quality
Min	red: 1599	3.800	0.0800	0.0000	0.600	0.00900	1.00	6.00	0.9871	2.720	0.2200	8.000	3.000
1st Qu	white:4898	6.400	0.2300	0.2500	1.800	0.03800	17.00	77.00	0.9923	3.110	0.4300	9.500	5.000
Median		7.000	0.2900	0.3100	3.000	0.04700	29.00	118.00	0.9949	3.210	0.5100	10.300	6.000
Mean		7.215	0.3397	0.3186	5.443	0.05603	30.53	115.70	0.9947	3.219	0.5313	10.490	5.818
3rd Qu		7.700	0.4000	0.3900	8.100	0.06500	41.00	156.00	0.997	3.320	0.6000	11.300	6.000
Max		15.900	1.5800	1.6600	65.800	0.61100	289.00	440.00	1.039	4.010	2.0000	14.900	9.000

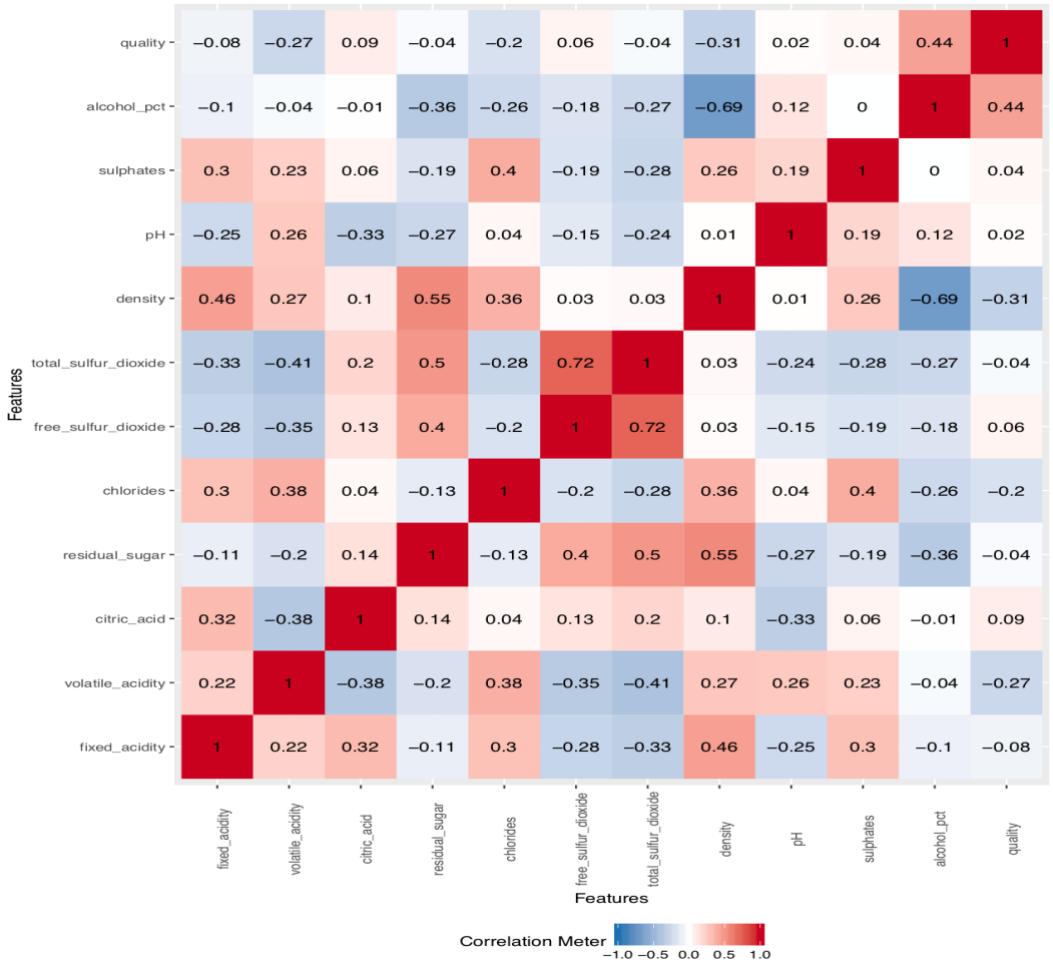
To better understand the ranges of variables, the variables need to be researched to determine the industry standard of each as well if one variable impacts another variable.

The acidity of wine impacts the taste of wine making it either sour or tart. When a wine lacks acidity is can be considered to be flat in taste. The acids also play a major role in impacting the pH level of a wine as well as the color, and lifespan of a wine. Acidity in wine is divided into two categories, fixed acidity and volatile acidity. The citric acid variable is a derivative of the fixed acidity variable. If a wine is produced with "warm climate grapes, [the wine] can be low in acid, more or less depending on the variety" Nierman (2004). Wines that have a pH level as high as 4.0 makes the wine's taste much softer and more popular with consumers. Volatile acidity levels should be low and barely detectable.

Also, wines with more sugar content will result in a higher alcohol percentage. The sugars transform into higher alcohols during the fermentation process. Higher alcohol percentages can impact the aromatic effects of wine in either a positive or negative way, but higher alcohol percentages normally do not have an impact on a quality rating, such as the one in the wine data

set. Sulfur dioxide refers to the fruit preservatives that are used during the fermentation process. The yeast that is also used during this process also created sulfites (sulfur dioxide). It is very rare to find a wine that is free from sulfur dioxides.

Correlation of Attributes:



The correlation of attributes shows that alcohol and density are negatively correlated, but the percentage of alcohol in the wine is positively correlated with quality. This insight is different than what normally occurs. Normally the percent of alcohol would not be correlated with a sensory attribute such as “quality.” The fact that we have a positive correlation between alcohol and quality could indicate that the percent of alcohol positively impacts the aromatic effects of the wine and could possibly give a good quality score by experts. More research would need to be done to determine the causation between the two attributes.

Data Preparation

The wine quality data set contains 4,898 white wine varieties from the Vinho Verde region of Portugal with 12 attributes about the chemical consistency of each white wine. There was also another red wine data set that contained 1,599 variables and 12 attributes. Both data sets were combined to create one data set. This resulted in a data set containing 6,497 observations and 13 attributes. An attribute was added to be able to distinguish between red and white wine if needed.

The data was checked for complete cases and NAs and all observations and variables were complete and no missing values present.

Discretization took place on the quality attribute, which was our target variable. This variable ranged from 3 to 9 and needed to be converted from an integer to a factor to classify it as either "good" or "bad" quality wine.

Data slicing was a step that was completed to split data into training and testing sets. The training data set was used specifically for our model building. It was ensured that the test data set was not manipulated and kept in its natural form. The slicing was done with the 2/3 rule, where 2/3 of the data is used for training and the remaining 1/3 for testing. The data was sampled to create and index so that not all one color was used for the sample and the other for the test.

kNN Model

The caret package provides a train() method for training the data using various algorithms. Different parameter values for different algorithms need to be passed through the train() function. However, before the train() method is called, trainControl() needs to be used first to control for computational nuances and allows for training the models.

Three parameters are set within the trainControl() method. The "method" parameter holds the details about resampling. The "method" can be set with various values like "boot", "boot632", "cv", "repeatedcv", "LOOCV", "LGOCV" etc. For this analysis, repeatedcv was used which was repeated cross-validation.

The "number" parameter holds the number of resampling iterations, 10-fold cross validation specification. The "repeats" parameter contains the complete sets of folds to compute for the repeated cross-validation. This means the 10-fold validation 3 times, so it's super extra groovy cross validation.

Before training the knn classifier, set.seed() is used to allow for replication of the results.

In training the knn classifier, the train() method is passed with "method" as "knn". Since the model is trying to predict wine quality, that parameter is passed first. The quality~. denotes our formula for using all attributes in the classifier and quality, again, is on the left as the target variable. The "trControl" parameter is passed with results from the trainControl() method. The "preProcess" parameter is for preprocessing the training data, which is especially helpful because some of the data points such as sulfur dioxide and sulphates have outliers that need to be controled for.

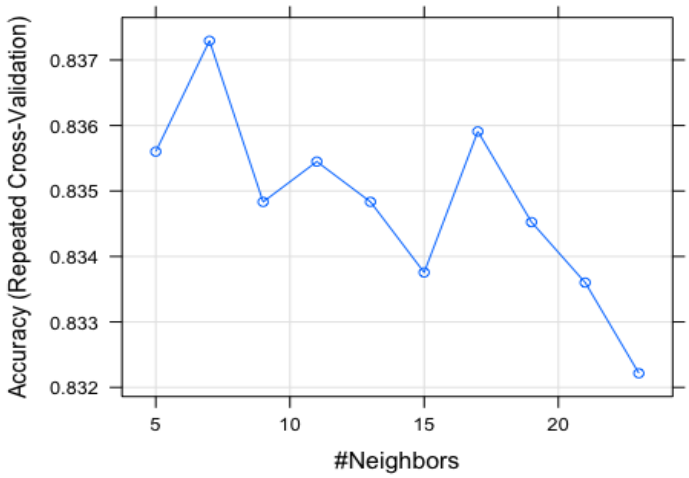
As discussed earlier data preprocessing is a mandatory task. Thus pass 2 values into the "preProcess" parameter "center" & "scale". These parameters center and scale the data, because the data has a wide variety of ranges, and values. To explain further, after preProcessing is passed, the training data variables are converted to have a mean value of about "0" and standard deviation of about "1". The data was normalized during this process.

The "tuneLength" parameter held an integer value and will be used for tuning our algorithm!

Trained kNN Model Result

The kNN model shows Accuracy and Kappa metrics for different k-values, with the objective of optimizing the model by selecting the best k-value based on Accuracy. The kNN model selected k = 7 using Accuracy to pick the optimal model (e.g. largest value of accuracy used, or most accurate k value).

There is variation in Accuracy versus the various K values by plotting these in a graph:



Test Set Prediction using kNN Model

Now that the kNN model is trained, and using k = 7, the model is ready for the prediction of wine quality using the test data set. Prediction is performed using the predict() method from the caret package.

As mentioned previously, the caret package provides the predict() method for predicting results of the previously trained model. Two arguments are passed into the method; The first parameter is the trained model (knn_fit) and the second parameter "newdata" holds the testing data frame (wine.test), which was created earlier in the Data Slicing process. The predict() method returns a list of predicted values, which are saved as knn.pred to reference later.

Using caret's confusionMatrix() method, will retune the statistics of the model's results and evaluate performance accuracy of how the model classified the variables.

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  bad good
##       bad  1589  237
##       good  143  197
##
##                Accuracy : 0.8246
##                  95% CI : (0.8079, 0.8404)
```

```
##     No Information Rate : 0.7996
##     P-Value [Acc > NIR] : 0.001788
##
##                   Kappa : 0.4042
##  Mcnemar's Test P-Value : 1.835e-06
##
##             Sensitivity : 0.9174
##             Specificity : 0.4539
##          Pos Pred Value : 0.8702
##          Neg Pred Value : 0.5794
##              Prevalence : 0.7996
##          Detection Rate : 0.7336
##    Detection Prevalence : 0.8430
##       Balanced Accuracy : 0.6857
##
##        'Positive' Class : bad
```

The KNN model thus reports, with a kvalue=7, accuracy of 82.46%. This is not a bad accuracy but modifying the variables ran against the model could potentially result in a higher accuracy. If the color variable from the attributes in the model will this improve the accuracy of the kNN model?

Again, the model shows Accuracy and Kappa metrics for different k-values, this time using 11 predictors and not using color as one of those 11 predictors, with the objective of optimizing the model by selecting the best k-value based on Accuracy. The model selected k = 17 using Accuracy to pick the optimal model (e.g. largest value of accuracy used, or most accurate k value).

```
Confusion Matrix and Statistics
##
##           Reference
## Prediction  bad good
##       bad  1626  269
##       good  106  165
##
##                Accuracy : 0.8269
##                  95% CI : (0.8103, 0.8426)
##     No Information Rate : 0.7996
##     P-Value [Acc > NIR] : 0.0007083
##
##                   Kappa : 0.3712
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9388
##             Specificity : 0.3802
##          Pos Pred Value : 0.8580
##          Neg Pred Value : 0.6089
##              Prevalence : 0.7996
##          Detection Rate : 0.7507
##    Detection Prevalence : 0.8749
```

```
##       Balanced Accuracy : 0.6595
##
##        'Positive' Class : bad
```

The model improved slightly, and the accuracy reported 82.69%, versus the previous 82.46%. It's a small change, though, and the number of nearest neighbors greatly increased. The color attribute would be an easy way for the model to group neighbors, and since that attribute was removed, it must work a little harder, to get better accuracy.

The model was ran one more time removing some variables that were correlated. For instance, fixed acidity is an overall measure of things like citric acid and volatile acid combined, and sulfites are derivatives of our sulfur dioxide variables. Given the correlation that exists here, as we can see strongly from our correlation matrix (*plot_correlation(wine, type = 'continuous','Review.Date')*), Also, the color attribute was added back into the model for this run.

The model used 8 predictors and settled on a k-value of 17. The model ran a bit faster, which means computing time was saved. If there is an increase in model performance it can be leveraged that time savings is a factor of which model to use.

```
Confusion Matrix and Statistics
##
##           Reference
## Prediction  bad good
##       bad  1623  273
##       good  109  161
##
##                Accuracy : 0.8236
##                  95% CI : (0.8069, 0.8395)
##     No Information Rate : 0.7996
##     P-Value [Acc > NIR] : 0.002536
##
##                   Kappa : 0.3588
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9371
##             Specificity : 0.3710
##          Pos Pred Value : 0.8560
##          Neg Pred Value : 0.5963
##              Prevalence : 0.7996
##          Detection Rate : 0.7493
##    Detection Prevalence : 0.8753
##       Balanced Accuracy : 0.6540
##
##        'Positive' Class : bad
```

Unfortunately, the accuracy decreased to 82.36%. Thus, our fit2 KNN model, the one without color, which gave the highest prediction accuracy.

Creating SVM and Random Forests models, also using the caret package, could potentially improve our prediction accuracy.

SVM Classifier using caret in R

The training and testing data was examined using SVM and the R caret package. The principle behind SVM (support vector machines) is to build a hyperplane that separates data for different classes. The procedure of building this hyperplane is the main task of a SVM classifier, and can vary. The most important thing the model considers as it builds the hyperplane is maximizing the distance from the hyperplane to the nearest point of either class. Each of those points are individually known as a Support Vector - hence, "support vector machine".

Training the SVM Model

A svmctrl variable was created by utilizing the trainControl() method. In this method three parameters are set. The first specifies the controlling the training of the model using repeated cross validation, the number specifies that it is 10 fold (k-fold), and the repeats are set to 3.

The new SVM classification model is a linear model, and it is created by setting the svmLinear variable using the train() method from the caret package.It is a linear Support Vector Machine model, as the method is specified to be "svmLinear". The label variable is the variable that the model is attempting to predict. With the model controlled by our svmctrl variable as described above, and the data standardized using preProcess to center and scale (normalize) the variables that all have differing ranges to a mean of zero and a standard deviation of 1. Finally, the tuneLength parameter allows the ability to finely tune our algorithm.

The model shows that it maintained a tuning parameter of "C" constant at 1, which is reasonable given that this is a linear model.

Predictions using SVM

The SVM trained model, using C=1 (tuning parameter for a linear model), it is ready to predict the testing data labels. This will be done in a similar fashion to how kNN was implemented, using caret's predict() method.

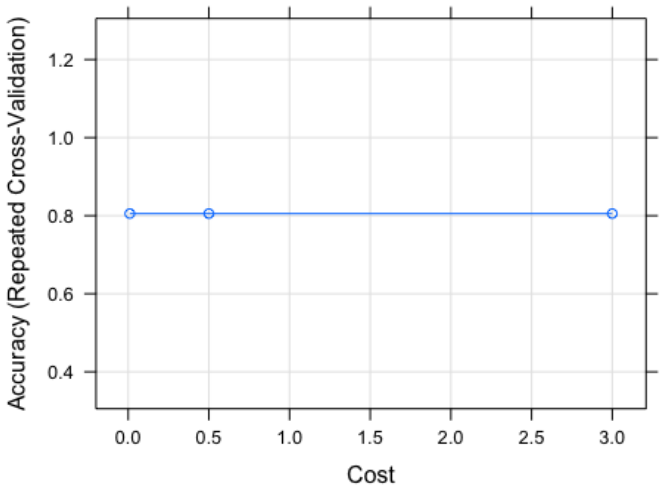
The confusionMatrix is ran to determine our test accuracy:

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  bad good
##       bad  1732  434
##       good    0    0
##
##                Accuracy : 0.7996
##                  95% CI : (0.7821, 0.8163)
##     No Information Rate : 0.7996
##     P-Value [Acc > NIR] : 0.5128
##
##                   Kappa : 0
##  Mcnemar's Test P-Value : <2e-16
```

```
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.7996
##          Neg Pred Value :    NaN
##              Prevalence : 0.7996
##          Detection Rate : 0.7996
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : bad
```

The matrix reveals that this model underperformed our KNN model, from an accuracy perspective, with 79.96% of quality predicted correctly.

Now that the linear SVM model was built, it can take a customized C value using grid searching. To do this values of C (using expand.grid()) into a "grid" dataframe, and the dataframe is used to test our classifier at specific varying C values. Caret's tuneGrid parameter can be used for this tuning parameter.



The summary and plot displayed above show that the classifier gives the highest accuracy at C = 0.01. Actually, it shows that the accuracy is constant no matter which option is choosen, meaning that cost level has not impact on the model. The model is ready to make some predictions on our test set.

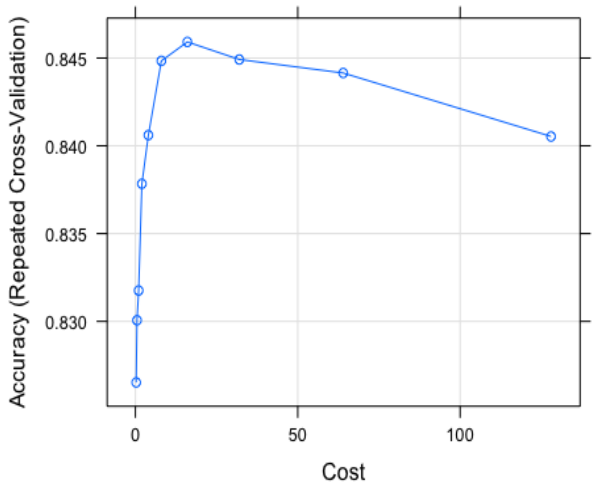
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  bad good
##       bad  1732  434
##       good    0    0
##
##                Accuracy : 0.7996
##                  95% CI : (0.7821, 0.8163)
##     No Information Rate : 0.7996
```

```
##     P-Value [Acc > NIR] : 0.5128
##
##                   Kappa : 0
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.7996
##          Neg Pred Value :    NaN
##              Prevalence : 0.7996
##          Detection Rate : 0.7996
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : bad
##
```

By adjusting the C parameter to C = 0.01, there are not any prediction accuracy increases. The linear model still predicts with 79.96% accuracy, compared with our best KNN at 82.69%.

Non-Linear SVM

Another model is created to predict the quality using non-linear kernel, such as radial basis function. To use the RBF kernel, the train() method's "method" parameter needed to change to "svmRadial" from the previous svmLinear. The Radial model will select the approriate cost (C) value and sigma (essentially standard deviation) for the model. Note that when a larger sigma is present, the model decision tends to be flexible and smooth, and thought it tends to make wrong classifications while more often, it avoids the hazard of overfitting.



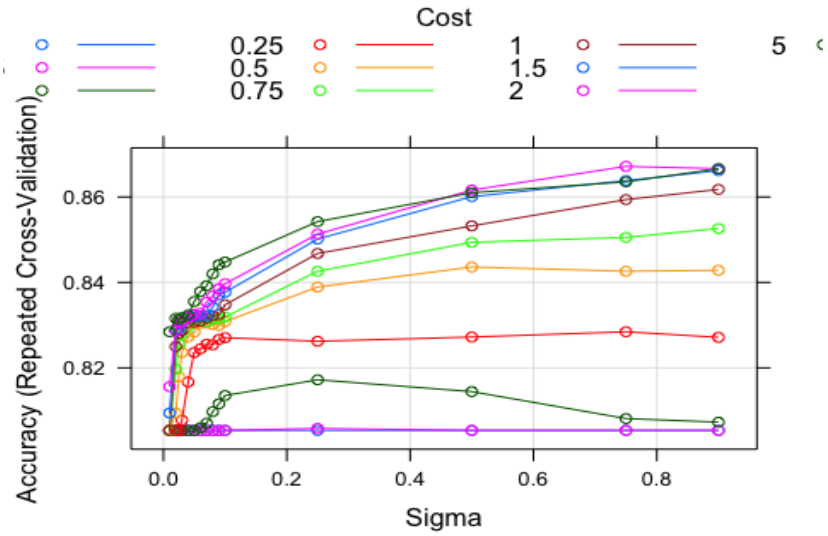
As seen above, the model is trained on a constant sigma of 0.08496634. Also, the smaller sigma gives more accurate predictions - but also increases the risk of over-fitting. The C value selected by the model for optimality was C = 16.

The model is ready to use its accuracy on the test set.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  bad good
##       bad  1634  246
##       good   98  188
##
##                Accuracy : 0.8412
##                  95% CI : (0.8251, 0.8563)
##     No Information Rate : 0.7996
##     P-Value [Acc > NIR] : 3.983e-07
##
##                   Kappa : 0.4318
##  Mcnemar's Test P-Value : 2.268e-15
##
##             Sensitivity : 0.9434
##             Specificity : 0.4332
##          Pos Pred Value : 0.8691
##          Neg Pred Value : 0.6573
##              Prevalence : 0.7996
##          Detection Rate : 0.7544
##    Detection Prevalence : 0.8680
##       Balanced Accuracy : 0.6883
##
##        'Positive' Class : bad
```

84.12% accuracy, which is the best model returned so far, versus both the linear SVM models and the KNN.

Further tuning the model with different values of C and sigma will hopefully increase the accuracy. Again, the grid search approach was used to implement this.



After a lengthy run of the large expanded grid radial SVM model, it returned the optimal sigma and C values of 0.75 and 2, respectively. Using this model, it can be tested against the testing set for accuracy.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  bad good
##       bad  1644  210
##       good   88  224
##
##                Accuracy : 0.8624
##                  95% CI : (0.8472, 0.8767)
##     No Information Rate : 0.7996
##     P-Value [Acc > NIR] : 1.449e-14
##
##                   Kappa : 0.5201
##  Mcnemar's Test P-Value : 2.394e-12
##
##             Sensitivity : 0.9492
##             Specificity : 0.5161
##          Pos Pred Value : 0.8867
##          Neg Pred Value : 0.7179
##              Prevalence : 0.7996
##          Detection Rate : 0.7590
##    Detection Prevalence : 0.8560
##       Balanced Accuracy : 0.7327
##
##        'Positive' Class : bad
```

Accuracy was gained relative to the base radial model from above, as this model's accuracy is 86.24% compared to the previous 84.12%. This gain in accuracy also comes with a decreased risk of overfitting, particularly as mentioned above w.r.t. sigma being so small. As sigma has been increased from the initial radial model, the overall risk of overfitting has gone down while model performance has improved.

So far on the svm.pred.radial.grid model, gives the most accurate prediction results. However, Random Forest models are another method of prediction that tend to have highly accurate classification results, and as such will be manipulated next.

Random Forest

Caret was used to train a model using the Random Forest method.

To use the Random Forest model, the train() method's "method" parameter must be set to "rf" from the previous svmRadial. The RF model will then select the appropriate mtry value based on accuracy to reveal the diagnostic ability of the model; that is, does it diagnose (predict) accurately. Following an automatic optimization of the model is a manual tuning leveraging the tuneGrid approach to try and improve model performance.

In running this model, note that the same repeated cross validation controls as all of thje previous models has been used, with the addition of classProbs. classProbs is a logical control that allows indication of whether class probabilities should be computed during classification along with predicted values in each resample run. Since the RF method is dependent upon various probabilities of multiple resamples, the classProbs() method must be set to TRUE, where its default was FALSE for the previous models.

The model results are as follows:

```
## Random Forest
##
## 4331 samples
##   12 predictor
##    2 classes: 'bad', 'good'
##
## Pre-processing: centered (12), scaled (12)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 3898, 3898, 3898, 3898, 3898, 3897, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.8805537  0.5612422
##    3    0.8803237  0.5660164
##    4    0.8791695  0.5650532
##    5    0.8790928  0.5667277
##    6    0.8784763  0.5660125
##    7    0.8783204  0.5653053
##    8    0.8765506  0.5592438
##    9    0.8762434  0.5587958
##   10    0.8758574  0.5589288
##   12    0.8746272  0.5551558
```

Here is shown that accuracy was used to select the optimal model, as noted above, and that the model settled on a mtry value of 2. Using this model to predict agains the entire test data set, and record the accuracy, it is shown as follows:

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  bad good
##       bad  1680  209
##       good   52  225
##
##                Accuracy : 0.8795
##                  95% CI : (0.865, 0.8929)
##     No Information Rate : 0.7996
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.565
##  Mcnemar's Test P-Value : < 2.2e-16
```

```
##
##             Sensitivity : 0.9700
##             Specificity : 0.5184
##          Pos Pred Value : 0.8894
##          Neg Pred Value : 0.8123
##              Prevalence : 0.7996
##          Detection Rate : 0.7756
##    Detection Prevalence : 0.8721
##       Balanced Accuracy : 0.7442
##
##        'Positive' Class : bad
```

Model accuracy is reported at 87.95%, which is easily the best performing model so far. Now the model can be tuned using the caret package to assess for futher improved performance.

Tuning will commence on one parameter: mtry. mtry, along with ntree, are widely recognized as the two factors that will have the biggest impact on a RF model.

Direct from the help page for the randomForest() function in R, mtry is the "Number of variables randomly sampled as candidates at each split". Ntree is the "Number of trees to grow".

To tune, a new model is created using the same control and tuneLength as in previous models, but this time mtry=floor(sqrt(ncol(wine.train))) or mtry=3 – set manually.

The model runs with mtry constat at 3, and accuracy output reports as follows:

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  bad good
##       bad  1669  204
##       good   63  230
##
##                Accuracy : 0.8767
##                  95% CI : (0.8621, 0.8903)
##     No Information Rate : 0.7996
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.562
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9636
##             Specificity : 0.5300
##          Pos Pred Value : 0.8911
##          Neg Pred Value : 0.7850
##              Prevalence : 0.7996
##          Detection Rate : 0.7705
##    Detection Prevalence : 0.8647
##       Balanced Accuracy : 0.7468
##
```

```
##        'Positive' Class : bad
##
```

87.9% accuracy is similarly obtained.

Decision Tree

For training a Decision Tree classifier using caret, the train() method should be passed with "method" parameter set to "rpart". There is another package "rpart" that is specifically available for decision tree implementation. Accordingly, caret is a bit simpler because it links its train function with others for a more "plug and play" type approach.

For the decision tree model, the target variable quality is passed, as in previous models. The "quality~." once again denotes the formula for using all attributes in the classifier as predictors. The "trControl" parameter will be passed with results from our trianControl() method similar to previous models.

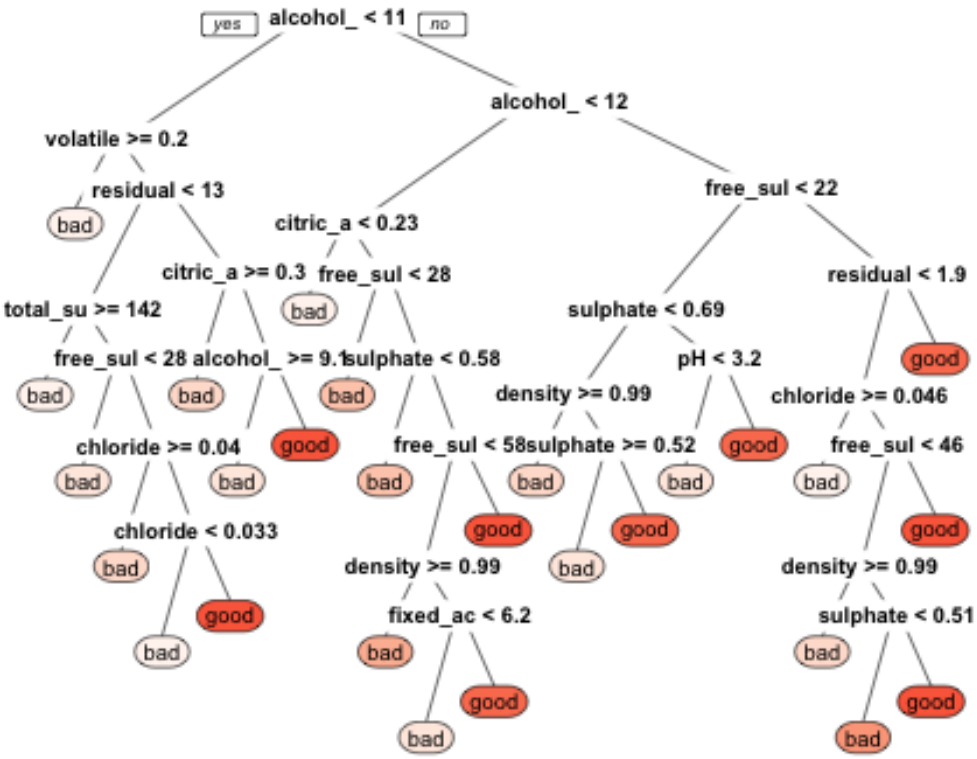
In terms of tuning our model, research shows we can use different criteria while splitting our nodes of the tree as a way to tune our model.

To select the specific strategy, the parameter "parms" is passed into the train() method. It should contain a list of parameters for the rpart method. For splitting criteria, a "split" parameter with values either "information" for information gain or "gini" for gini index must also be added. Both information gain and gini index to find the most accurate model are analyzed herein.

Below is shown the result of the initially trained decision tree model by printing out the dtree.fit variable. It shows the accuracy metrics for different values of the cp; the complexity parameter of the tree.

```
## CART
##
## 4331 samples
##   12 predictor
##    2 classes: 'bad', 'good'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 3899, 3898, 3898, 3897, 3898, 3899, ...
## Resampling results across tuning parameters:
##
##   cp           Accuracy   Kappa
##   0.004744958  0.8302984  0.3557012
##   0.005931198  0.8295302  0.3348221
##   0.006326611  0.8297606  0.3355760
##   0.006820878  0.8293753  0.3335772
##   0.007117438  0.8293753  0.3335772
##   0.008303677  0.8291413  0.3366335
##   0.013048636  0.8193654  0.2653197
##   0.016607355  0.8158271  0.2422359
##   0.020166074  0.8145943  0.2433389
##   0.025504152  0.8098205  0.1679176
```

```
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.004744958.
```



Prediction of Decision Tree on Test data

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  bad good
##       bad  1611  264
##       good  121  170
##
##                Accuracy : 0.8223
##                  95% CI : (0.8055, 0.8381)
##     No Information Rate : 0.7996
##     P-Value [Acc > NIR] : 0.00419
##
##                   Kappa : 0.3672
##  Mcnemar's Test P-Value : 4.588e-13
##
##             Sensitivity : 0.9301
##             Specificity : 0.3917
##          Pos Pred Value : 0.8592
```

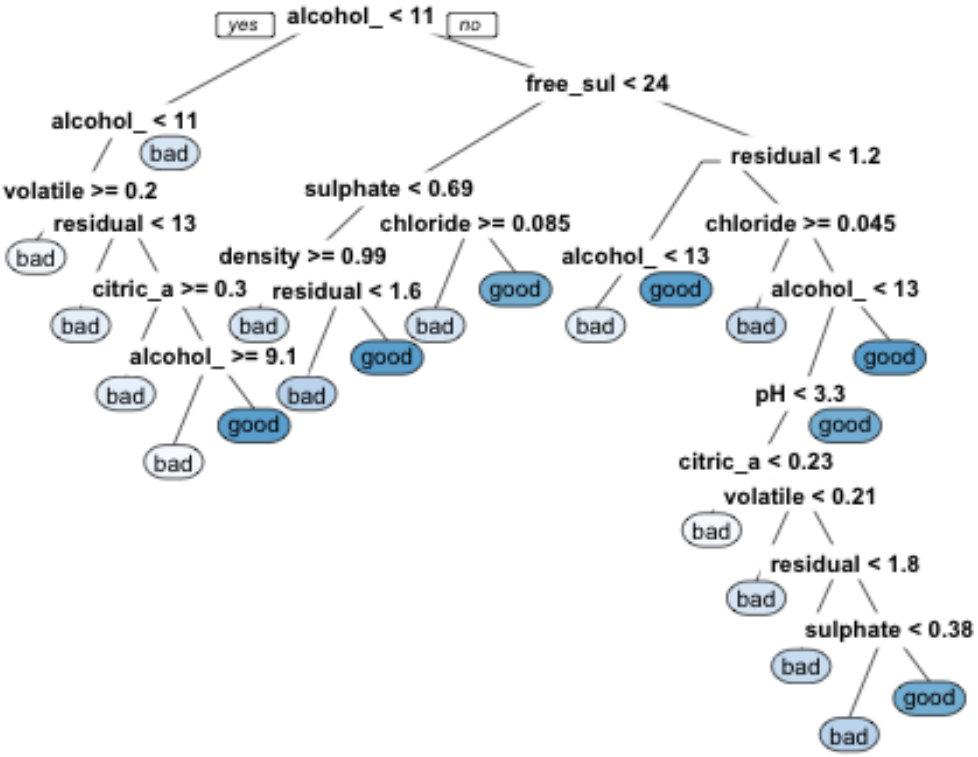
```
##          Neg Pred Value : 0.5842
##              Prevalence : 0.7996
##          Detection Rate : 0.7438
##    Detection Prevalence : 0.8657
##       Balanced Accuracy : 0.6609
##
##        'Positive' Class : bad
```

The above results show that the classifier with the criterion as information gain is giving 82.23% of accuracy for the test set.

Training the Decision Tree Classifier with criterion as gini index

Next is programmed a decision tree classifier using the splitting criterion of gini index (versus the previous information gain). This model's results show accuracy metrics for different values of the cp; the complexity parameter of the tree.

```
## CART
##
## 4331 samples
##   12 predictor
##    2 classes: 'bad', 'good'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 3899, 3898, 3898, 3897, 3898, 3899, ...
## Resampling results across tuning parameters:
##
##   cp           Accuracy   Kappa
##   0.004744958  0.8289826  0.3616752
##   0.005931198  0.8302182  0.3503554
##   0.006326611  0.8306792  0.3507520
##   0.006820878  0.8302187  0.3485428
##   0.007117438  0.8300648  0.3453694
##   0.008303677  0.8272936  0.3293491
##   0.013048636  0.8174423  0.2785669
##   0.016607355  0.8137475  0.2542771
##   0.020166074  0.8118971  0.2297410
##   0.025504152  0.8076646  0.1323638
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.006326611.
```



Prediction using Gini Index Decision Tree

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  bad good
##       bad  1649  286
##       good   83  148
##
##                Accuracy : 0.8296
##                  95% CI : (0.8131, 0.8452)
##     No Information Rate : 0.7996
##     P-Value [Acc > NIR] : 0.000211
##
##                   Kappa : 0.3554
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9521
##             Specificity : 0.3410
##          Pos Pred Value : 0.8522
##          Neg Pred Value : 0.6407
##              Prevalence : 0.7996
##          Detection Rate : 0.7613
##    Detection Prevalence : 0.8934
```

```
##       Balanced Accuracy : 0.6465
##
##        'Positive' Class : bad
```

The model performance achieved 82.96% accuracy, which is an improvement over the information gain model, but does not outperform the original Random Forest model run previously.

Conclusion

In summation, various models were run to predict wine quality using 13 predominantly chemical and quantitative features. Models run include KNN (k-nearest neighbor), Support Vector Machine (both linear and radial), Random Forest, and Decision Tree (both using information gain criterion and gini index criterion).

Though none of the models were able to produce more than 87% accuracy in predicting the test set results, the random forest model was able to be tuned using the number of samples taken at each node split (mtry, discussed previously) to achieve the highest prediction accuracy of 87.95%.

Accordingly, the model rf.fit should be used for predicting wine quality.

Nathaniel Hoffelmeyer, Kirby Hood
IST 565 Data Mining

Abstract

Attribute Descriptions:

fixed acidity: most acids involved with wine or fixed or nonvolatile

volatile acidity: the amount of acetic acid in wine, which at too high of levels can lead to an unpleasant, vinegar taste, should be very low or not able to be detected.

citric acid: found in small quantities, citric acid can add ‘freshness’ and flavor to wines

residual sugar: the amount of sugar remaining after fermentation stops, it’s rare to find wines with less than 1 gram/liter and wines with greater than 45 grams/liter are considered sweet

chlorides: the amount of salt in the wine

free sulfur dioxide: the free form of SO2 exists in equilibrium between molecular SO2 (as a dissolved gas) and bisulfite ion; it prevents microbial growth and the oxidation of wine

total sulfur dioxide: amount of free and bound forms of S02; in low concentrations, SO2 is mostly undetectable in wine, but at free SO2 concentrations over 50 ppm, SO2 becomes evident in the nose and taste of wine

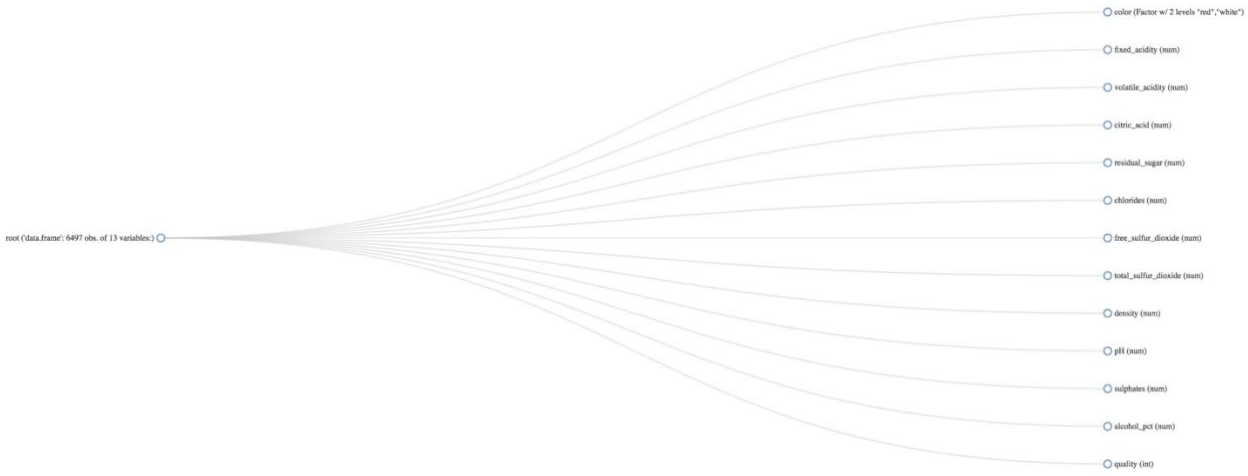
density: the density of water is close to that of water depending on the percent alcohol and sugar content

pH: describes how acidic or basic a wine is on a scale from 0 (very acidic) to 14 (very basic); most wines are between 3-4 on the pH scale

sulfites: a wine additive which can contribute to sulfur dioxide gas (S02) levels, wich acts as an antimicrobial and antioxidant

alcohol: the percent alcohol content of the wine

quality (score between 0 and 10 given by wine experts) – Output variable



Sources:

C. (n.d.). © 2018. Retrieved June 10, 2018, from http://www.vinhoverde.pt/en/homepage Vinho Verde wine region

C. (n.d.). © 2018. Retrieved June 10, 2018, from http://www.vinhoverde.pt/en/history-of-vinho-verde Vinho Verde history

Hoffower, H., & Gal, S. (2018, June 8). 15 memorable Anthony Bourdain quotes that show why the celebrity chef and author was so beloved. Retrieved June 18, 2018, from http://www.businessinsider.com/anthony-bourdain-best-quotes-food-travel-life-2018-6

Nierman, Doug. (2004) Fixed Acidity. Retrieved June 10, 2018 from http://waterhouse.ucdavis.edu/whats-in-wine/fixed-acidity

Wine Quality Data Set http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/

Unexecuted Code:

```
Read in the data set
wine <- read.csv("~/Desktop/Education/Syracuse University/Spring 2018/IST 565/Final Project/winequality-allcolors.csv")

############## Exploratory Data Analysis ####################

# Load in Data Explorer
library(DataExplorer)

# The first thing we will do in EDA is checking the dimension of the input dataset and the time of variables.
plot_str(wine)

# Summary of all the attributes
summary(wine)
str(wine)

# Are there any missing values? Fortunately NO :)
plot_missing(wine)

# Analyze represent continuous vars with histogram
plot_histogram(wine)
# Maybe you're a fan of density plots --> we can do that too!
plot_density(wine)

# Multivariate Analysis
# Correlation
plot_correlation(wine, type = 'continuous','Review.Date')

# Categorical variables
```

```
plot_bar(wine)

# Create a nice sharable report
create_report(wine)

# Using Caret to model and predict quality
## Install the Caret Package
library(caret)

#check for NA values
anyNA(wine)

######################## DISCRETIZE ###################################

summary(wine$quality) # look at the distribution of the quality variable
# Discretize quality
min_qual = 0
max_qual = 10
bins = 2
width = 3
wine$quality <- cut(wine$quality, breaks = 2, labels = c("bad", "good"))
str(wine)
anyNA(wine)
summary(wine$quality)
#View(wine)
#is.na(wine$quality)

##################### DATA SLICING ####################
# First let's set a Training and Testing set
set.seed(313)
wine.index <- sample(6497, 4331)
```

```
wine.train <- wine[wine.index, ]
wine.test  <- wine[-wine.index, ]

dim(wine.train); dim(wine.test)

##################### KNN Model ######################
trctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
set.seed(313)
knn_fit <- train(quality ~., data = wine.train, method = "knn",
                 trControl=trctrl,
                 preProcess = c("center", "scale"),
                 tuneLength = 10)

knn_fit # view the model results
plot(knn_fit) # plot model
knn.pred <- predict(knn_fit, newdata = wine.test) #predict test data set
#knn.pred #View predicted values (optional)
# How Accurate?
confusionMatrix(knn.pred,wine.test$quality)

# second KNN model
set.seed(313)
knn_fit2 <- train(quality ~
fixed_acidity+volatile_acidity+citric_acid+residual_sugar+chlorides+free_sulfur_dioxide+
                    total_sulfur_dioxide+density+pH+sulphates+alcohol_pct,
                  data = wine.train, method = "knn",
                  trControl=trctrl,
                  preProcess = c("center", "scale"),
                  tuneLength = 10)
knn_fit2
knn.pred2 <- predict(knn_fit2, newdata = wine.test)
```

```
confusionMatrix(knn.pred2,wine.test$quality)

# third kNN model
set.seed(313)
knn_fit3 <- train(quality ~ citric_acid+residual_sugar+chlorides+
                    density+pH+sulphates+alcohol_pct+color,
                  data = wine.train, method = "knn",
                  trControl=trctrl,
                  preProcess = c("center", "scale"),
                  tuneLength = 10)
knn_fit3
knn.pred3 <- predict(knn_fit3, newdata = wine.test)
confusionMatrix(knn.pred3,wine.test$quality)

#################### LINEAR SVM Model ####################
svmctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
set.seed(313)
svmLinear <- train(quality ~., data=wine.train, method="svmLinear", trControl=svmctrl,
preProcess = c("center","scale"),
                   tuneLength = 10)
svmLinear #view the model
svm.pred <- predict(svmLinear, newdata=wine.test)
#svm.pred #view the raw predicted values (optional)
confusionMatrix(svm.pred, wine.test$quality) # accuracy

# second Linear SVM model
grid <- expand.grid(C = c(0.01, 0.5, 3))
svmLinearGrid <- train(quality ~., data = wine.train, method = "svmLinear",
                       trControl=svmctrl,
                       preProcess = c("center", "scale"),
                       tuneGrid = grid,
```

```
                tuneLength = 10)
svmLinearGrid
plot(svmLinearGrid)
svm.pred.grid <- predict(svmLinearGrid, newdata=wine.test)
confusionMatrix(svm.pred.grid, wine.test$quality)

#################### RADIAL SVM Model ####################
set.seed(313)
svmRadial <- train(quality ~., data = wine.train, method = "svmRadial",
                   trControl=svmctrl,
                   preProcess = c("center", "scale"),
                   tuneLength = 10)
svmRadial
plot(svmRadial)
svm.pred.radial <- predict(svmRadial, newdata=wine.test)
confusionMatrix(svm.pred.radial, wine.test$quality)

#second radial svm model
grid.radial <- expand.grid(sigma = c(0.01, 0.02, 0.025, 0.03, 0.04,
                                     0.05, 0.06, 0.07,0.08, 0.09, 0.1, 0.25, 0.5, 0.75,0.9),
                           C = c(0.01, 0.05, 0.1, 0.25, 0.5, 0.75,
                                 1, 1.5, 2,5))
set.seed(313)
svmRadialGrid <- train(quality ~., data = wine.train, method = "svmRadial",
                   trControl=svmctrl,
                   preProcess = c("center", "scale"),
                   tuneGrid = grid.radial,
                   tuneLength = 10)
svmRadialGrid
plot(svmRadialGrid)
svm.pred.radial.grid <- predict(svmRadialGrid, newdata = wine.test)
```

```
confusionMatrix(svm.pred.radial.grid, wine.test$quality)

#################### RANDOM FOREST MODEL ####################

rfctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 3, classProbs = TRUE)
set.seed(313)
rf.fit <- train(quality ~., data=wine.train, method="rf", trControl=rfctrl, preProcess =
c("center","scale"),
                tuneLength = 10)
rf.fit
rf.pred <- predict(rf.fit, newdata=wine.test)
confusionMatrix(rf.pred,wine.test$quality)

#second Random Forest model
metric <- "Accuracy"
mtry <- floor(sqrt(ncol(wine.train)))
tunegrid <- expand.grid(.mtry=mtry)
set.seed(313)
rf.fit2 <- train(quality ~., data=wine.train, method="rf",
                 metric=metric,
                 tuneGrid=tunegrid, trControl=rfctrl,
                 preProcess = c("center","scale"),
                 tuneLength = 10)
rf.fit2
rf.pred2 <- predict(rf.fit2, newdata=wine.test)
confusionMatrix(rf.pred2,wine.test$quality)

#################### DECISION TREE MODEL ####################
dtctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
set.seed(333)
dtree.fit <- train(quality ~., data = wine.train, method = "rpart", parms = list(split = "information"),
```

```
                trControl = dtctrl,
                tuneLength = 10)
dtree.fit
library(rpart.plot)
prp(dtree.fit$finalModel, box.palette = "Reds", tweak = 1.2) #graph decision tree
dtree.pred <- predict(dtree.fit, newdata = wine.test)
confusionMatrix(dtree.pred, wine.test$quality)

#change information to gini
set.seed(333)
dtree.fit.gini <- train(quality ~., data = wine.train, method = "rpart", parms = list(split="gini"),
                        trControl=dtctrl,
                        tuneLength=10)
dtree.fit.gini
prp(dtree.fit.gini$finalModel,box.palette = "Blues", tweak = 1.2) #plot decision tree
dtree.pred.gini <- predict(dtree.fit.gini, newdata = wine.test)
confusionMatrix(dtree.pred.gini,wine.test$quality)
```