PART A: ARTIFICIALLY INTELLIGENT WINE TASTING

Task A2:

```
#=======================
#   TASK A2
#=======================

#Import Red Wine Dataset

winequality.red <- read.csv("~/Desktop/UCL Modules/Selected Topics in Statistics/ICA 2/Q1/Data/winequality/winequality-red.csv", sep=";")

#Rename dataset
red.wine <- winequality.red

#Add new color column
red.wine$color <- "red"

#-------------------
#Import White Wine Dataset

winequality.white <- read.csv("~/Desktop/UCL Modules/Selected Topics in Statistics/ICA 2/Q1/Data/winequality/winequality-white.csv", sep=";")

#Rename Dataset
white.wine <- winequality.white

#Add new color column
white.wine$color <- "white"
#----------------------

#Create Joint Dataset

wine.set <- rbind(red.wine, white.wine)

#============================
```

Task A1, A3, A4, A5, A6:

Executive Summary

The vision of our food startup is to create artificial intelligence with capabilities of tasting and appraising food, with a long-term outlook towards creating an AI, which can cook and create its own cuisine. The first food group our company has gathered extensive data on is white and red wines. Our dataset contains 11 chemical components of the wine along with a sensory quality variable taking discrete values from 0 to 10.

This report is divided into 4 main sections. The first section looks at the feasibility and application of this dataset towards our AI acquiring the ability to taste wine. The following section does exploratory data analysis to get an intuition on the structure of the variables and their relationship with one another. In the third section machine learning models are trained on our dataset and their performance is compared in a benchmarking study. The final section conducts further analysis by modifying our dataset, introducing new training models, and empirically investigating the questions discussed in the first section given the limitations established.

The main conclusions from this report are the following:

- The wine quality dataset collected has limited scope in terms of the company's long-term objective to create an AI that can produce and taste wine at a high level. However, it can be used to establish the taste preferences of sommeliers and the existence of relationship between chemical composition and quality.
- To this end, modern machine learning tools were benchmarked in a classification and regression task to see which algorithm does the best prediction. It was observed that xgboost is the best performer in classification task and random forest is best at regression task.
- Wine colour seems to play an important role in determining quality. This raises the question if the sommeliers were aware of the quality before tasting thus biasing the result or were they completely unaware of this fact.
- The results of hypothesis test show that quality has significant relationship with respect to chemical composition and cannot be overlooked entirely as human bias. This could be an important exploratory conclusion towards company's long-term objectives.

Introduction

Due to the great leaps in computational technology and know-how, machines are beginning to learn more than just trivial tasks. Indeed, the limits of their capabilities seem boundless. However, their ability to learn has a major glass ceiling - quality of date available. The dataset collected contains results of chemical composition of wines along with the median sensory score of 3 sommeliers on a subjective scale of 0 to 10. However, it does not contain the results of the extensive chemical testing and non-expert gustatory analysis, which severely restricts our ability to generalise our results and answer inferential questions.

In light of this, our CTO and others have raised important questions on the extent of learning that can be achieved by our AI on the collected wine samples. This section of the report will explicitly state the limitations of our dataset and outline the questions it can answer before any further analysis is done.

1) *Can we use the sommelier/wine data to create an AI with super-human performance in wine tasting?*

The data collections report states, "no expense was spared in tracking down the most accomplished sommeliers and tracking down the highest quality wines hiring the former to taste the latter". The implication of this is that the dataset obtained here is not random. The dataset is a sample of "quality wines" rather than wines in general, and any categorisation by sommeliers is creating a hierarchy within "quality wines". We do not know the composition of wines not included in the dataset and why they were classified as not being "quality" to begin with.

Subsequently, two philosophical question arise here. First, who created the initial definition of "quality" if the only thing sommeliers are doing is picking the best amongst the best? Second, what is the definition of super-human performance given that we cannot determine from our dataset what separates an expert performance in wine tasting from a non-expert, and a super-human from an expert? Hence, this question is out of scope for the dataset provided.

2) *Which components of wine make a wine a good wine?*

This question can be answered if we include certain caveats and rephrase the question as, "Which components of wine *in this dataset* make a good wine *for the sommelier".* To get a more general result on which components of wine make a good wine we would need the results of project "AI gustometer" which contains results of chemical analysis and non-expert gustatory analysis.

3) *Can the AI use the data to create the perfect wine, I.e., wine whose quality exceeds all that we have seen?*

We cannot make an inference beyond the given dataset without further experiment, assumptions or assertions. For example, if we assume that each variable is i.i.d with some consistent and predictable relation to quality (say, linear or convex), we could potentially create a wine that exceeds all that we have seen within the dataset. However, this assumption is unrealistic as our variables are not independent of one another. Hence, this question cannot be answered from the dataset provided.

4) *Whether human perception of wine quality is all but subjective, i.e. , perhaps there is no empirically verifiable correlate of "good" or "bad" wine. For any such perception could be entirely a result of human biases, incompetence, unintentional priming, and self-delusion, based on say high quality label on the wine bottle, the price tag or an authoritative expert opinion. And if so, what would it be that AIs could or would learn from humans?*

This question can be answered to an extent from our dataset. The general premise of this question is; can the data provided be considered empirical proof of correlation between variables and quality or is the relationship based entirely on random human idiosyncrasies. One simple way to see how our variables correlate with quality is through hypothesis testing and assessing the confidence intervals. This would tell us how significant is the explanatory power being lent to the quality indicator by our independent variables. In case the relationship is completely idiosyncratic to human behaviour, we would see low confidence intervals (high p-values) on our predictors. Hence, we can find out if our variables show *statistically* significant relation to good

and bad wines in the dataset. But to verify this result we may need to conduct further experimentation such as a/b testing on chemical components.

The aim of this report is to conduct a benchmarking study to see how well median wine quality (as determined by sommelier) can be predicted from chemical composition based on different machine learning techniques. It also determines if wine colour adds explanatory power over and above chemical composition or vice-versa. Finally, we conduct a hypothesis test to determine if a causal relation can exist between chemical composition and quality of wine.

Exploratory Data Analysis

Multiple levels of exploratory data analysis has been conducted with the aim of getting an intuition on how our variables are distributed amongst red and white wine, and subsequent relation quality. For visualization purposes the quality variable has been grouped into a bucket of 3 later in the analysis. However, this approach is abandoned when modeling our dataset. Some of the visualization techniques have been inspired from kaggle kernels used on a different wine dataset with different set of variables.[1] For visualization in higher dimension T-Sne was also implemented.

Univariate Summaries:

To start our exploration we begin with plotting densities of each variable in our dataset separately for white and red wine, and comparing it to the respective 5 number summaries which covers maximum and minimum values, mean, median, and quantile information. The full table containing 5 number summaries for the variables is in the appendix. The important take away from those tables are:

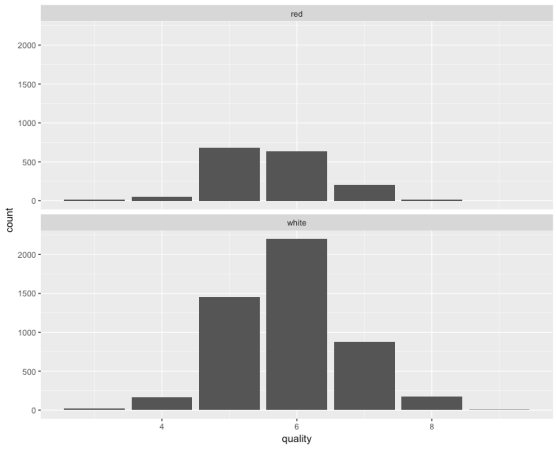
- Our dataset is unbalanced having more white wine (4898) than red wine (1599)
- All values are positive.
- There are no missing values or confounding values that should not theoretically exist.

Univariate Density Plots:

In all plots red wine is above and white wine is below.

1) Quality

```
wine.set$color: red
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  3.000   5.000   6.000   5.636   6.000   8.000
------------------------------------------------
wine.set$color: white
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  3.000   5.000   6.000   5.878   6.000   9.000
```

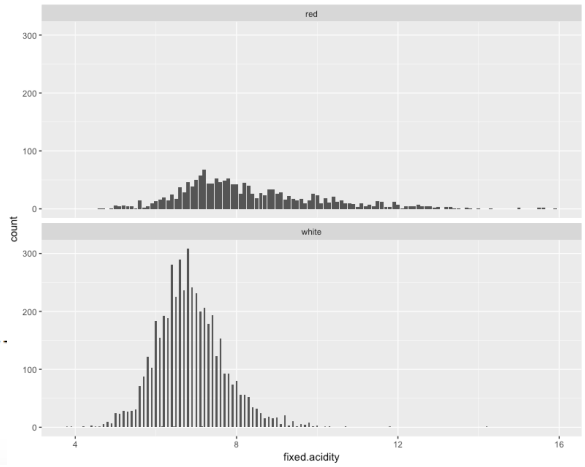


[1] In particular:
https://www.kaggle.com/meepbobeep/intro-to-regression-and-classification-in-r (corrgram)
https://www.kaggle.com/danielpanizzo/red-and-white-wine-quality (variable plots)
https://www.kaggle.com/sagarnildass/red-wine-analysis-by-r (multivariate plots)

We see that quality follows a normal distribution for both red and white wine with the latter showing lesser variance. Both have their median at 6 and minimum at 3. Red wine maximum is at 8 whereas white is at 9.

2) Fixed acidity

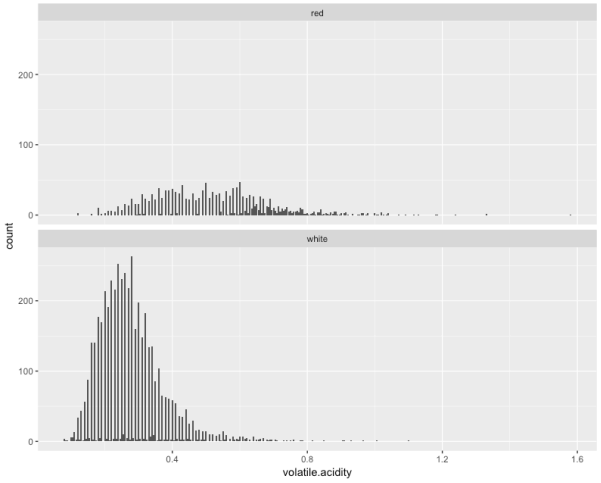
```
wine.set$color: red
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   4.60    7.10    7.90    8.32    9.20   15.90
------------------------------------------------
wine.set$color: white
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  3.800   6.300   6.800   6.855   7.300  14.200
>
```



Fixed acidity has distinct distribution for red and white wine. Red wine has a more uniform spread when compared with the normally distributed white wine. The even spread of red wine is between 1st quartile and 3rd quartile with tails showing low frequency. The mean of normal distribution of white wine is at 6.855.

3) Volatile acidity

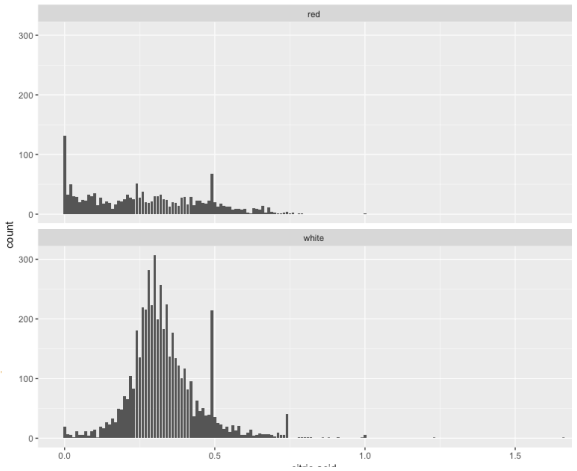
```
wine.set$color: red
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.1200  0.3900  0.5200  0.5278  0.6400  1.5800
------------------------------------------------
wine.set$color: white
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0800  0.2100  0.2600  0.2782  0.3200  1.1000
```



Volatile acidity has different distribution for red and white wine. Red wine shows a bimodal distribution spread whereas white wine shows a normal distribution that is skewed to the left with a long right tail. Median for red wine is at 0.5200 units whereas white wine is at 0.2600 units. The minimum and maximum values for red wine (0.12, 1.58) are also much higher than white wine (0.800 and 1.100).

4) Citric Acid

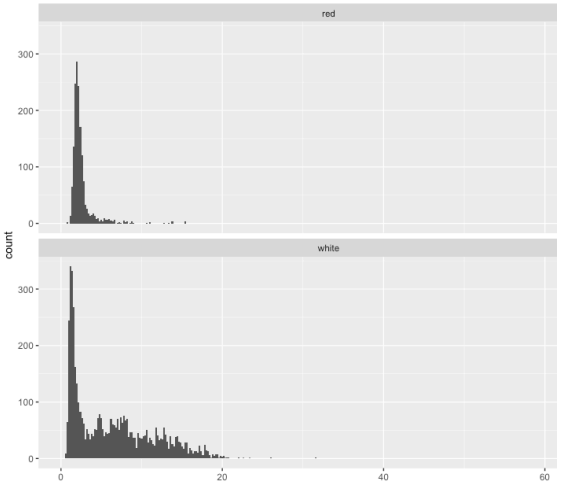
```
wine.set$color: red
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.000   0.090   0.260   0.271   0.420   1.000
------------------------------------------------------
wine.set$color: white
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.2700  0.3200  0.3342  0.3900  1.6600
```



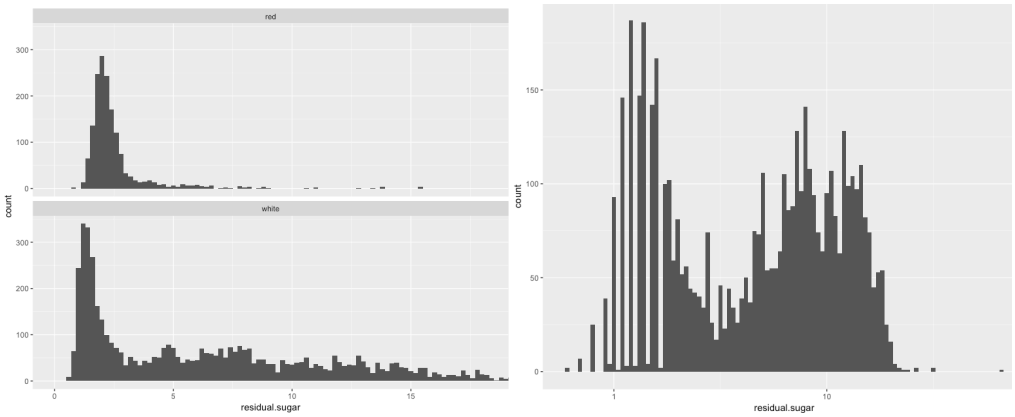
Citric acid has very distinct distribution for red and white wine. For red wine we see a uniform-ish distribution with 2 peaks at 0 and 0.48 units. White wine has a normal distribution with a single distinct peak away from the centre at 0.48 units as well. Attempt to log transform the data to form normality, however this did not work.

5) Residual Sugar

```
wine.set$color: red
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.900   1.900   2.200   2.539   2.600  15.500
------------------------------------------------------
wine.set$color: white
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.600   1.700   5.200   6.391   9.900  65.800
```



Residual sugar for white wine seems to have a massive outlier at around 70 units. To get a better view of distribution we omit this data point.

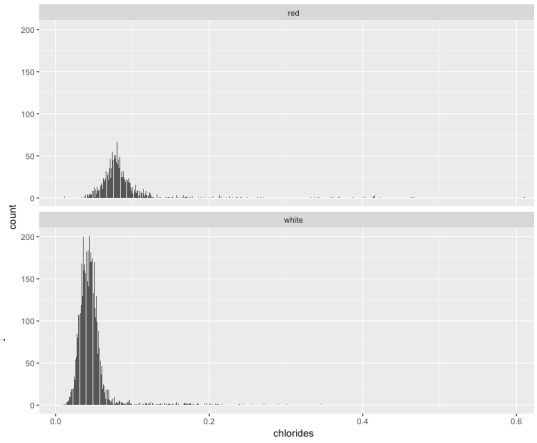


We see that both red and white wine have an asymmetric distribution with white wine having more populous tail. To understand this tail effect better I log transformed white wine using the log10 scale on the x-axis and replotted the graph.

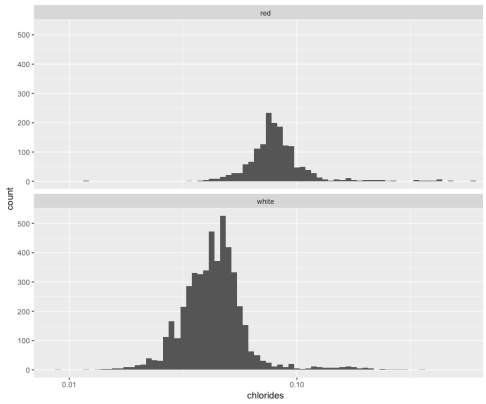
We now observe a bimodal distribution for white wine in log-10 scale with the first distribution being between 1 and 5 and the second between 6 and 13.

6) Chlorides

```
wine.set$color: red
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.01200 0.07000 0.07900 0.08747 0.09000 0.61100
------------------------------------------------------------
wine.set$color: white
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.00900 0.03600 0.04300 0.04577 0.05000 0.34600
```

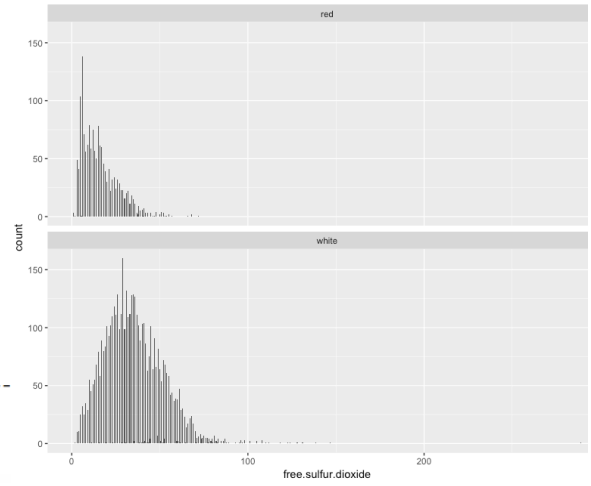


Both red and white wine has normal distributions with mean (0.087 and 0.045). Since both red and white wine have a long right tail, I use the long transformation to base 10 again to explore this tail. The peak of the white wine distribution has become less narrow and the centres have moved closer together due to change in scale. Log transformation has made the two distributions more similar.



7) Free Sulphur-Dioxide

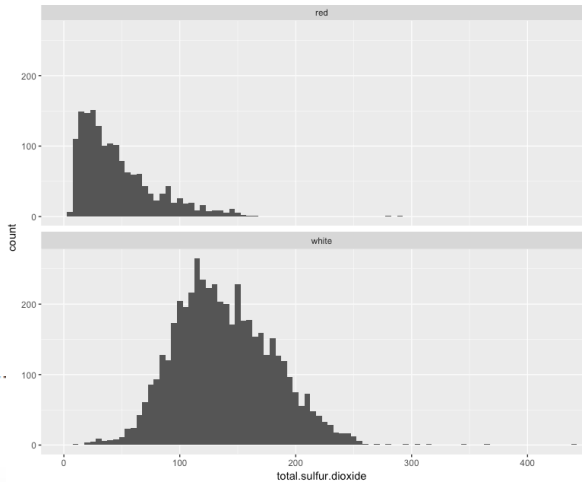
```
wine.set$color: red
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    7.00   14.00   15.87   21.00   72.00
------------------------------------------------
wine.set$color: white
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   2.00   23.00   34.00   35.31   46.00  289.00
>
```



Red wines have a left skew between 1 and 72 units. White wines have a normal distribution between 1 and 100 mg/dm^3 with a long tail and outlier at 289 units. The median value of red wine is 14 units whereas white is 34 units.

8) Total Sulphur Dioxide

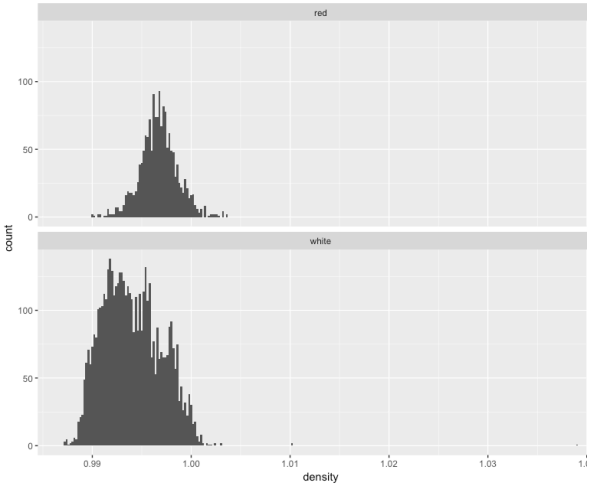
```
wine.set$color: red
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   6.00   22.00   38.00   46.47   62.00  289.00
------------------------------------------------
wine.set$color: white
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    9.0   108.0   134.0   138.4   167.0   440.0
>
```



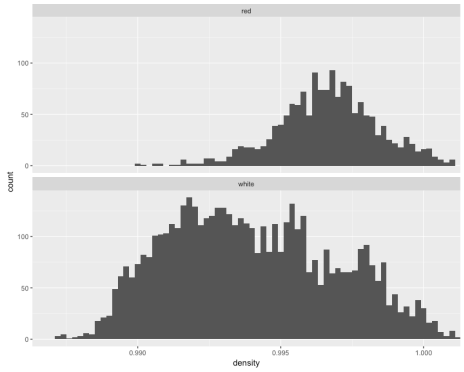
This has a similar distribution to free sulphur-dioxide. Red wines have a left skew and white wine are more normally distributed. White wine has a median at 134 units whereas red has median at 38 units. The skewness means that there is heavy variation between the mean and median of red wine. White wine has a massive outlier at 440 units and red at 289 units.

9) Density

```
wine.set$color: red
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.9901  0.9956  0.9968  0.9967  0.9978  1.0037
------------------------------------------------
wine.set$color: white
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.9871  0.9917  0.9937  0.9940  0.9961  1.0390
```

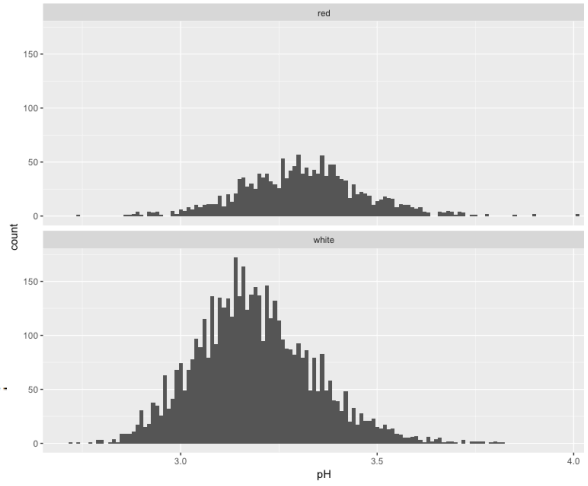


Both red and white wine have a normal distribution with large outliers. To get a better understanding of our distribution we will omit them. Red wines have bulk of their mass between 0.99 and 1.004 units with the bulk of the mass near 0.996 units. White wine has a more spread distribution with mass allocated between 0.990 and 1.000.



10) pH

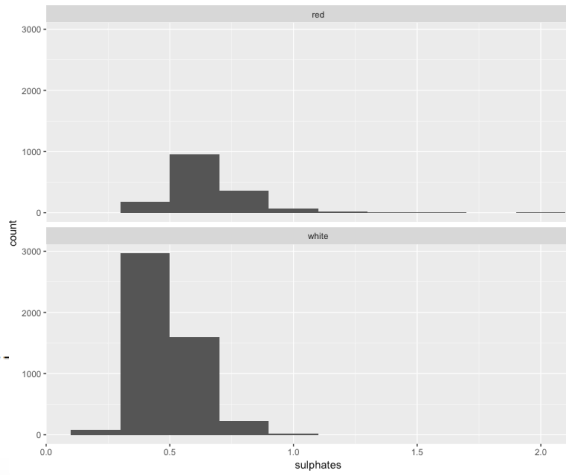
```
wine.set$color: red
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.740   3.210   3.310   3.311   3.400   4.010
------------------------------------------------
wine.set$color: white
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.720   3.090   3.180   3.188   3.280   3.820
```



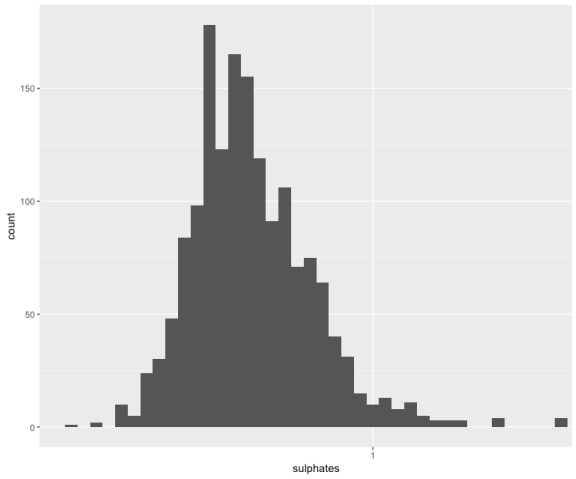
Both wines have a normal distribution with white wine having a lower variance than red wine. White wine's mean is slightly to the left of red wine (3.188 and 3.311).

11) Sulphates

```
wine.set$color: red
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.3300  0.5500  0.6200  0.6581  0.7300  2.0000
------------------------------------------------
wine.set$color: white
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.2200  0.4100  0.4700  0.4898  0.5500  1.0800
>
```



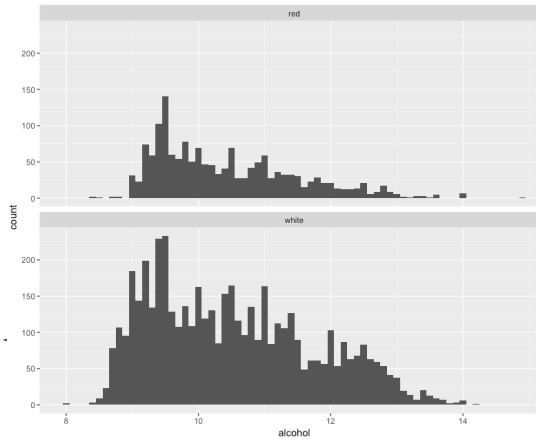
Red wine is showing a long right tail due to its outlier at 2.000. We will log transform it to get a better visualization. White wine has a normal distribution with no outliers.



After transformation red wines' normal distribution is more evident.

12) Alcohol

```
wine.set$color: red
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   8.40    9.50   10.20   10.42   11.10   14.90
------------------------------------------------
wine.set$color: white
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   8.00    9.50   10.40   10.51   11.40   14.20
```



Both red and white wine has asymmetric distribution and high degree of variance around the mean. The bulk of red wine is at alcohol concentration 9.5%. White wine sees multiple towers at different intervals between 9% and 10%.

These univariate plots seem to indicate that white wine has far more symmetric distribution with respect to each variable than red wine. This could be due to our unbalanced dataset. As white wine is far better represented here than red wine, its normal distribution is more pronounced. It is now worth checking how these variables relate to one another through bivariate plots.

Bivariate Plots

The following bivariate plots assess the correlation between variables for red and white wine. Since our main goal is to determine the relation between all variables with quality, we will focus on the last row and column.

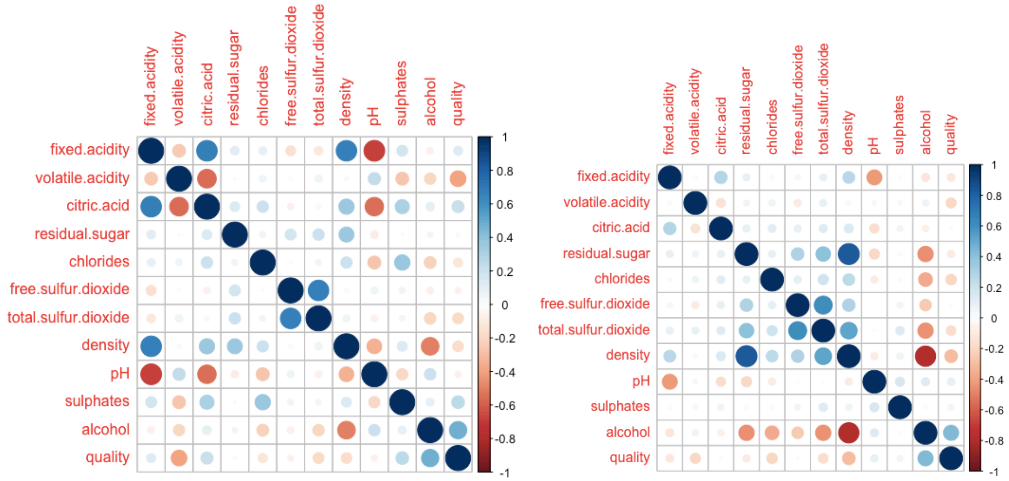


Figure 18 and 19: Correlation Table for White Wine (Right) and Red Wine (Left)

In the correlation plot for red wine we see that quality has highest correlation (by Pearson) with alcohol (0.4762) and volatile acidity (-0.391). White wine quality also has a strong correlation with alcohol (0.4356) but does not have a good correlation with volatile acidity (-0.1947). Instead, we see it have a better correlation with density (-0.3071).

Another way to visualize the above correlation graph is through a custom made correlogram. The correlogram designed below shows the intensity of correlation between variables in the lower triangle of the correlogram matrix, similar to the one above. However, in the triangle above the diagonal we plot correlation ellipses which give u an indication of the variance between two variables along with the trend line. These are called correlation ellipse.

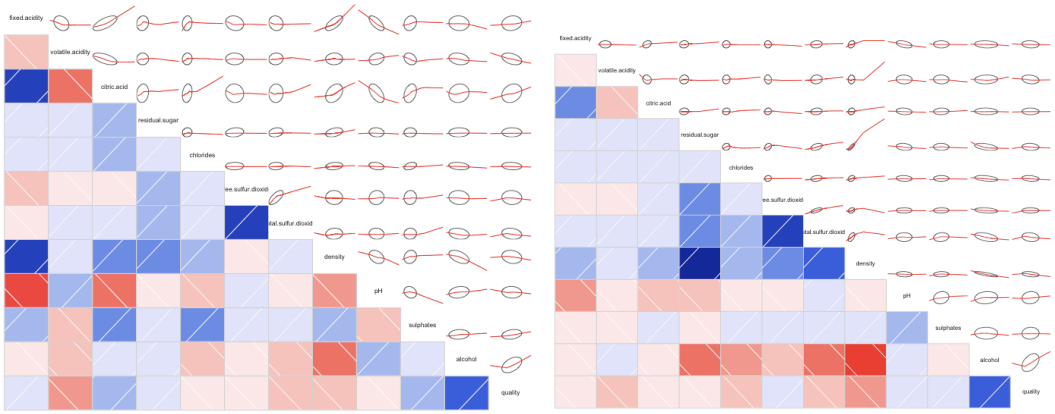


Figure 20 and 21: Corrgram for White Wine (Right) and Red Wine (Left)

For red wine we see that quality and alcohol have an interesting relationship that isn't visible simply from the correlation graph above. We see that the relation is linear and upward sloping but not all the way through. There is a slight kink downwards before moving up. The variance is also quite high. This pattern is repeated in white wine as well. For white wine, density vs quality plot only shows a slight downward curve but the variance is much lower compared to quality vs alcohol plot.

Multivariate plots

The result from our bivariate plots showed that there are two variables that are strongly linked with quality for red (alcohol content, volatile acidity) and white wine (alcohol, density). It is worth plotting the relationship between these two variables and quality to test if we see some sort of a trend or clustering. For easier visualization (only), the quality of wine was divided into three buckets - Poor (1-4), Average (5, 6), Great (7-10) and the plotting was done only for the poor and great buckets. The idea is to visualize if a trend line can determine the relationship with any confidence between the two extreme buckets.

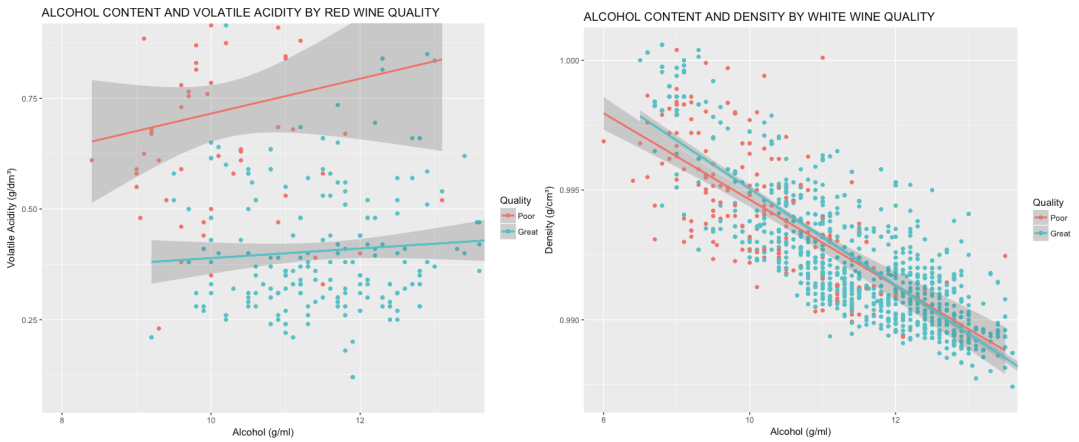


Fig 22 and 23: Result of Multivariate Plot Visualization – Red Wine (Left) White Wine (Right)

The plot scatter in red wine seems to show more of a clustering trend with high quality wine predominantly having lower volatile acidity and high alcohol content. White wine shows a stronger linear trend, however the trend line for poor and high quality wines are not well differentiated. However, a cluster relationship is still quite informative with low quality wines predominating the region with high density and low alcohol content and the great quality wines doing the opposite.

One interesting visualization technique to look at multivariate datasets is known as tSNE. This technique allows us to look at high dimensional data in two or three dimensions. We ran the algorithm for 3 different datasets – red wine, white wine, and combined. From the following figures we can see that quality labels 5 (blue), pink (6) and yellow (7) are separable clusters in red but to a lesser degree in white and full dataset. It would be worth looking into non-linear classifiers to see if they can discriminate the quality labels.

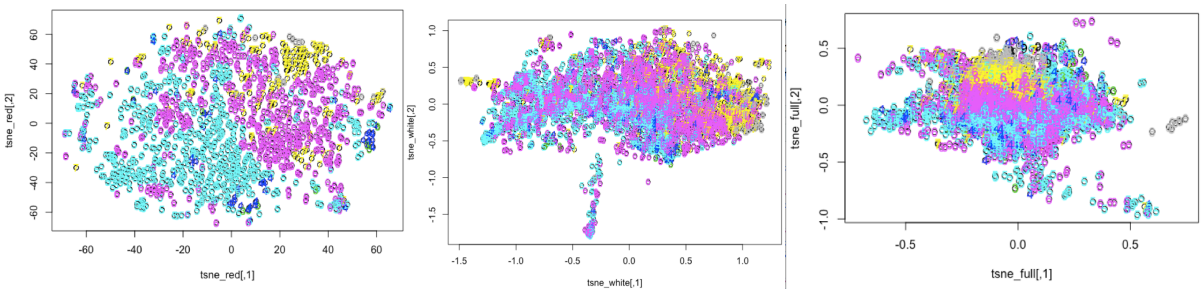


Figure 23: tsne visualization for red wine, Fig24: tsne visualization for white wine, Fig 25: tsne visualization full wine set

4. Training, Prediction, and Results of Benchmarking

The results of our exploratory analysis have given us some indication on the structure of the data. We see that our variables in most cases lack any form of linearity in their relation with the quality variable. Hence, it would be worth our resources to focus attention on non-linear classifiers like support vector machines, ensemble models, and neural networks. In addition, a dummy variable "color" has been included which is 1 if the wine is white and 0 if red to assess whether this has an impact on quality. The individual assessment on the explanatory power of colour in our model is given at the end of this section.

Although our quality variable is discrete and hence a classification model would be more appropriate, it is important to understand that taste is on a continuous spectrum. Hence our prediction task will be both classification and regression.

4.1 Training Setup

For our training setup we randomly split the dataset into two parts in the ratio 80:20. The 20% split will be kept in cold storage and used to verify the best hyperparameters obtained during the training and validation step. Since our dataset has imbalanced classes as seen from the following table:

3	4	5	6	7	8	9
30	216	2138	2836	1079	193	5

Table 13: Distribution of wine quality in our full dataset. Bulk of the observations is concentrated at 5 and 6 and barely any observations in 3 and 9.

We will make a balanced split on our dataset thus ensuring our classifier has a chance to learn and predict each label. The results of split are as follows:

3	4	5	6	7	8	9
24	173	1710	2269	863	154	4

Table 14: Train set

3	4	5	6	7	8	9
6	43	428	567	216	39	1

Table 15: Test set

4.2 Validation Setup

To validate our model a k-fold cross validation has been setup with 3 folds. K-fold cross validation is a robust procedure where data is divided into k-equal partitions with 1 partition kept for validating and the remaining for estimating the model. (Cortez, 2009) The process is repeated sequentially until all the data has had a chance to be in both the training and the validation set. Due to high computational demand, the 3-fold variant is chosen despite the 5 and 10-fold variants of the algorithm.

4.3 Models and Hyperparameters

R programming's MLR package has been used to construct the learners and tune subsequent hyperparameters in a tuning grid. For both regression and classification task the same hyperparameters have been tuned unless stated otherwise. The parameters not mentioned were kept at default setting. The details of which are enumerate below:

4.3.1 Non-Linear Support Vector Machine

SVM's are classifiers that project data points into higher dimensional space so that it is easier to discriminate between different categories. To make our SVM non-linear a radial basis function (RBF) was chosen. The gamma ($2^{(-5:5)}$) and cost parameters ($10^{(-2:3)}$) were tuned to find the best fit for the model. The gamma parameter determines the influence a single training example has with low values meaning less and high values meaning more. (RBF SVM Parameters, 2018) The cost parameter determines the bias variance tradeoff with low cost implying higher bias and high cost implying higher variance.

4.3.2 Neural Network

Neural networks are a series of connected classifiers (nodes) that attempt to identify relationships existing in a dataset through a process known as gradient descent. Neural networks are difficult to train due to the large number of hyperparameters and work effectively only on large datasets. Our dataset by neural network standards could be too small. Yet, it is worth attempting to see the results. For this report the neural network hyperparameters tuned were:

- Number of hidden layers : 3-10
- Learning rate of gradient descent (discrete values): 0.5, 1.0, 1.5 and 2.0.
- Momentum of gradient descent (discrete values): 0.5, 1.0, 1.5, 2.0

For the regression neural network only the number of nodes were changed.

4.3.3 Random Forest

Random forest are ensemble learners for classification and regression tasks that operate by constructing a multitude of decision trees at training, and output the class that is the mode of the classes (classification) or mean prediction (regression) of individual trees. (Wikipedia) The hyperparameters tuned for random forests are:

- ntree determines the number of trees to grow in each iteration of forest (discrete in sizes of 10): 10-100
- mtry is number of predictor variables to use: 1-12
- nodesize is the minimum size of terminal nodes. Setting this larger causes smaller trees to grow. Range: 1-30

4.4 Performance Measure

To see how well our models are doing we need to define our performance measure in the form of error statistic. The aim of our tuning grid is to select the model which minimizes the error statistic on the validation

set. For classification task the performance measure I have chosen is mean misclassification error (MMCE), which is:

$$MMCE = Mean\ (response \neq truth)$$

For prediction task the performance measure I have chosen is mean squared error (MSE), which is:

$$MSE = Mean\ [(response - truth)^2]$$

The result of our performance on the validation set can be visualized with the help of a violin plot shown below:

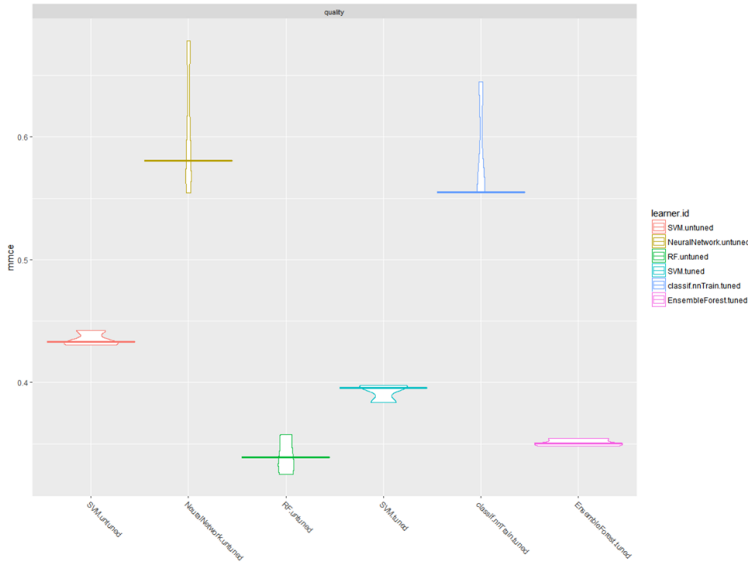


Fig 26: Violin plot of classification task showing misclassification error. Neural network shows high variation in error based on the validation split for both the tuned and un-tuned version. It also performs poorly overall (60% tuned and 58% un-tuned mmce). The unturned random forest is the best performer (34% mmce) but the tuned random forest shows far lower variation.

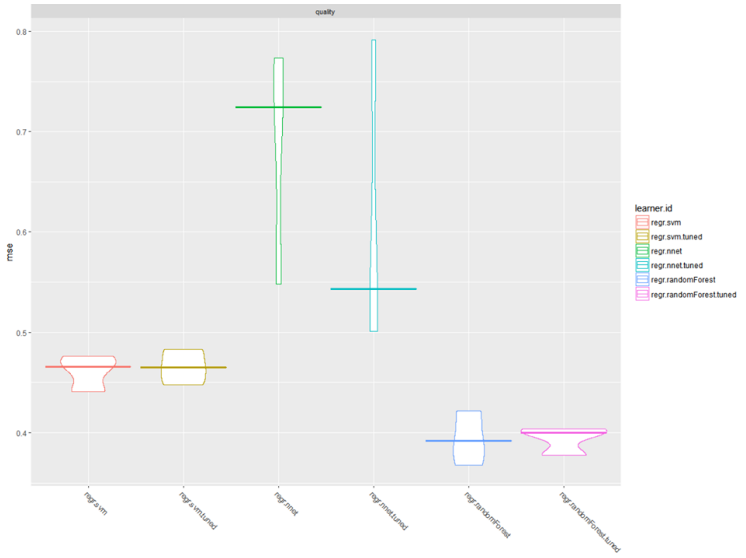


Fig 27: Violin plot of regression task showing mean square error. Neural network again shows high variation in error based on validation split for both tuned and un-tuned version. It also performs poorly overall (68% unturned and 61% unturned MSE). Random forest un-tuned performs marginally better than tuned version but the un-tuned version has lower variance in result.

4.5 Best Parameters and Prediction Results

Having established our performance measure we can now choose the best hyperparameters for our models from the validation set results. The best-tuned models for our classification and regression task are the following:

Model	Parameters	
	Classification	**Regression**
Support Vector Machines	Cost = 10, Gamma = 1	Cost = 1, Gamma = 0.125
Neural Network	Hidden = 5, Learning Rate = 1, Momentum = 0.5	Size = 6
Random Forest	Ntree = 80, Mtry = 2, Nodesize = 1	Ntree = 100, Mtry = 3, Nodesize = 1

Table 16: Best Model Hyperparameters for Classification and Regression Task

We now bring the test set out of cold storage and run our best-tuned models on it. The results of classification task can be summarized with the help of confusion matrix as follows:

	predicted							
true	3	4	5	6	7	8	9	-err.-
3	0	0	0	6	0	0	0	6
4	0	0	22	20	1	0	0	43
5	0	0	304	121	2	1	0	124
6	0	1	108	425	32	1	0	142
7	0	0	6	116	93	1	0	123
8	0	0	0	27	12	0	0	39
9	0	0	0	0	1	0	0	1
-err.-	0	1	136	290	48	3	0	478

	predicted							
true	3	4	5	6	7	8	9	-err.-
3	0	0	0	6	0	0	0	6
4	0	0	0	43	0	0	0	43
5	0	0	0	428	0	0	0	428
6	0	0	0	567	0	0	0	0
7	0	0	0	216	0	0	0	216
8	0	0	0	39	0	0	0	39
9	0	0	0	1	0	0	0	1
-err.-	0	0	0	733	0	0	0	733

	predicted							
true	3	4	5	6	7	8	9	-err.-
3	0	0	0	6	0	0	0	6
4	0	3	22	18	0	0	0	40
5	0	2	322	102	2	0	0	106
6	0	0	83	458	25	1	0	109
7	0	0	5	91	118	2	0	98
8	0	0	0	16	11	12	0	27
9	0	0	0	0	1	0	0	1
-err.-	0	2	110	233	39	3	0	387

Table 17: Confusion matrix for classification task. Left is SVM with 478/1300 errors, middle is neural network with 733/1300 errors, and right is random forest with 387/1300 errors.

4.5 Benchmark Comparison

For a complete comparison of the models MSE and MMCE is not enough. It is also important to report the confidence intervals and a comparison with a naïve baseline. In our case, the naïve baseline model chosen is a trainer that only predicts the mean values. For our classification task this would be 6 and our regression task 5.819. The result of our benchmark table are given below:

Classification	
Model	**Model Error (MMCE)**
SVM Tuned	34.84 ± 2.59
SVM Un-tuned	43.61 ± 2.69
Neural Network Tuned	56.38 ± 2.69
Neural Network Un-tuned	67.07 ± 2.55
Random Forest Tuned	30.84 ± 2.48
Random Forest Un-tuned	31.23 ± 2.51
Naïve model	56.38 ± 2.69

Regression	
Model	**Model Error (MMSE)**
SVM Tuned	48.72 ± 2.72
SVM Un-tuned	49.45 ± 2.72
Neural Network Tuned	59.86 ± 2.66
Neural Network Un-tuned	76.3 ± 2.31
Random Forest Tuned	38.14 ± 2.07
Random Forest Un-tuned	38.46 ± 2.09
Naïve model	76.3 ± 2.3

Table 18: Result of benchmark table. Neural networks perform the worst in every iteration for both classification and regression task. Tuned random forest perform the best on test set even though they could not out perform the un-tuned version in the validation set. Random forest in the regression task seems to outperform all other classifiers by a very high margin. All models perform better than the naïve model other than the neural networks in the regression task.

4.6 Predictive Power of Wine Colour

At the beginning of the training step we added a new dummy variable signifying the colour of the wine. To determine its importance with respect to other chemicals in predicting quality a generalized linear model (GLM) with greedy backward stepwise feature selection algorithm was implemented. It assesses the best feature based on change in loess R-square values. In order to ensure robustness of result, repeated cross-validation was implemented with ten folds and repeated three times.

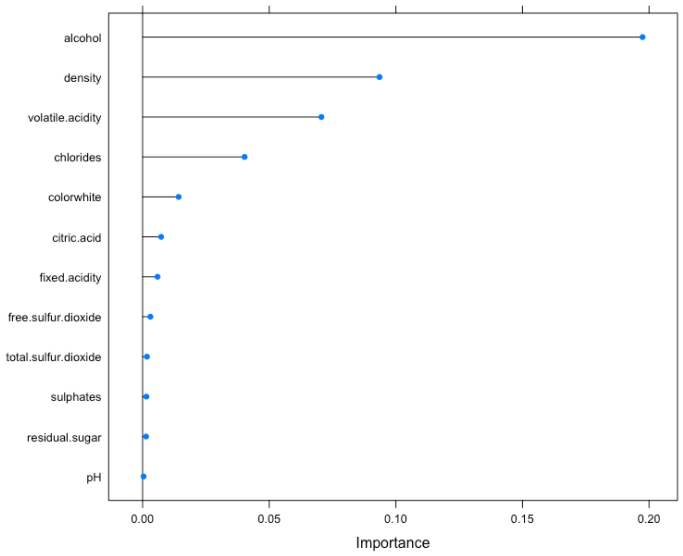


Figure 28: Showing feature importance with respect to predicting quality based on change in Loess R-Square value.

We see that wine colour offer some explanatory power ranking 5th amongst all variables and ahead of majority of chemical variables.

5 Further Analyses

With our results we can now re-approach the initial questions that the CTO had posed. Although super-human performance in wine tasting is out of the scope of this dataset, we can explore new modelling techniques to get better prediction results on quality which can hopefully be used as a basis for improving AI's wine tasting capabilities as better datasets become available. We can also apply variable transformations to see if it leads to a better understanding of components that make a wine good.

With this framework in mind further modifications have been made to our dataset and training model from the previous section. Residual sugar, chlorides and sulphates have been log transformed to impose a more normal behaviour and reduce the impact of outliers on our training algorithm.[2] A new XGBoost algorithm has been implemented for both tasks and tuned to see if the results improve. The results are as follows:

[2] See univariate exploratory data analysis for details

Classification		Regression	
Model	**Model Error (MMCE)**	**Model**	**Model Error (MMSE)**
SVM Tuned	34.15 ± 2.57	SVM Tuned	44.755 ± 2.70
SVM Un-tuned	41 ± 2.60	SVM Un-tuned	45.29 ± 2.70
Neural Network Tuned	56.38 ± 2.69	Neural Network Tuned	58.68 ± 2.6
Neural Network Untuned	56.38 ± 2.69	Neural Network Untuned	76.23 ± 2.3
Random Forest Tuned	28.76 ± 2.46	Random Forest Tuned	36.04 ±2.61
Random Forest Untuned	29.92 ± 2.48	Random Forest Untuned	36.38 ± 2.61
XGBoost Tuned	29.57 ± 2.48	XGBoost Tuned	37.42 ± 2.6
XGBoost Untuned	37.32 ± 2.62	XGBoost Untuned	-
Naïve Model	56.38 ± 2.69	Naïve Model	76.3 ± 2.3

Table 19: Benchmark result for new tuned classifiers after log transformation of our variables. We see an improvement in most of the error statistics compared to previous benchmark table (highlighted in yellow). XGboost is very effective in classification task and can show performance close to random forest levels for regression task. XGBoost un-tuned left blank because it produced NaNs.

To explore whether wine quality is purely subjective and based on random human bias, a hypothesis test has been setup with the help of a logistic regression model. If the scenario posed by the philosopher is correct then we should see high p-values on our independent variables and fail to reject the null hypothesis, which signifies the random relation. The result of the logistic model is shown in table 21.

We can conclude from the above table that at least for this dataset a discernable relationship exists between quality and chemical composition. To establish the relationship more comprehensively further sampling and experiments on chemical composition is necessary.

```
Call:
glm(formula = quality ~ fixed.acidity + volatile.acidity + citric.acid +
    residual.sugar + chlorides + free.sulfur.dioxide + total.sulfur.dioxide +
    density + pH + sulphates + alcohol + colorwhite, data = wine.set)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.7796  -0.4671  -0.0444   0.4561   3.0211

Coefficients:
                       Estimate Std. Error t value Pr(>|t|)
(Intercept)           1.048e+02  1.414e+01   7.411 1.42e-13 ***
fixed.acidity         8.507e-02  1.576e-02   5.396 7.05e-08 ***
volatile.acidity     -1.492e+00  8.135e-02 -18.345  < 2e-16 ***
citric.acid          -6.262e-02  7.972e-02  -0.786   0.4322
residual.sugar        6.244e-02  5.934e-03  10.522  < 2e-16 ***
chlorides            -7.573e-01  3.344e-01  -2.264   0.0236 *
free.sulfur.dioxide   4.937e-03  7.662e-04   6.443 1.25e-10 ***
total.sulfur.dioxide -1.403e-03  3.237e-04  -4.333 1.49e-05 ***
density              -1.039e+02  1.434e+01  -7.248 4.71e-13 ***
pH                    4.988e-01  9.058e-02   5.506 3.81e-08 ***
sulphates             7.217e-01  7.624e-02   9.466  < 2e-16 ***
alcohol               2.227e-01  1.807e-02  12.320  < 2e-16 ***
colorwhite           -3.613e-01  5.675e-02  -6.367 2.06e-10 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 0.5374377)

    Null deviance: 4953.7  on 6496  degrees of freedom
Residual deviance: 3484.7  on 6484  degrees of freedom
AIC: 14418

Number of Fisher Scoring iterations: 2
```

Table 20: Result of logistic regression. Most variables are significant at the 99% confidence interval. Interestingly, white wine has a strong negative coefficient indicating red wine is better preferred by sommeliers in terms of quality.

Concluding Remarks

In this report we established the scope of questions that can be answered with our wine dataset. We saw that red wine and white wine are governed by different distributions but this was no conclusive due to imbalanced dataset. The best classifier for predicting wine quality is xgboost for classification task and random forest for regression task. We also established a significant statistical relationship between the variables in the dataset and quality.

Bibliography

Cortez, e. (2009). Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems* , 547-553.

RBF SVM Parameters. (2018). Retrieved from scikit learn: http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html

Wikipedia. (n.d.). *Random Forest*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Random_forest

Appendix

Univariate Summary Tables

```
fixed.acidity    volatile.acidity  citric.acid
Min.   : 3.800   Min.   :0.0800   Min.   :0.0000
1st Qu.: 6.400   1st Qu.:0.2300   1st Qu.:0.2500
Median : 7.000   Median :0.2900   Median :0.3100
Mean   : 7.215   Mean   :0.3397   Mean   :0.3186
3rd Qu.: 7.700   3rd Qu.:0.4000   3rd Qu.:0.3900
Max.   :15.900   Max.   :1.5800   Max.   :1.6600
residual.sugar     chlorides       free.sulfur.dioxide
Min.   : 0.600   Min.   :0.00900   Min.   :  1.00
1st Qu.: 1.800   1st Qu.:0.03800   1st Qu.: 17.00
Median : 3.000   Median :0.04700   Median : 29.00
Mean   : 5.443   Mean   :0.05603   Mean   : 30.53
3rd Qu.: 8.100   3rd Qu.:0.06500   3rd Qu.: 41.00
Max.   :65.800   Max.   :0.61100   Max.   :289.00
total.sulfur.dioxide    density             pH
Min.   :  6.0        Min.   :0.9871   Min.   :2.720
1st Qu.: 77.0        1st Qu.:0.9923   1st Qu.:3.110
Median :118.0        Median :0.9949   Median :3.210
Mean   :115.7        Mean   :0.9947   Mean   :3.219
3rd Qu.:156.0        3rd Qu.:0.9970   3rd Qu.:3.320
Max.   :440.0        Max.   :1.0390   Max.   :4.010
  sulphates         alcohol         quality
Min.   :0.2200   Min.   : 8.00   Min.   :3.000
1st Qu.:0.4300   1st Qu.: 9.50   1st Qu.:5.000
Median :0.5100   Median :10.30   Median :6.000
Mean   :0.5313   Mean   :10.49   Mean   :5.818
3rd Qu.:0.6000   3rd Qu.:11.30   3rd Qu.:6.000
Max.   :2.0000   Max.   :14.90   Max.   :9.000
   color
Length:6497
Class :character
Mode  :character
```

```
fixed.acidity   volatile.acidity  citric.acid    residual.sugar
Min.   : 4.60   Min.   :0.1200   Min.   :0.000   Min.   : 0.900
1st Qu.: 7.10   1st Qu.:0.3900   1st Qu.:0.090   1st Qu.: 1.900
Median : 7.90   Median :0.5200   Median :0.260   Median : 2.200
Mean   : 8.32   Mean   :0.5278   Mean   :0.271   Mean   : 2.539
3rd Qu.: 9.20   3rd Qu.:0.6400   3rd Qu.:0.420   3rd Qu.: 2.600
Max.   :15.90   Max.   :1.5800   Max.   :1.000   Max.   :15.500
  chlorides       free.sulfur.dioxide total.sulfur.dioxide
Min.   :0.01200   Min.   : 1.00       Min.   :  6.00
1st Qu.:0.07000   1st Qu.: 7.00       1st Qu.: 22.00
Median :0.07900   Median :14.00       Median : 38.00
Mean   :0.08747   Mean   :15.87       Mean   : 46.47
3rd Qu.:0.09000   3rd Qu.:21.00       3rd Qu.: 62.00
Max.   :0.61100   Max.   :72.00       Max.   :289.00
   density             pH          sulphates         alcohol
Min.   :0.9901   Min.   :2.740   Min.   :0.3300   Min.   : 8.40
1st Qu.:0.9956   1st Qu.:3.210   1st Qu.:0.5500   1st Qu.: 9.50
Median :0.9968   Median :3.310   Median :0.6200   Median :10.20
Mean   :0.9967   Mean   :3.311   Mean   :0.6581   Mean   :10.42
3rd Qu.:0.9978   3rd Qu.:3.400   3rd Qu.:0.7300   3rd Qu.:11.10
Max.   :1.0037   Max.   :4.010   Max.   :2.0000   Max.   :14.90
   quality         color
Min.   :3.000   Length:1599
1st Qu.:5.000   Class :character
Median :6.000   Mode  :character
Mean   :5.636
3rd Qu.:6.000
Max.   :8.000
```

```
fixed.acidity    volatile.acidity  citric.acid
Min.   : 3.800   Min.   :0.0800   Min.   :0.0000
1st Qu.: 6.300   1st Qu.:0.2100   1st Qu.:0.2700
Median : 6.800   Median :0.2600   Median :0.3200
Mean   : 6.855   Mean   :0.2782   Mean   :0.3342
3rd Qu.: 7.300   3rd Qu.:0.3200   3rd Qu.:0.3900
Max.   :14.200   Max.   :1.1000   Max.   :1.6600
residual.sugar     chlorides       free.sulfur.dioxide
Min.   : 0.600   Min.   :0.00900   Min.   :  2.00
1st Qu.: 1.700   1st Qu.:0.03600   1st Qu.: 23.00
Median : 5.200   Median :0.04300   Median : 34.00
Mean   : 6.391   Mean   :0.04577   Mean   : 35.31
3rd Qu.: 9.900   3rd Qu.:0.05000   3rd Qu.: 46.00
Max.   :65.800   Max.   :0.34600   Max.   :289.00
total.sulfur.dioxide    density             pH
Min.   :  9.0        Min.   :0.9871   Min.   :2.720
1st Qu.:108.0        1st Qu.:0.9917   1st Qu.:3.090
Median :134.0        Median :0.9937   Median :3.180
Mean   :138.4        Mean   :0.9940   Mean   :3.188
3rd Qu.:167.0        3rd Qu.:0.9961   3rd Qu.:3.280
Max.   :440.0        Max.   :1.0390   Max.   :3.820
  sulphates         alcohol         quality
Min.   :0.2200   Min.   : 8.00   Min.   :3.000
1st Qu.:0.4100   1st Qu.: 9.50   1st Qu.:5.000
Median :0.4700   Median :10.40   Median :6.000
Mean   :0.4898   Mean   :10.51   Mean   :5.878
3rd Qu.:0.5500   3rd Qu.:11.40   3rd Qu.:6.000
Max.   :1.0800   Max.   :14.20   Max.   :9.000
   color
Length:4898
Class :character
Mode  :character
```

Table A1: Full univariate summaries. Table on the left shows the full dataset. Table in the middle shows red wine and table on the right shows white wine.

CODE APPENDIX

```
#=======================
#   TASK A2
#=======================


#Import Red Wine Dataset


winequality.red <- read.csv("~/Desktop/UCL Modules/Selected Topics in Statistics/ICA 2/Q1/Data/winequality/winequality-red.csv", sep=";")


#Rename dataset
red.wine <- winequality.red


#Add new color column
red.wine$color <- "red"


#-------------------
#Import White Wine Dataset


winequality.white <- read.csv("~/Desktop/UCL Modules/Selected Topics in Statistics/ICA 2/Q1/Data/winequality/winequality-white.csv", sep=";")


#Rename Dataset
white.wine <- winequality.white


#Add new color column
white.wine$color <- "white"
#----------------------


#Create Joint Dataset
```

```
wine.set <- rbind(red.wine, white.wine)

#=============================

#=============================

#   TASK A(3)

#=============================

## PERFORM EXPLORATORY DATA ANALYSIS

#This report will explore 12 chemical properties of a set of 6497 variants of the Portugese "vinho verde".1599 red wines, and 4898 white wines. The set of wines was evaluated by three assessors who privded the scores between 0(disgusting) and 10(excellent) for each wine.

#Libraries

library(ggplot2)

library(tidyr)

library(dplyr)

library(lubridate)

library(gridExtra)

library(reshape2)

library(GGally)

library(corrplot)

library(corrgram)

## SEE DATASET

str(wine.set)

## DATA SET DESCRIPTIVE STATISTICS (MEAN AND 5 NUMBER SUMMARY)

summary(white.wine)

summary(red.wine)

summary(wine.set)
```

```
#==========================================

## UNIVARIATE PLOT SECTION

#1) Quality
ggplot(data = wine.set, aes(x = quality)) + geom_bar() + facet_wrap(~color, ncol = 1)
by(wine.set$quality, wine.set$color, summary)

#2) Fixed acidity
ggplot(data = wine.set, aes(x = fixed.acidity)) + geom_bar() + facet_wrap(~color, ncol = 1)
by(wine.set$fixed.acidity, wine.set$color, summary)

#3) Volatile Acidity
ggplot(data = wine.set, aes(x = volatile.acidity)) + geom_bar() + facet_wrap(~color, ncol = 1)
by(wine.set$volatile.acidity, wine.set$color, summary)

#4) Citric Acid
ggplot(data = wine.set, aes(x = citric.acid)) + geom_bar() + facet_wrap(~color, ncol = 1)
by(wine.set$citric.acid, wine.set$color, summary)

#Log transformation didn't help.

#5) Residual Sugar
ggplot(data = wine.set,aes(x = residual.sugar)) + geom_histogram(binwidth = 0.2) +
facet_wrap(~color, ncol = 1)
by(wine.set$residual.sugar, wine.set$color, summary)

ggplot(data = wine.set, aes(x = residual.sugar)) + geom_histogram(binwidth = 0.2) +
coord_cartesian(xli = c(0, quantile(wine.set$residual.sugar, 0.99))) + facet_wrap(~color, ncol = 1)

ggplot(data = subset(wine.set, color == "white"), aes(x = residual.sugar)) + geom_histogram(binwidth
= 0.02) + scale_x_log10()
```

#6) Chlorides

```
ggplot(data = wine.set, aes(x = chlorides)) + geom_bar() + facet_wrap(~color, ncol = 1)

by(wine.set$chlorides, wine.set$color, summary)
```

```
ggplot(data = wine.set ,aes(x = chlorides)) + geom_histogram(binwidth = 0.025) + scale_x_log10() +
facet_wrap(~color, ncol = 1)
```

#7) Free Sulfur Dioxide

```
ggplot(data = wine.set, aes(x = free.sulfur.dioxide)) + geom_bar() + facet_wrap(~color, ncol = 1)

by(wine.set$free.sulfur.dioxide, wine.set$color, summary)
```

#8) Total Sulfur Dioxide

```
ggplot(data = wine.set, aes(x = total.sulfur.dioxide)) + geom_histogram(binwidth = 5) +
facet_wrap(~color, ncol = 1)

by(wine.set$total.sulfur.dioxide, wine.set$color, summary)
```

#Log transform doesn't help.

#9) Density

```
ggplot(data = wine.set, aes(x = density)) + geom_histogram(binwidth = 0.0002) + facet_wrap(~color,
ncol = 1)

by(wine.set$density, wine.set$color, summary)
```

```
ggplot(data = wine.set, aes(x = density)) + geom_histogram(binwidth = 0.0002) +
coord_cartesian(xlim = c(0.987, quantile(wine.set$density, 0.99))) + facet_wrap(~color, ncol = 1)


#10) pH


ggplot(data = wine.set, aes(x = pH)) + geom_histogram(binwidth = 0.01) + facet_wrap(~color, ncol =
1)

by(wine.set$pH, wine.set$color, summary)



#11) Sulphates

ggplot(data = wine.set,aes(x = sulphates)) + geom_histogram(binwidth = 0.2) + facet_wrap(~color,
ncol = 1)

by(wine.set$sulphates, wine.set$color, summary)


#Log transform sulphates

ggplot(data = subset(wine.set, color == "red"), aes(x = sulphates)) + geom_histogram(binwidth =
0.02) + scale_x_log10()


#12) Alcohol


ggplot(data = wine.set,aes(x = alcohol)) + geom_histogram(binwidth = 0.1) + facet_wrap(~color, ncol
= 1)

by(wine.set$alcohol, wine.set$color, summary)


#=======================================


## BIVARITE PLOT


#Red wine



wsr <- cor(subset(wine.set
```

```
            ,color == "red"
            ,select = -c(color))
        ,method = "pearson")

corrplot(wsr)
wsr
#cor(red.wine$quality, red.wine$volatile.acidity)
#cor(red.wine$quality, red.wine$alcohol)

#White wine

wsw <- cor(subset(wine.set
            ,color == "white"
            ,select = -c(color))
        ,method = "pearson")
corrplot(wsw)
wsw


#Another way to visualize - Corrgram (Intro to Regression - Kaggle) - Better way to visualize than above. Blue is positive correlation, red is negative correlation.
corrgram(red.wine, lower.panel = panel.shade, upper.panel = panel.ellipse)
corrgram(white.wine, lower.panel = panel.shade, upper.panel = panel.ellipse)


#-----Exploring our highly correlated variables further----------

#Create new variables that group quality into 3 categorical variables - namely Poor, Average, Great.
wine.set$quality.group <- cut(wine.set$quality, c(1,4,6,10), labels = c("Poor", "Average", "Great"))
```

```
#=====================================
#     MULTIVARIATE PLOT SECTION
#=====================================



#================================================
#Cluster, Collinearity or Non Linearity
#================================================
#Red Wine
#------------------------------
ggplot(data = subset(wine.set, color == "red" & quality.group != "Average")
     ,aes(x = alcohol, y = volatile.acidity)) +
  geom_point(aes(colour = quality.group)) +
  geom_smooth(aes(colour = quality.group)
              ,method = lm) +
  coord_cartesian(xlim = c(min(wine.set$alcohol), quantile(wine.set$alcohol, 0.99))
                  ,ylim = c(min(wine.set$volatile.acidity), quantile(wine.set$volatile.acidity, 0.99))) +
  labs(title="ALCOHOL CONTENT AND VOLATILE ACIDITY BY RED WINE QUALITY"
     ,x = "Alcohol (g/ml)"
     ,y = "Volatile Acidity (g/dm³)"
     ,color = "Quality")


#We observe clustering here. Poor quality alcohol are clustered more towards the top left with high volatility and low alcohol content whereas great quality wines are clustered near the bottom right with high alcohol content and low volatility.


#---------------------
#   White Wine
#--------------------


ggplot(data = subset(wine.set, color == "white" & quality.group != "Average")
     ,aes(x = alcohol, y = density)) +
```

```
 geom_point(aes(colour = quality.group)) +
 geom_smooth(aes(colour = quality.group)
          ,method = lm) +
 coord_cartesian(xlim = c(min(wine.set$alcohol), quantile(wine.set$alcohol, 0.99))
              ,ylim = c(min(wine.set$density), quantile(wine.set$density, 0.99))) +
 labs(title="ALCOHOL CONTENT AND DENSITY BY WHITE WINE QUALITY"
    ,x = "Alcohol (g/ml)"
    ,y = "Density (g/cm³)"
    ,color = "Quality")

#We see linearity. The ratio of good to bad quality wines seems to be increasing as density decreases
and alcohol content increaes.

#--------------------------

#T-Sne
library(tsne)

##Plotting
tsne_red <- tsne(red.wine[1:12], perplexity = 50, max_iter = 600)
plot(tsne_red)
text(tsne_red, labels = red.wine$quality, col = red.wine$quality)

tsne_white <- tsne(white.wine[1:12], perplexity = 50, max_iter = 30)
plot(tsne_white)
text(tsne_white, labels = white.wine$quality, col = white.wine$quality)

tsne_full <- tsne(wine.set[1:12], perplexity = 50, max_iter = 10)
plot(tsne_full)
text(tsne_full, labels = wine.set$quality, col = wine.set$quality)
```

```
#==================================
#       TASK A(4)
#==================================

#package installs
#install.packages("deepnet")
#install.packages("randomForest")
#install.packages("caret")
#install.packages("caTools")

#Library
library(caret)
library(mlr)
library(randomForest)
library(caTools)
library(Metrics)

#PREPROCESSING

#Create dummy variable for colors
color.dummy <- dummyVars("~color", data = wine.set)
dummy.frame <- data.frame(predict(color.dummy, newdata = wine.set))

#Create new dataset with dummy variables
wine.set <- cbind(wine.set, dummy.frame)
#Drop colors
wine.set <- subset(wine.set, select = -c(color))
#Drop colorred because of collinearity
wine.set <- subset(wine.set, select = -c(colorred))

#CREATE BALANCED TRAIN-TEST SPLIT BASED ON QUALITY
```

```
sample = sample.split(wine.set$quality, SplitRatio = 0.8)

wineset.train <- subset(wine.set, sample == TRUE)

wineset.test <- subset(wine.set, sample == FALSE)


#Check if balanced dataset

hist(wineset.train$quality)

table(wineset.train$quality)

hist(wineset.test$quality)

table(wineset.test$quality)

#---------------------------------


#CLASSIFICATION TASKs

wine.class = makeClassifTask(id = "quality",data = wineset.train, target = "quality")


#----------------------------------------

#MAKE LEARNERS

#----------------------------------------

#Constructing some untuned learners

learners <- makeLearners(c("classif.svm", "classif.nnTrain","classif.randomForest"), ids =
c("SVM.untuned", "NeuralNetwork.untuned", "RF.untuned"))


#SVM

lrn.SVM.untuned <- makeLearner("classif.svm")

#Get parameter set

getParamSet("classif.svm")


#Tuning grid SVM

tunegridparamSVM <- makeParamSet(

makeDiscreteParam("cost", values = 10^(-2:3)),

makeDiscreteParam("gamma", values = 2^(-5:5))

)
```

```
lrn.SVM.tuned <- makeTuneWrapper(lrn.SVM.untuned, cv3, mmce, tunegridparamSVM,
makeTuneControlGrid())

lrn.SVM.tuned$id <- "SVM.tuned"


#----------------------------------

#Neural Network

lrn.NN.untuned <- makeLearner("classif.nnTrain", hidden = c(3))

#Get parameter set

getParamSet("classif.nnTrain")


#Tuning grid NN

tunegridparamNN <- makeParamSet(

makeIntegerParam("hidden", lower = 3, upper = 10),

makeDiscreteParam("learningrate", values = c(0.5, 1.0, 1.5, 2.0)),

makeDiscreteParam("momentum", values = c(0.5, 1.0, 1.5, 2.0))

)

lrn.NN.tuned <- makeTuneWrapper(lrn.NN.untuned, cv3, mmce, tunegridparamNN,
makeTuneControlGrid())

#-------------------------------------------------

#Ensemble Random Forest

lrn.ERF.untuned <- makeLearner("classif.randomForest")

#Get parameter set

getParamSet("classif.randomForest")


#Tuning grid Forest

tunegridparamRF <- makeParamSet(

makeIntegerParam("ntree", lower = 10, upper = 100),

makeIntegerParam("mtry", lower = 1, upper = 12),

makeIntegerParam("nodesize", lower = 1, upper = 30)

)
```

```
lrn.ERF.tuned <- makeTuneWrapper(lrn.ERF.untuned, cv3, mmce, tunegridparamRF,
makeTuneControlGrid())

lrn.ERF.tuned$id <- "EnsembleForest.tuned"


#-----------------------------------------------


#Adding tuned classifiers to list of learners

learners <- c(learners, list(lrn.SVM.tuned, lrn.NN.tuned, lrn.ERF.tuned))


#-----------------------------------------------


#Specify measures to report

measures <-list(mmce)

#(mmce = mean misclassification error)


#-----------------------------------------------


#Run the Benchmarking Experiment

bmresults <- benchmark(learners, wine.class, cv3, measures)


#-----------------------------------------------


#ACCESSING BENCHMARK PERFORMANCE


#Get BMR Best Tune Results

getBMRTuneResults(bmresults)


#Performance of each iteration

getBMRPerformances(bmresults)


#Aggregate performance
```

```
getBMRAggrPerformances(bmresults)

#Return as data frame
getBMRPerformances(bmresults, as.df = TRUE)

#-------------------------------------------------

#PREDICTIONS
getBMRPredictions(bmresults)
#put as.df = TRUE to get a dataframe of above results
getBMRPredictions(bmresults, as.df = TRUE)

#-------------------------------------------------------

#VISUALIZE BENCHMARK EXPERIMENTS

#Box plot of mmce
plotBMRBoxplots(bmresults, measure = mmce, pretty.names = FALSE)

plotBMRBoxplots(bmresults, measure = mmce, style = "violin", pretty.names = FALSE) + aes(color =
learner.id) + theme(strip.text.x = element_text(size = 8))

#Visualizing aggregate performances
plotBMRSummary(bmresults, pretty.names = FALSE)

#Stacked bar charts
#plotBMRRanksAsBarChart(bmresults, pretty.names = FALSE)

#-------------------------------------------------------------
#RUN PREDICTION ON HELD-OUT TEST SET
```

```
#SVM Classification Tuned
lrnsvm.test = makeLearner("classif.svm", predict.type = "response", cost = 10, gamma = 1)
modsvm = train(lrnsvm.test, wine.class)
predsvm = predict(modsvm, task = wine.class)
svmtest.pred = predict(modsvm, newdata = wineset.test)
svmtest.pred
performance(svmtest.pred)
#Confidence Interval
1.96*sqrt(performance(svmtest.pred) * (1- performance(svmtest.pred))/1300)

#Neural Network Classification
lrnNN.test = makeLearner("classif.nnTrain", hidden = 5, learningrate = 1, momentum = 0.5)
modNN = train(lrnNN.test, wine.class)
predNN = predict(modNN, task = wine.class)
NNtest.pred = predict(modNN, newdata = wineset.test)
NNtest.pred
performance(NNtest.pred)
#Confidence Interval
1.96*sqrt(performance(NNtest.pred) * (1- performance(NNtest.pred))/1300)


#Random Forest Classification
lrnrf.test = makeLearner("classif.randomForest", ntree = 80, mtry = 2, nodesize = 1)
modrf = train(lrnrf.test, wine.class)
predrf = predict(modrf, task = wine.class, interval = "confidence")
testrf.pred = predict(modrf, newdata = wineset.test)
testrf.pred
performance(testrf.pred)
#Confidence Interval
1.96*sqrt(performance(testrf.pred) * (1- performance(testrf.pred))/1300)
```

```
#Naive Classifier - Guesses most frequent class 6 all the time
(1-567/1300)
#Confidence Interval
1.96*sqrt((0.5638462) * (1- 0.5638462)/1300)



#=================================================================
set.seed(123)

#=================================================================
#                   REGRESSION TASK
#=================================================================

#-----------------------------------------
#           MAKE LEARNERS
#-----------------------------------------
wine.regr = makeRegrTask(id = "quality", data = wineset.train, target = "quality")

#Define Cross Validation Strategy

cv <- makeResampleDesc("CV", iters = 3)

#Define performance measure
measures <-list(mse)

#-----------------------------------------
# Create tuned regrSVM
regrSVM <- makeLearner("regr.svm")
```

```
#Get parameter set

getParamSet("regr.svm")


#Tuning grid regrSVM

tunegridregrSVM <- makeParamSet(

makeDiscreteParam("cost", values = 10^(-2:3)),

makeDiscreteParam("gamma", values = 2^(-5:5))

)

learn.regrSVM.tuned <- makeTuneWrapper(regrSVM, cv3, tunegridregrSVM,
makeTuneControlGrid(), measures = measures)

lrn.regrSVM.tuned <- "regrSVM.tuned"


bmr.regr.untunedSVM <- benchmark(regrSVM, wine.regr, cv3, measures)

bmr.regr.tunedSVM <- benchmark(learn.regrSVM.tuned, wine.regr, cv3, measures)

#---------------------------------------------


# Create tuned regr.NeuralNetwork

regrNN <- makeLearner("regr.nnet")


#Get parameter set

getParamSet("regr.nnet")


#Tune grid regrNN

tunegridregrNN <- makeParamSet(

makeIntegerParam("size", lower = 3, upper = 10)

)


learn.regrNN.tuned <- makeTuneWrapper(regrNN, cv3, tunegridregrNN, measures = measures,
makeTuneControlGrid())


bmr.regr.untunedNN <- benchmark(regrNN, wine.regr, cv3, measures)

bmr.regr.tunedNN <- benchmark(learn.regrNN.tuned, wine.regr, cv3, measures)
```

```
#-----------------------------------------------------

#Create tuned regr.RF

regrRF <- makeLearner("regr.randomForest")

#Get parameter set

getParamSet("regr.randomForest")

#Tune grid regrRF

tunegridregrRF <- makeParamSet(

makeIntegerParam("ntree", lower = 10, upper = 100),

makeIntegerParam("mtry", lower = 1, upper = 12),

makeIntegerParam("nodesize", lower = 1, upper = 30)

)

learn.regrRF.tuned <- makeTuneWrapper(regrRF, cv3, tunegridregrRF, measures = measures,
makeTuneControlGrid())

bmr.regr.untunedRF <- benchmark(regrRF, wine.regr, cv3, measures)

bmr.regr.tunedRF <- benchmark(learn.regrRF.tuned, wine.regr, cv3, measures)

#-----------------------------------------------------------

#Merging Benchmark Results

#-----------------------------------------------------------

#Merging Benchmark Results
```

```
mergeBMR <- mergeBenchmarkResults(list(bmr.regr.untunedSVM, bmr.regr.tunedSVM,
bmr.regr.untunedNN, bmr.regr.tunedNN, bmr.regr.untunedRF, bmr.regr.tunedRF))

#============================================================

#ACCESSING BENCHMARK PERFORMANCE

#Get BMR Tune Results

getBMRTuneResults(mergeBMR)

#Performance of each iteration

getBMRPerformances(mergeBMR)

#Aggregate performance

getBMRAggrPerformances(mergeBMR)

#Return as data frame

getBMRPerformances(mergeBMR, as.df = TRUE)

#-----------------------------------------------

#PREDICTIONS

getBMRPredictions(mergeBMR)

#put as.df = TRUE to get a dataframe of above results

getBMRPredictions(mergeBMR, as.df = TRUE)

#-----------------------------------------------

#VISUALIZE BENCHMARK EXPERIMENTS

#Box plot of mmce

plotBMRBoxplots(mergeBMR, measure = mse, pretty.names = FALSE)
```

```
plotBMRBoxplots(mergeBMR, measure = mse, style = "violin", pretty.names = FALSE) + aes(color =
learner.id) + theme(strip.text.x = element_text(size = 8))


#Visualizing aggregate performances

plotBMRSummary(mergeBMR, pretty.names = FALSE)


#Stacked bar charts

#plotBMRRanksAsBarChart(mergeBMR, pretty.names = FALSE)


#------------------------------------------------------------


#RUN PREDICTION ON HELD-OUT TEST SET


#SVM Regression

lrnRsvm.test = makeLearner("regr.svm", predict.type = "response", cost = 1, gamma = 0.125)

modRsvm = train(lrnRsvm.test, wine.regr)

predsvm = predict(modRsvm, task = wine.regr)

Rsvmtest.pred = predict(modRsvm, newdata = wineset.test)

Rsvmtest.pred

performance(Rsvmtest.pred)

#Confidence Interval

1.96*sqrt(performance(Rsvmtest.pred) * (1- performance(Rsvmtest.pred))/1300)



#Neural Network Classification

lrnRnn.test = makeLearner("regr.nnet", size = 6)

modRnn = train(lrnRnn.test, wine.regr)

predRnn = predict(modRnn, task = wine.regr)

Rnntest.pred = predict(modRnn, newdata = wineset.test)

Rnntest.pred
```

```
performance(Rnntest.pred)
#Confidence Interval
1.96*sqrt(performance(Rnntest.pred) * (1- performance(Rnntest.pred))/1300)

#Random Forest Classification
lrnRrf.test = makeLearner("regr.randomForest", ntree = 100, mtry = 3, nodesize = 1)
modRrf = train(lrnRrf.test, wine.regr)
predRrf = predict(modRrf, task = wine.regr)
testRrf.pred = predict(modRrf, newdata = wineset.test)
testRrf.pred
performance(testRrf.pred)
#Confidence Interval
1.96*sqrt(performance(testRrf.pred) * performance(testRrf.pred)/1300)



#Naive Classifier - Guess Mean 5.819 all the time
msenaive = mse(5.819, wineset.test$quality)
print(msenaive)
#Confidence Interval
1.96*sqrt(msenaive * (1- msenaive)/1300)

#=========================================
#    FEATURE IMPORTANCE
#=========================================
library(caret)
#Most important variables with regards to quality
#prepare training scheme
control <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
#train model
model <- train(quality~., data=wine.set, method="glmStepAIC", trControl=control)
```

```
#estimate variable importance
importance <- varImp(model, scale = FALSE)

#sumarize importance
print(importance)

# plot importance
plot(importance)

#================================================
#           TASK A5 Hypothesis Testing
#================================================

#Fitting a glm model on our dataset and checking the confidence level to see if quality is a random function of the variables in the dataset or if there is significant explantory power in the variables.

modelglm = glm(quality ~ fixed.acidity + volatile.acidity + citric.acid + residual.sugar + chlorides + free.sulfur.dioxide + total.sulfur.dioxide + density + pH + sulphates + alcohol + colorwhite, data = wine.set)

#summarize the model to check significance level
summary(modelglm)



#=======================
#   TASK A5 New Variables, New Models
#=======================

set.seed(123)

#Import Red Wine Dataset
```

```
winequality.red <- read.csv("~/Desktop/UCL Modules/Selected Topics in Statistics/ICA 2/Q1/Data/winequality/winequality-red.csv", sep=";")

#Rename dataset
red.wine <- winequality.red

#Add new color column
red.wine$color <- "red"

#-------------------
#Import White Wine Dataset

winequality.white <- read.csv("~/Desktop/UCL Modules/Selected Topics in Statistics/ICA 2/Q1/Data/winequality/winequality-white.csv", sep=";")

#Rename Dataset
white.wine <- winequality.white

#Add new color column
white.wine$color <- "white"
#----------------------

#Create Joint Dataset

wine.set <- rbind(red.wine, white.wine)



#package installs
#install.packages("deepnet")
#install.packages("randomForest")
#install.packages("caret")
#install.packages("caTools")
```

```
#install.packages("xgboost")

#Library
library(caret)
library(mlr)
library(randomForest)
library(caTools)
library(xgboost)
library(Metrics)

#PREPROCESSING

#Create dummy variable for colors
color.dummy <- dummyVars("~color", data = wine.set)
dummy.frame <- data.frame(predict(color.dummy, newdata = wine.set))

#Create new dataset with dummy variables
wine.set <- cbind(wine.set, dummy.frame)
#Drop colors
wine.set <- subset(wine.set, select = -c(color))
#Drop colorred because of collinearity
wine.set <- subset(wine.set, select = -c(colorred))

#-----------------------------------------------
#CREATE NEW VARIABLES
wine.set$log.residual.sugar <- log(wine.set$residual.sugar)
wine.set$log.chlorides <- log(wine.set$chlorides)
wine.set$log.sulphates <- log(wine.set$sulphates)

#Drop old variables
wine.set <- subset(wine.set, select = -c(chlorides, sulphates, residual.sugar))
```

```
#CREATE BALANCED TRAIN-TEST SPLIT BASED ON QUALITY
sample = sample.split(wine.set$quality, SplitRatio = 0.8)
wineset.train <- subset(wine.set, sample == TRUE)
wineset.test <- subset(wine.set, sample == FALSE)

#Check if balanced dataset
hist(wine.set$quality)
table(wine.set$quality)
hist(wineset.train$quality)
table(wineset.train$quality)
hist(wineset.test$quality)
table(wineset.test$quality)
#---------------------------------

#CLASSIFICATION TASKs
wine.class = makeClassifTask(id = "quality",data = wineset.train, target = "quality")

#----------------------------------------
#MAKE LEARNERS
#----------------------------------------
#Constructing some untuned learners
learners <- makeLearners(c("classif.svm", "classif.nnTrain","classif.randomForest","classif.xgboost"),
ids = c("SVM.untuned", "NeuralNetwork.untuned", "RF.untuned", "xgboost.untuned"))

#SVM
lrn.SVM.untuned <- makeLearner("classif.svm")
#Get parameter set
getParamSet("classif.svm")

#Tuning grid SVM
```

```
tunegridparamSVM <- makeParamSet(
makeDiscreteParam("cost", values = 10^(-2:3)),
makeDiscreteParam("gamma", values = 2^(-5:5))
)
lrn.SVM.tuned <- makeTuneWrapper(lrn.SVM.untuned, cv3, mmce, tunegridparamSVM,
makeTuneControlGrid())
lrn.SVM.tuned$id <- "SVM.tuned"

#----------------------------------
#Neural Network
lrn.NN.untuned <- makeLearner("classif.nnTrain", hidden = c(3))
#Get parameter set
getParamSet("classif.nnTrain")

#Tuning grid NN
tunegridparamNN <- makeParamSet(
makeIntegerParam("hidden", lower = 3, upper = 10),
makeDiscreteParam("learningrate", values = c(0.5, 1.0, 1.5, 2.0)),
makeDiscreteParam("momentum", values = c(0.5, 1.0, 1.5, 2.0))
)
lrn.NN.tuned <- makeTuneWrapper(lrn.NN.untuned, cv3, mmce, tunegridparamNN,
makeTuneControlGrid())
#-------------------------------------------------
#Ensemble Random Forest
lrn.ERF.untuned <- makeLearner("classif.randomForest")
#Get parameter set
getParamSet("classif.randomForest")

#Tuning grid Forest
tunegridparamRF <- makeParamSet(
makeIntegerParam("ntree", lower = 10, upper = 100),
makeIntegerParam("mtry", lower = 1, upper = 12),
```

```
makeIntegerParam("nodesize", lower = 1, upper = 30)

)


lrn.ERF.tuned <- makeTuneWrapper(lrn.ERF.untuned, cv3, mmce, tunegridparamRF,
makeTuneControlGrid())

lrn.ERF.tuned$id <- "EnsembleForest.tuned"

#------------------------------------------------

#XGBOOST

lrn.xgb.untuned <- makeLearner("classif.xgboost")

#Get parameter set

getParamSet("classif.xgboost")


#Tuning Grid XGBOOST

tunegridparamxgb <- makeParamSet(

#The number of trees in the model (each one built sequentially)

makeDiscreteParam("nrounds", values = c(100,200,300,400,500)),

#number of splits in each tree

makeDiscreteParam("max_depth", values = c(1,5,10)),

# "shrinkage" - prevents overfitting

makeDiscreteParam("eta", values = c(0.1, 0.3, 0.5)),

#L2 regularization - prevents overfitting

makeNumericParam("lambda", lower = -1, upper = 0, trafo = function(x) 10^x)

)


lrn.xgb.tuned <- makeTuneWrapper(lrn.xgb.untuned, cv3, mmce, tunegridparamxgb,
makeTuneControlGrid())

lrn.xgb.tuned$id <- "xgb.tuned"

#------------------------------------------------


#Adding tuned classifiers to list of learners

learners <- c(learners, list(lrn.SVM.tuned, lrn.NN.tuned, lrn.ERF.tuned, lrn.xgb.tuned))
```

```
#-------------------------------------------------

#Specify measures to report
measures <-list(mmce)
#(mmce = mean misclassification error, did not take accuracy since accuracy is 1 - mmce.)

#-------------------------------------------------

#Run the Benchmarking Experiment
bmresults <- benchmark(learners, wine.class, cv3, measures)

#-------------------------------------------------

#ACCESSING BENCHMARK PERFORMANCE

#Get BMR Best Tune Results
getBMRTuneResults(bmresults)

#Performance of each iteration
getBMRPerformances(bmresults)

#Aggregate performance
getBMRAggrPerformances(bmresults)

#Return as data frame
getBMRPerformances(bmresults, as.df = TRUE)

#-------------------------------------------------

#PREDICTIONS
getBMRPredictions(bmresults)
```

```
#put as.df = TRUE to get a dataframe of above results

getBMRPredictions(bmresults, as.df = TRUE)


#-------------------------------------------------


#VISUALIZE BENCHMARK EXPERIMENTS


#Box plot of mmce

plotBMRBoxplots(bmresults, measure = mmce, pretty.names = FALSE)


#Violin plot of mmce

plotBMRBoxplots(bmresults, measure = mmce, style = "violin", pretty.names = FALSE) + aes(color =
learner.id) + theme(strip.text.x = element_text(size = 8)) + geom_violin(draw_quantiles = c(0.025,
0.975))


#Visualizing aggregate performances

plotBMRSummary(bmresults, pretty.names = FALSE)


#Stacked bar charts

#plotBMRRanksAsBarChart(bmresults, pretty.names = FALSE)


#-------------------------------------------------------------


#RUN PREDICTION ON HELD-OUT TEST SET


#SVM Classification

lrnsvm.test = makeLearner("classif.svm", predict.type = "response", cost = 1, gamma = 1)

modsvm = train(lrnsvm.test, wine.class)

predsvm = predict(modsvm, task = wine.class)

svmtest.pred = predict(modsvm, newdata = wineset.test)

svmtest.pred

performance(svmtest.pred)

calculateConfusionMatrix(svmtest.pred)
```

```
#Confidence interval
1.96*sqrt(performance(svmtest.pred) * (1 - performance(svmtest.pred))/1300)

#Neural Network Classification
lrnNN.test = makeLearner("classif.nnTrain", hidden = 5, learningrate = 1.0, momentum = 0.5)
modNN = train(lrnNN.test, wine.class)
predNN = predict(modNN, task = wine.class)
NNtest.pred = predict(modNN, newdata = wineset.test)
NNtest.pred
performance(NNtest.pred)
calculateConfusionMatrix(NNtest.pred)
1.96*sqrt(performance(NNtest.pred) * (1 - performance(NNtest.pred))/1300)

#Random Forest Classification
lrnrf.test = makeLearner("classif.randomForest", ntree = 100, mtry = 5, nodesize = 4)
modrf = train(lrnrf.test, wine.class)
predrf = predict(modrf, task = wine.class)
testrf.pred = predict(modrf, newdata = wineset.test)
testrf.pred
performance(testrf.pred)
calculateConfusionMatrix(testrf.pred)
1.96*sqrt(performance(testrf.pred) * (1 - performance(testrf.pred))/1300)

#XGB Classification
lrnxgb.test = makeLearner("classif.xgboost", nrounds = 200, max_depth = 3, eta = 0.1, lambda = 0.1)
modxgb = train(lrnxgb.test, wine.class)
predrf = predict(modxgb, task = wine.class)
testxgb.pred = predict(modxgb, task = wine.class)
testxgb.pred
performance(testxgb.pred)
calculateConfusionMatrix(testxgb.pred)
```

```
1.96*sqrt(performance(testxgb.pred) * (1 - performance(testxgb.pred))/1300)

#=================================================================

set.seed(123)

#=================================================================
#                   REGRESSION TASK
#=================================================================

#-----------------------------------------
#           MAKE LEARNERS
#-----------------------------------------
wine.regr = makeRegrTask(id = "quality", data = wineset.train, target = "quality")

#Define Cross Validation Strategy

cv <- makeResampleDesc("CV", iters = 3)

#Define performance measure
measures <-list(mse)

#-----------------------------------------
# Create tuned regrSVM
regrSVM <- makeLearner("regr.svm")

#Get parameter set
getParamSet("regr.svm")

#Tuning grid regrSVM
tunegridregrSVM <- makeParamSet(
makeDiscreteParam("cost", values = 10^(-2:3)),
```

```
makeDiscreteParam("gamma", values = 2^(-5:5))

)

learn.regrSVM.tuned <- makeTuneWrapper(regrSVM, cv3, tunegridregrSVM,
makeTuneControlGrid(), measures = measures)

lrn.regrSVM.tuned <- "regrSVM.tuned"


bmr.regr.untunedSVM <- benchmark(regrSVM, wine.regr, cv3, measures)

bmr.regr.tunedSVM <- benchmark(learn.regrSVM.tuned, wine.regr, cv3, measures)

#-----------------------------------------------


# Create tuned regr.NeuralNetwork

regrNN <- makeLearner("regr.nnet")


#Get parameter set

getParamSet("regr.nnet")


#Tune grid regrNN

tunegridregrNN <- makeParamSet(

makeIntegerParam("size", lower = 3, upper = 10)

)


learn.regrNN.tuned <- makeTuneWrapper(regrNN, cv3, tunegridregrNN, measures = measures,
makeTuneControlGrid())


bmr.regr.untunedNN <- benchmark(regrNN, wine.regr, cv3, measures)

bmr.regr.tunedNN <- benchmark(learn.regrNN.tuned, wine.regr, cv3, measures)


#-------------------------------------------------------


#Create tuned regr.RF

regrRF <- makeLearner("regr.randomForest")
```

```
#Get parameter set
getParamSet("regr.randomForest")

#Tune grid regrRF

tunegridregrRF <- makeParamSet(
makeIntegerParam("ntree", lower = 10, upper = 100),
makeIntegerParam("mtry", lower = 1, upper = 12),
makeIntegerParam("nodesize", lower = 1, upper = 30)
)

learn.regrRF.tuned <- makeTuneWrapper(regrRF, cv3, tunegridregrRF, measures = measures,
makeTuneControlGrid())

bmr.regr.untunedRF <- benchmark(regrRF, wine.regr, cv3, measures)
bmr.regr.tunedRF <- benchmark(learn.regrRF.tuned, wine.regr, cv3, measures)
#------------------------------------------------------------
#Create tuned regr.xgb
regrxgb <- makeLearner("regr.xgboost")
#Get Parameter Set
getParamSet("regr.xgboost")

#Tune grid regr.xgb

tunegridregrxgb <- makeParamSet(
#The number of trees in the model (each one built sequentially)
makeDiscreteParam("nrounds", values = c(100,200,300,400,500)),
#number of splits in each tree
makeDiscreteParam("max_depth", values = c(1,5,10)),
# "shrinkage" - prevents overfitting
makeDiscreteParam("eta", values = c(0.1, 0.3, 0.5)),
```

```
#L2 regularization - prevents overfitting

makeNumericParam("lambda", lower = -1, upper = 0, trafo = function(x) 10^x)

)


learn.regrxgb.tuned <- makeTuneWrapper(regrxgb, cv3, tunegridregrxgb, measures = measures,
makeTuneControlGrid())


bmr.regr.untunedxgb <- benchmark(regrxgb, wine.regr, cv3, measures)

bmr.regr.tunedxgb <- benchmark(learn.regrxgb.tuned, wine.regr, cv3, measures)


#------------------------------------------------------------


#Merging Benchmark Results


#------------------------------------------------------------


#Merging Benchmark Results


mergeBMR <- mergeBenchmarkResults(list(bmr.regr.untunedSVM, bmr.regr.tunedSVM,
bmr.regr.untunedNN, bmr.regr.tunedNN, bmr.regr.untunedRF, bmr.regr.tunedRF,
bmr.regr.untunedxgb, bmr.regr.tunedxgb))


#============================================================

#ACCESSING BENCHMARK PERFORMANCE


#Get BMR Tune Results

getBMRTuneResults(mergeBMR)


#Performance of each iteration

getBMRPerformances(mergeBMR)


#Aggregate performance
```

```
getBMRAggrPerformances(mergeBMR)

#Return as data frame
getBMRPerformances(mergeBMR, as.df = TRUE)

#-------------------------------------------------

#PREDICTIONS
getBMRPredictions(mergeBMR)
#put as.df = TRUE to get a dataframe of above results
getBMRPredictions(mergeBMR, as.df = TRUE)

#-------------------------------------------------

#VISUALIZE BENCHMARK EXPERIMENTS

#Box plot of mse
plotBMRBoxplots(mergeBMR, measure = mse, pretty.names = FALSE)

#Violin plot of mse
plotBMRBoxplots(mergeBMR, measure = mse, style = "violin", pretty.names = FALSE) + aes(color =
learner.id) + theme(strip.text.x = element_text(size = 8)) + geom_violin(draw_quantiles = c(0.025,
0.25, 0.75, 0.0975))

#Visualizing aggregate performances
plotBMRSummary(mergeBMR, pretty.names = FALSE)

#Stacked bar charts
#plotBMRRanksAsBarChart(mergeBMR, pretty.names = FALSE)

#-------------------------------------------------------------
```

```
#RUN PREDICTION ON HELD-OUT TEST SET

#SVM Regression
lrnRsvm.test = makeLearner("regr.svm", predict.type = "response", cost = 1, gamma = 0.0625)
modRsvm = train(lrnRsvm.test, wine.regr)
predsvm = predict(modRsvm, task = wine.regr)
Rsvmtest.pred = predict(modRsvm, newdata = wineset.test)
Rsvmtest.pred
performance(Rsvmtest.pred)
1.96*sqrt(performance(Rsvmtest.pred) * (1 - performance(Rsvmtest.pred))/1300)

#Neural Network Regression
lrnRnn.test = makeLearner("regr.nnet", size = 3)
modRnn = train(lrnRnn.test, wine.regr)
predRnn = predict(modRnn, task = wine.regr)
Rnntest.pred = predict(modRnn, newdata = wineset.test)
Rnntest.pred
performance(Rnntest.pred)
1.96*sqrt(performance(Rnntest.pred) * (1 - performance(Rnntest.pred))/1300)

#Random Forest Regression
lrnRrf.test = makeLearner("regr.randomForest", ntree = 90, mtry = 3, nodesize = 1)
modRrf = train(lrnRrf.test, wine.regr)
predRrf = predict(modRrf, task = wine.regr)
testRrf.pred = predict(modRrf, newdata = wineset.test)
testRrf.pred
performance(testRrf.pred)
1.96*sqrt(performance(testRrf.pred) * (1 - performance(testRrf.pred))/1300)

#XGBoost
```

```
lrnRxgb.test = makeLearner("regr.xgboost", nrounds = 100, max_depth = 10, eta = 0.1, lambda =
0.774)

modRxgb = train(lrnRxgb.test, wine.regr)

predRxgb = predict(modRxgb, task = wine.regr)

testRxgb.pred = predict(modRxgb, newdata = wineset.test)

testRxgb.pred

performance(testRxgb.pred)

1.96*sqrt(performance(testRxgb.pred) * (1 - performance(testRxgb.pred))/1300)


#=========================================

#    FEATURE IMPORTANCE

#=========================================


#Feature selection for classification task using randomForest.importance

fv1 = generateFilterValuesData(wine.class, method = "randomForest.importance")

fv1[["data"]]

plotFilterValues(fv1, n.show = 20)


#Feature selection for regression task using randomForest.importance

fv2 = generateFilterValuesData(wine.regr, method = "randomForest.importance")

fv2[["data"]]

plotFilterValues(fv2, n.show = 20)


#-----------------------------------------------

#install.packages("FSelector")

library(FSelector)

#information gain - classification task

fv3 = generateFilterValuesData(wine.class, method = c("information.gain"))

fv3[["data"]]

plotFilterValues(fv3, n.show = 20)
```

```
#information gain - regression task
fv4 = generateFilterValuesData(wine.regr, method = c("information.gain"))
fv4[["data"]]
plotFilterValues(fv4, n.show = 20)

#(GOT ERROR INITIALLY. Used following command to correct it in terminal:
#sudo ln -f -s $(/usr/libexec/java_home)/jre/lib/server/libjvm.dylib /usr/local/lib  )

#-----------------------------------------------
#gain.ratio - classification task
fv5 =  generateFilterValuesData(wine.class, method = c("gain.ratio"))
fv5[["data"]]
plotFilterValues(fv5, n.show = 20)

#gain.ratio - regression task
fv6 =  generateFilterValuesData(wine.regr, method = c("gain.ratio"))
fv6[["data"]]
plotFilterValues(fv6, n.show = 20)
```