CSC1001 Team Project Report

胥博凯 Xu Bokai	阮之泰 Ruan Zhitai	彭鼎翔 Peng Dingxiang
119010355 (33.3%)	119010254 (33.3%)	119010243 (33.3%)

Work Load:

Xu Bokai:

1) Grow the regression Tree
2) Implement M5' algorithm
3) Tree-graphing class
4) Tree object compiler
5) Write 1/3 report

Ruan Zhitai:

1) Write Data Processing Module
2) Process all the data
3) Grow a classification tree without pruning
4) Write 1/3 report

Peng Dingxiang:

1) Prune the regression Tree, write tree-pruning class
2) Construct tree class
3) Write 1/3 report
4) Write predictor class: input a tree object, return a prediction of score

Question we are interested to answer:

We want to **predict the score** of wine using 11 given attributes. To realize such a goal, we adopt **Regression Tree** (CART) and use **M5' Algorithm** to further improve the performance of our model.

(As a supplement, a member of our team Ruan Zhitai developed a classification tree to make comparison with our regression model.)

Data used:

We use training data-set from UCI which has 1119 data (80% of which were used to grow a tree and the remaining 20% were used to validate our model.) and use 480 testing data to test our model.

Approach we tried:

1) Our group member Ruan Zhitai suggest that we can use a formula which contains 11 independent variables to predict the socre, such as:
$$\boldsymbol{Y} = \beta_1 + \beta_1 \boldsymbol{X_1} + \beta_1 \boldsymbol{X_1} + \cdots + \beta_{12} \boldsymbol{X_{12}} + \boldsymbol{u}$$
This approach is straight-forward, and we will consider using it later.
2) Our group member Ruan Zhitai questioned the CART approach, he thought it too naïve and too simple. But just after we worked out the problem, he realized that CART algorithm is reliable.
3) Finally, we combine Linear Regression and CART into together. That's called M5' Model Tree, which further improves the performance of our model.

We didn't come to a conclusion before we worked out the problem, so we just try and try.

A Decision Tree is a hierarchically organized structure, with each node splitting data space into pieces based on value of a feature.

The goal of our mission is to construct a function

$$f : \mathfrak{X} \to \mathfrak{D}$$

such that the error

$$\sum_i \left| f\left(x^{(i)}\right) - y^{(i)} \right|^2$$

is minimized.

We divide our work into three parts.
We first prepare all the facilities for our program, including:

A data-importing python program:
This program will import data from csv files into python. It returns a dictionary which contains our desired data:

$$\mathbb{D} = \left\{X^{(i)}, Y^{(i)}\right\}_{i=1}^{N}$$

That is a python dictionary object with Nelements. In this case, we have N bottles of red wine. Here $X^{(i)}$refers to a 11-dimensional vector, because every single bottle of red wine contains 11 pointer which can tell us the quality of corresponding wine. Here is an example:

$$X^{(1)} = [fixed\ acidity = 6.2, volatile\ acidity = 0.56, citric\ acid = 0.09, chlorides = 0.053, free\ sulfer\ dioxide = 24, total\ sulfer\ dioxide = 32, density = 0.99402, pH = 3.54, sulphates = 0.6, alcohol = 11.3]$$

That is a 11-dimensional vector we desire. Every $X^{(i)}$ corresponds with a quality score $Y^{(i)}$, which is our target function. Here we know $Y^{(1)} = 5.0$. We construct a python dictionary to store those data.

We also use shelve library to save out model as .db file.

a tree-printing python program:
Given a tree object, the program can graph the tree using turtle module. This graph program can visualize our decision tree, making our work clearer to understand.
The printer can draw every node in our tree object on the screen. At the center of the node, you can see the line split into 2 sub-line. It means the wine is divided into two different parts $R_1$and R_2, where R_2 means those wine meet the condition. R_2 means those wine don't meet the condition. Moreover, the condition for each node will be displayed above the split point.
If the node does not have any child nodes, we call it a leaf. Behind the leaf, we can see a score, which is exactly the score of the red wine.

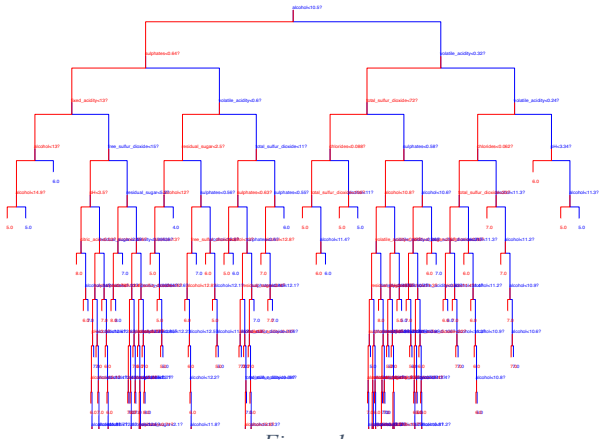


Figure 1

A classifier generating python program:
Given a tree object, the program will turn it into a python program according to logic. For example, given a node $\langle condition =' \ alcohol < 10.55' | left\ child\ node | right\ child\ node \rangle$, the program will write:

if alcohol < 10.55:
    [contents of left child node]
else:
    [contents of right child node]
By repeating writing logic structure above, we will get a classifier.

The program will output a classifier. We can post a vector $X^{(i)}$ to the classifier, it will return the corresponding quality score $Y^{(i)}$. Therefore, it actually implements our model.
By using classifier generating function, we can get a new classifier once we get a new tree object. It is useful when we need to prune the tree or evaluating our model.

The second part of our program is to generate a tree object using our data.
We can generate two kinds of decision tree, classification tree and regression tree. We will construct according to CART algorithm, which means Classification and Regression Tree.

Explain the working principle and logic of the approach used:
The construction of regression tree:

The mathematical expression of CART algorithm is very straightforward:

Start with a single region R_0 , and repeat the following procedure:
Firstly, pick a predictor (or pointer), and use greedy algorithm to find the best split point.
Given a pointer called alcohol. We examine the predictor alcohol.
We find that there are hundreds of different values of alcohol. So, we first sort the value in an ascending order. For example, if the original order is:

$$[wine\ 1\colon 11.3, wine\ 2\colon 9.2, wine\ 3\colon 11.7, wine\ 4\colon 11.8, wine\ 5\colon 9.2]$$

We can find some repetitive values in the list. Actually, we only need to keep one of those repetitive values. Then we can sort the list, yields

$$candidate\ list = [9.2, 11.3, 11.7, 11.8]$$

Now we assume that every value is a potential split point. So, first we assume that the first value 9.2 is our desired split value. Using that criteria, we divide those wines into two parts R_a and R_b, where R_a contains all red wine whose alcohol are equal or smaller than 9.2, R_a contains all red wine whose alcohol are greater than 9.2, that is,

$$R_a = [wine\ 2, wine\ 5]$$
$$R_b = [wine\ 1, wine\ 3, wine\ 4\ ]$$

According to CART algorithm, here we are to calculate RSS change of this split.
RSS refers to as Residual Sum of Squares, it is well defined:

$$RSS(Node) = \sum_{x_i \in R_a} (y_i - \bar{y}_{R_a})^2 + \sum_{x_i \in R_b} (y_i - \bar{y}_{R_b})^2$$

On the other hand, if we do not split those wine, in other words, those wine will be allocated to a leaf. The RSS of the leaf will be:

$$RSS(Leaf) = \sum_{x_i \in R_0} (y_i - \bar{y}_{R_0})^2$$

Therefore, we can calculate the decrease of RSS for the split process:

$$\boldsymbol{\Delta RSS = RSS(Leaf) - RSS(Node)}$$

We know that $RSS(Leaf)$ is a constant for a given leaf, so we only need to consider $RSS(Node)$.
A smaller $RSS(Node)$ means a larger decrease of RSS.

We don't need to consider other nodes $\{T \setminus this\ Node\}$ because they can be viewed as constant as well.

For the next step, we need to select an optimal split predictor(pointer) and an optimal split point.
Then we use greedy algorithm, iterating over every candidate to find an optimal value.
Then we only have to solve the problem:

$$\min_{p,s} \sum_{x_i \in R_a} (y_i - \bar{y}_{R_a})^2 + \sum_{x_i \in R_b} (y_i - \bar{y}_{R_b})^2$$

where p refers to our desired optimal predictor, s refers to our desired optimal split point.

We can use a 'for' loop to find the optimal s given p.
Next, we can find optimal p by comparing many local optimums. Here we provide a graph showing RSS for the root node:$RSS(Root\ Node)$.

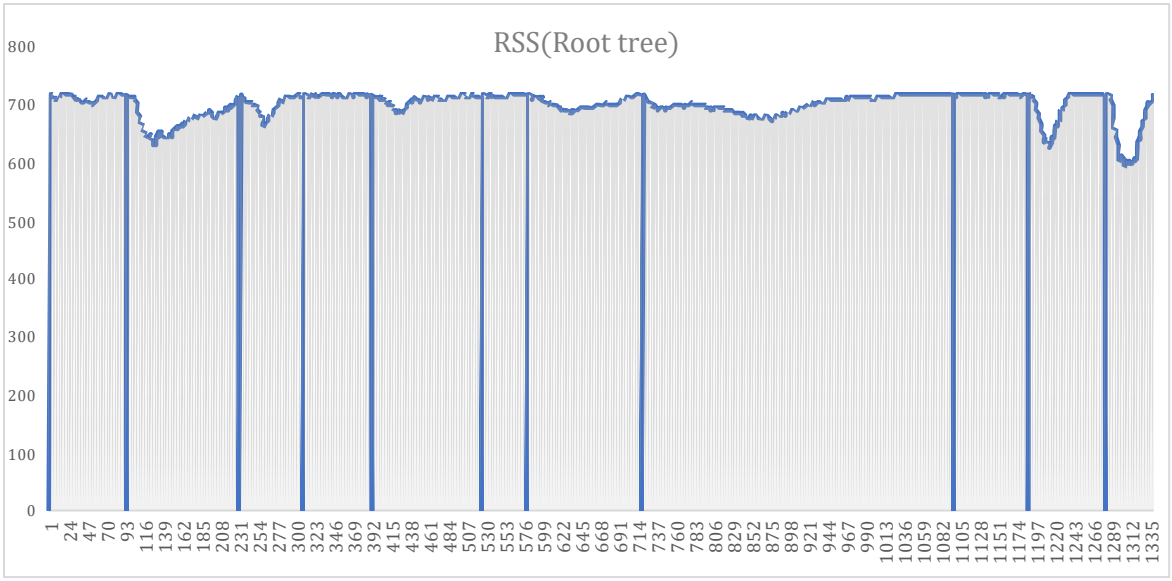


Figure 2

The first column is for predictor 'fixed acidity', we can find that it looks stable when we are iterating over every value. The second column is for predictor 'volatile acidity', we can find it decreases sharply when we are iterating over every value. We can find that the smallest $RSS(Root\ Node)$ occurs when we choose the predictor 'alcohol': the tenth column. So, we choose the tenth predictor.

After we split the sample space into two parts, we repeat the process until every sample space is "pure", that is, the last predictor of all wine in this deducted sample space are equal.
So, we get a fully grown tree:

Note that now we use 60% of our training dataset.

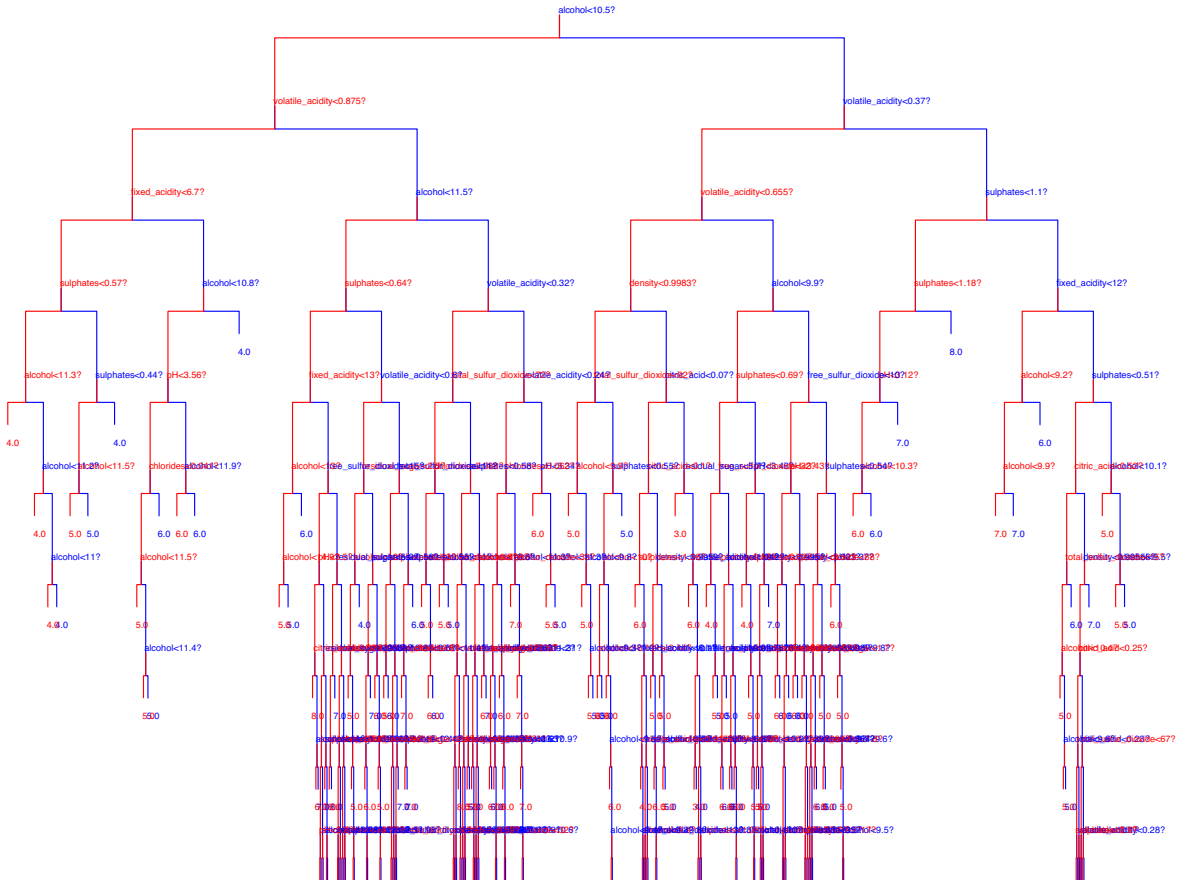


Figure 2

Use our compiler, we can generate a classifier python program to implement our algorithm:

```
def classifier(data_list):
    if alcohol<10.5:
        if volatile_acidity<0.37:
            if sulphates<1.1:
                if fixed_acidity<12:
                    if sulphates<0.51:
                        if alcohol<10.1:
                            if alcohol<9.5:
                                return 5.0
                            else:
                                return 5.0
                        else:
                            return 5.0
                    else:
                        if citric_acid<0.52:
                            if density<0.99556:
                                return 7.0
                            else:
                                if citric_acid<0.25:
                                    if citric_acid<0.23:
                                        if alcohol<10.1:
```

```
                                                return 6.0
                                            else:
                                                return 6.0
                                        else:
                                            if alcohol<10.1:
                                                return 5.0
                                            else:
                                                return 5.0
```

If we don't prune the tree, we can find the result is not as good as expected:
MSE (Mean Square Error) is: 0.6708333333333333.
This is relatively large error.
If we consider using MAE (Mean Absolute Error), which is more straightforward, we will find the result is: 0.5250.
which means that the average error of predicted score is 0.5.
We find a principle which can make us understand more about the algorithm. Here we have 1119 bottles of wine in our training dataset. Now we can try using 10%, 20%, 30% … 90%, 100% of our data from training dataset.

Table 1

	100%	90%	80%	70%	60%	50%	40%	30%	20%	~~10%~~
MSE	0.5792	0.6917	0.6771	0.6896	0.6708	0.7813	0.7229	0.7875	1.0063	~~0.858~~
MAE	0.4375	0.4958	0.4979	0.5063	0.5250	0.5646	0.5438	0.6083	0.7063	~~0.6~~

Note that the tree graph seems to be simpler if we only use 10% of data from our training dataset:

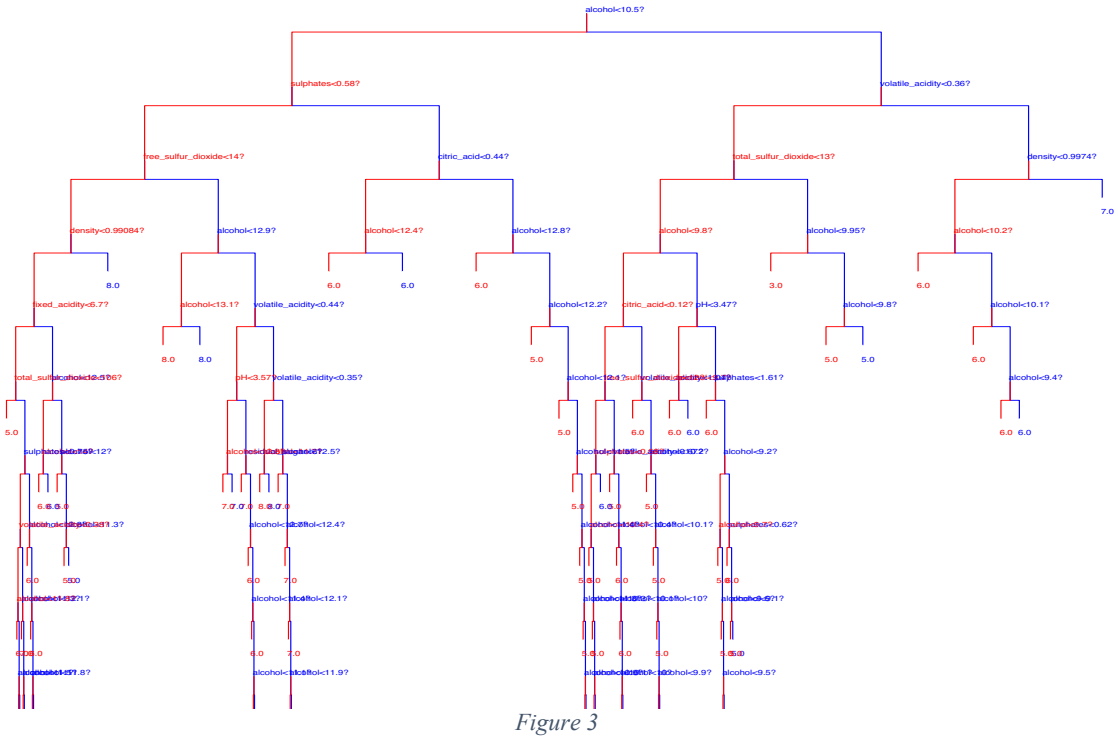


Figure 3

However, it seems not bad as expected. And more surprisingly we find that its accuracy is even higher than the case of 20%. It may be a coincidence. But if we consider the overall trend, we can clearly see that the MSE and MAE is increasing as we use less and less data.

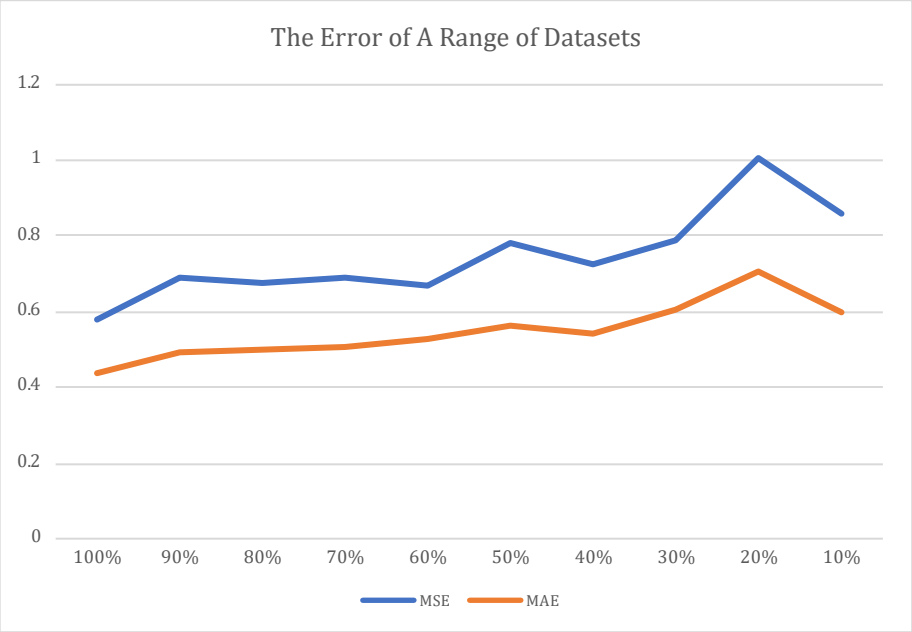


Figure 4

By using more data, we can find that actually we fail to significantly improve the quality of our classifier. So, we can try using a small number of our data, such as 60% or 30%. Then we use a new algorithm called pruning to optimize our regression tree.

Effort on the classification program:
Simultaneously, we are also working on the classification program, trying to obtain a better accuracy in the result.

Based on the basic classification tree program, we implemented two different types of computing methods.

1. This is a very basic computing method. Divide all the wines into two groups. All the scores of the wines in the first group are higher than 6, and for the second group, they are lower than 6 (The classification standard in the given question). Calculate the average score for all the parameters of the wine in the two groups respectively. In that case, we can obtain the average line of the parameters for all the wines in the superior wine group and the inferior wine group. After that, we can try to classify the new wines. If a parameter of the new wine is closer to the average line of the superior wine group, we give it one point, otherwise, it gets zero. After all the parameters are fully considered, we will get the result out of 11 points. If the result is larger or equal to 6 points, it goes to superior wine group, otherwise, it goes to inferior wine group.

   With this computing method, the success rate is around 70%.

2. The second computing method is called Gini impurity computing.

$$Gini(p) = \sum_{k=1}^{K} p_k(1-p_k) = 1 - \sum_{k=1}^{K} p_k^2$$

$$Gini(D, A) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2)$$

The basic computing method is given above. For every parameter of the wine, use one number in the training group as a dividing line. For all the wines above this line, part of them has a score lower than 6, while the other part of them has a superior score. For all the wine below this line, do the same thing. These two percentages are the impurity, considered as 'Pk' inside Gini(p) formula. After we compute the Gini(p) of both of these two groups, use the Gini (D, A) function to find out the final answer. For the lowest Gini (D, A), this number used as the division line is the threshold value in the node of the tree.

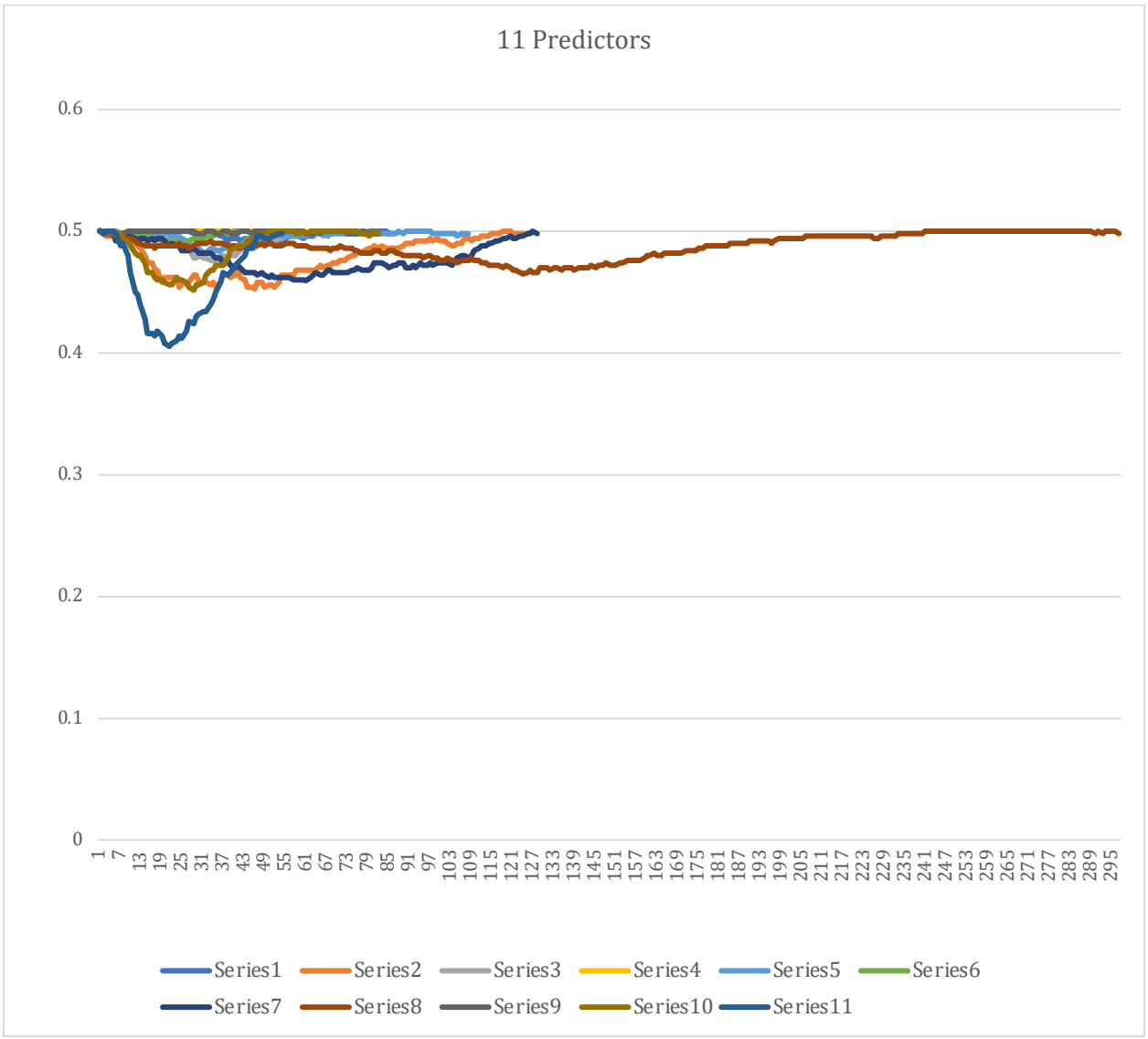


Figure 5

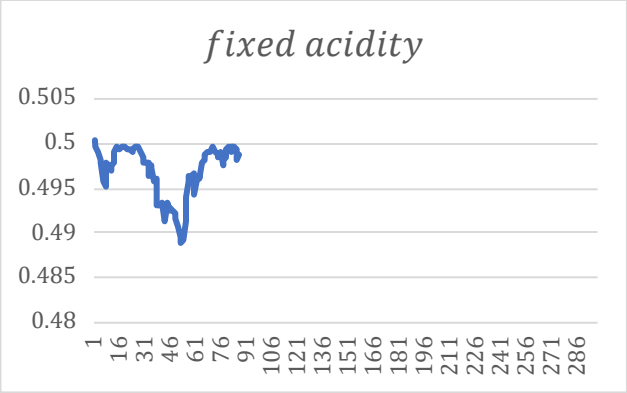


Figure 6

Maximum is roughly 0.5, and minimum is roughly 0.488, which is not good for a split point.

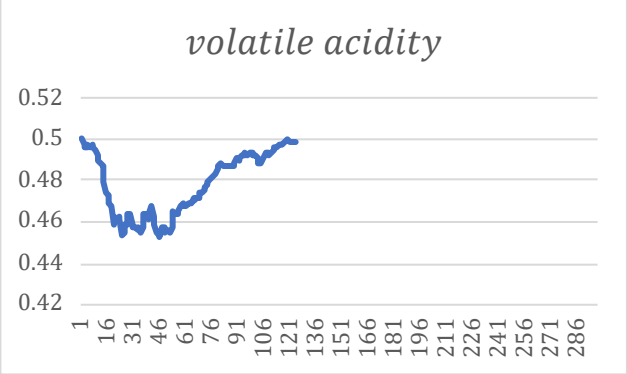


Figure 7

Maximum is roughly 0.5, and minimum is roughly 0.452, which is better than the first predictor

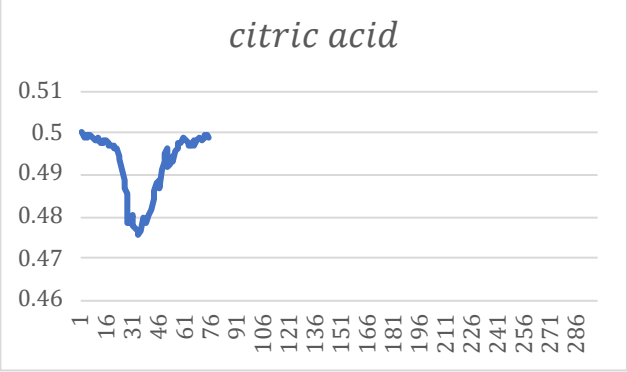


Figure 8

Maximum is roughly 0.5, and minimum is roughly 0.475, which is also not satisfying.

Figure 9

The maximized Gini index is 0.5. It is surprising that the minimum is only 0.4, which is the lowest among all candidates of split points. Therefore, we choose [alcohol No.20] split point. Then we look up the value of No.20 candidate. We find that the split point is alcohol == 10.4.

For these 11 parameters, we still choose alcohol as the first division standard, since it has the steepest slope.

After implementing the whole tree program, we have a new tree with different nodes.

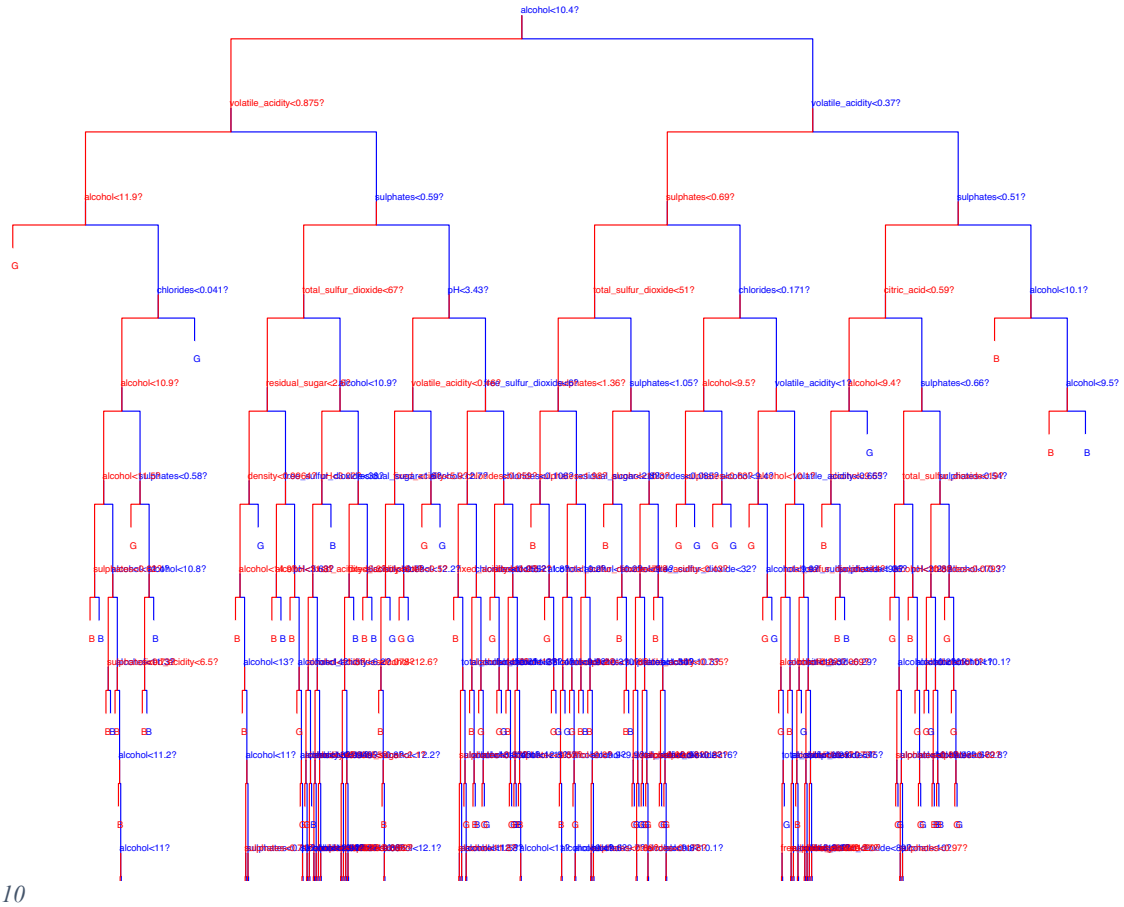


Figure 10

Inside this tree, leaf with a G stands for superior wine and B stands for an inferior wine. The accuracy of this fundamental tree is around 77%.

Summary of the final approach you used and why you chose that approach:

Our goal is to minimize MSE on our testing dataset. So, we always choose the best method to minimize the MSE.

Firstly, we have to mention that the distribution of scores is not even. We can draw the histogram of scores.

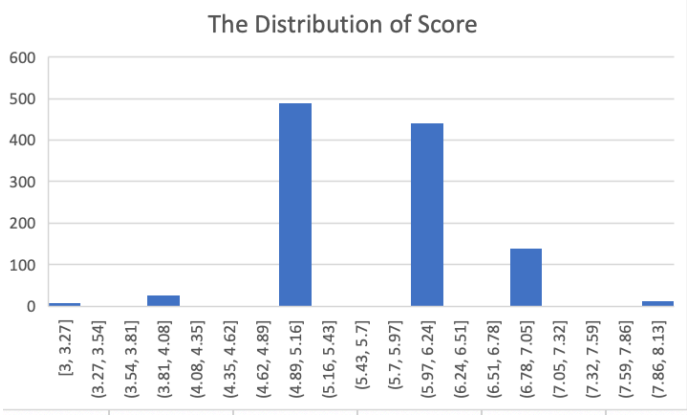


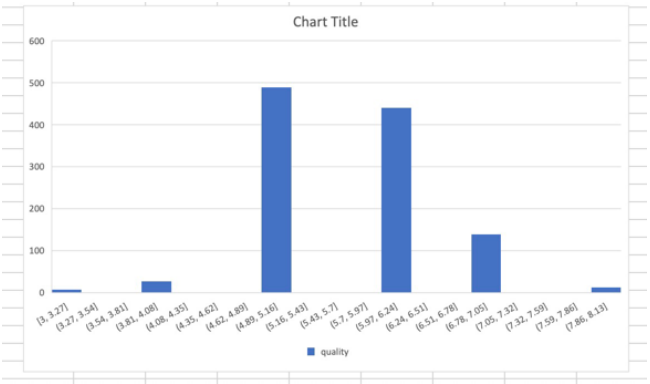
Figure 11

The score approximately complies with normal distribution. So, the extreme scores are rare. The wine with a score of 3, 4, and 8 accounts for less than 5%. Therefore, it is hard for CART to learn what is good wine and what is bad wine. The only way to construct a good enough model is to get a dataset whose distribution is even. Then we can consider removing the majority wine. However, after we remove enough wine with majority score, we find that the number of data becomes very small. That will cause much error.
So, in theory it's impossible to construct a good model using this kind of uneven and small dataset.
We can only expect to minimize the error of the majority score because this model serves for majority.
So, the MSE will inevitably become large.
I conclude that MSE on the entire dataset is:

$$MSE = MSE_{minority} + MSE_{majority}$$

The $MSE_{majority}$ is easy to minimize using Pruning or other algorithm like Multiple Variables Linear Regression. However, according our theory, $MSE_{minority}$ is hard to minimize. So, we just keep it as a constant and don't care about that.

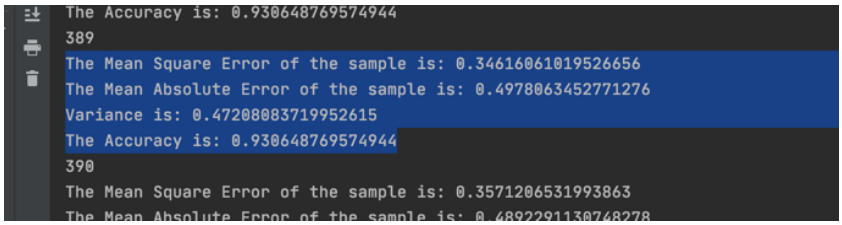
Extra: Removing extreme data
After the pruning program, we find out that the MSE of the sample is at least 0.48 and the MAE is at least 0.54. There must be some other methods to continue reducing these two values. Therefore, we searched on the internet and found a Master's thesis: 林翠香. (2010). *基于数据挖掘的葡萄酒质量识别*. (Doctoral dissertation, 中南大学). In this paper, the author claims that wines with a low score of 3 or 4 and with a high score of 8 can be considered as extreme data, because they only account for a small percentage in the whole dataset.



Distribution of wines with different scores in the dataset

Since regression tree program tends to be more responsible for the majority of all the data in the dataset, the small amount of extreme data can lead to a great deviation in MSE and MAE.

Based on this theory, we excluded all the extreme data in the building tree and pruning process. As a result:



MSE of the program has been reduced to 0.346, and MAE has been reduced to 0.5. It means that removing the extreme data can raise the quality of the program greatly.

Primary tree MSE is 0.65625.

It is too high, so we have to improve our model.

Firstly, we use Weakest Link Pruning to optimize out model.
Tree class:
The node class has the following important attributes.
1. Self.name: store the number of the node, which is used for node searching directly.
2. Self.left / right
3. Self.condition: store the condition of one index of the wine, such as alcohol<10.5.
4. Self.ID: store all the wine's ID that flow through the current node.
5. Self.type: if the node is not a leaf, its type is 'node'. If it is a leaf, its type is 'termianl'.
6. Self.result: store the predict score. Only leaf node has this attribute.
Function:
1. Record ID: store the IDs of flowing wine.
The tree class has the following important attributes.
1. Self.root
2. Self.size: save the number of nodes the tree has.
Function:
1. Add_root.
2. Add_left.

3. Add_right.
4. Delete (p): delete the children of p and return the IDs that p has saved.

The process of pruning tree:

General idea:
For each time of pruning, first put all former 60%of sample wines into the tree and we can get a list of predicted score in the order of the wine number. Then, use RSS Method to calculate each non-leaf node. Store the tuple (node's name, RSS's value) in a 'node_variance' list and sort it. Delete all the zero value (meaningless points) at first. Then, choose the node that has the smallest RSS's value as the pruning point in this round. Then turn the target node into leaf and assign the average score of the original score of all the IDs that the node has stored. The above is a whole process for a pruning. Repeat the manipulation until the length of the 'node_variance' list is less than one. Done.

regPrune class:

The regPrune has the following attributes.
1. Self.item (list): saving all the indexes of a wine.
2. Tree: pointing to the tree we have grown.
3. Data_dict (dict): save the number of the wine and its 11 index's statistics.
4. Cost_node (int): store the number of the children the current node has.
5. Node_list (dict): store both the number and the reference of node (non-leaf).
6. Convert(self.tree.root): find out all the non-leaf node and safe them into a list.
7. Simplify(self): delete all the meaningless nodes (its left and right children give the same predicted value) in the tree.

Functions:
8. Simplify (): delete all the meaningless nodes (its left and right children give the same predicted value) in the tree. In a for-loop, save the tuple ("tup") (node,name, result) in disc_list, where result equals discrim(). The above will be repeated through all the node_list. In another for-loop, if tup[1] equals 0, it means this node is meaningless or say its left and right children give the same predicted value. Finally, call the convert(self,root) to find out all normal nodes.
1. Convert (node): find all the non-leaf node under the current node and save them into a list. If length(node) does not equal 1 and it is not the parent of a leaf, call the convert(node,left) and (convert(node.right) and save the reference in node_list.
2. Yuce (node, i): "node" means the current node. "I" represents the ID of a wine. Given a wine, yuce() can return the predicted value by calling the filter function.
3. Guance (): retrieve the original score a wine.
4. Filter (pointer,a): According to the condition in each node, each wine will flow into a leaf and has a predicted score. Then, store its predicted value into self.predicted_ value. Based on the format of the node.condition (alcohol<10.5), find out the matching index of the wine and get that particular index of the wine. If wine's value is smaller than the condition value, call the filter(a.left). If larger or equal to the standard, call the filter(a.right). Repeat the process until reaching a leaf. Finally, return the self.predict_value.
5. RSST0 (node): this function is based on a formula: RSS(T) >= RSS(TR) + RSS(TL), where T is the current node TL, TR represent its left and right nodes. Firstly, find out all wine ID saved in the left children node. According to the IDs, find its matching original score and calculate the average score among all scores. Then find the variance. Repeat the above process for the right children node.

6. RSST1 (node): if we turn this node into a leaf, calculate its RSST1. By calling yuce() and the node.ID, store all the original score in result_list. Then, calculate the average value and the variance.
7. Length (node): find out the total number of the children node, including the current node.
8. Discrim (): calculate the (RSST1 - RSST0) / (|T0| - |T1|), where T0 means the inferior nodes the current node has and T1 equals 1.
9. Weaknode (): find out the node that has the minimum value of (RSST1 - RSST0) / (|T0| - |T1|). Use self.node_list. By calling discrim() for each time, find the minimum of the (RSST1 - RSST0) / (|T0| - |T1|) and save the number of the node in self.output.
10. processTree (): cut the node in self.output[0]. Use node.ID and calculate the average value of all the original value. Then assign the average score to node.result. Delete node.condition, node.left/reight and turn the node.type from "node" to "terminal". Modify the tree.size by minus self.cost_length (the number of its children node).
11. getNewTree (): return the current tree, which can be used to save in the document.

Logic:
Once you initialize the regPrune Class, the program is pruning.

Tiger Conjecture I: (Raised by our teammate Ruan Zhitai)
When we are pruning the tree, the first few nodes to be cut will be those whose score equals to its children. This is right in theory:

$$R(t) \geq R(t_L) + R(t_R)$$

The equality holds if and only if $\bar{y}(t) = \bar{y}(t_L) = \bar{y}(t_R)$.
So, the RSS decrease of those nodes will be 0, which is the lowest, therefor they should be cut at the beginning.

We got 593 candidate trees by applying Weakest Link Pruning.

Then we will make cross validations: We use the latter 20% of our training dataset to evaluate the goodness of our tree models:

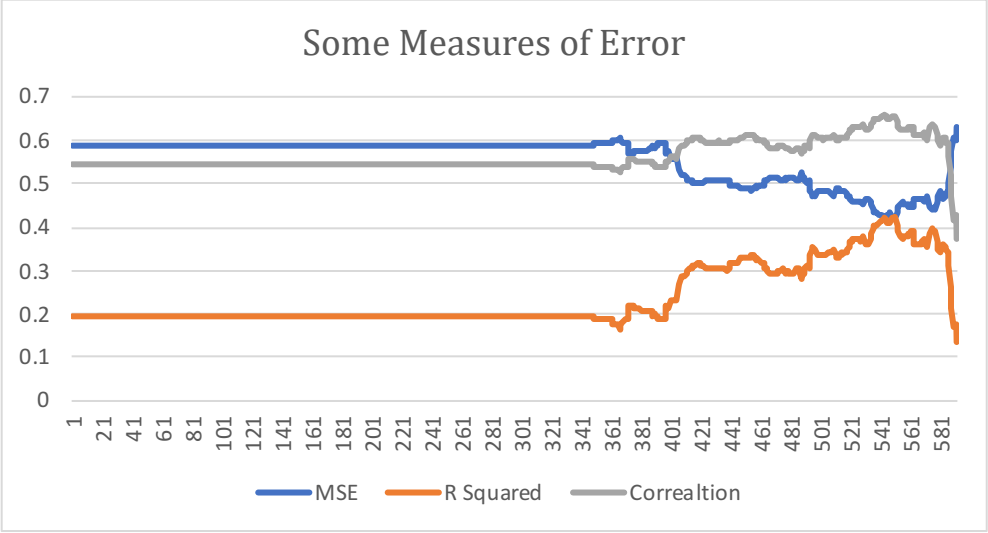


Figure 12

Here we only focus on MSE:

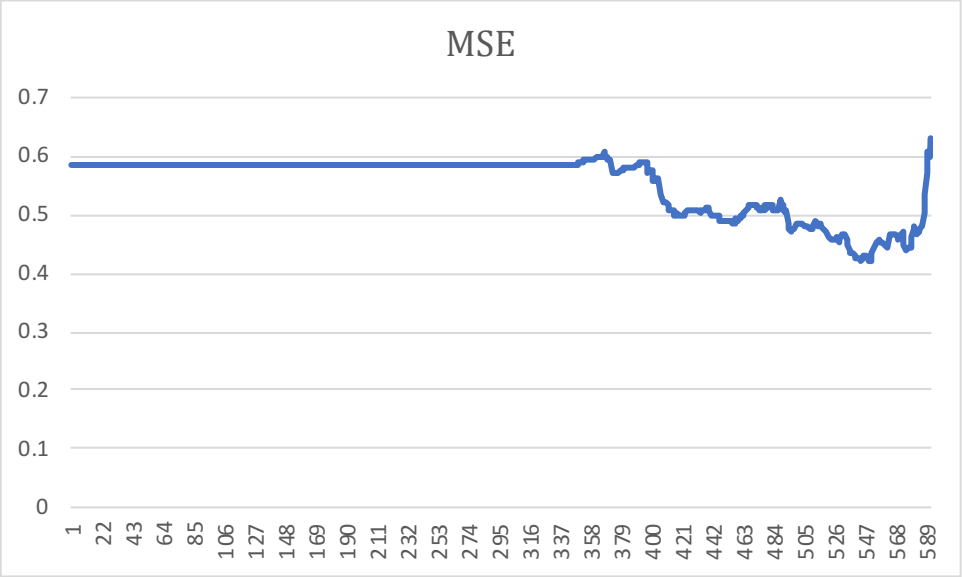


Figure 13

We can find that as we prune the tree, the MSE decreases. Its minimum value is around 0.42067, which belongs to our 551th candidate tree.
However, the MSE increases sharply when we continue pruning the tree. The reason behind is straight-forward: as we continue pruning the tree, it will become a null tree with only one node. That is unreliable.
Due to the property of MSE, we can select the best candidate tree at some point between Tree 1 and Tree 589.
After we finish the pruning process, we can find that:

Summary of the results:

551th tree is the optimized tree with MSE of 0.4206706619509683.
Here we provide the graph of optimized CART tree:

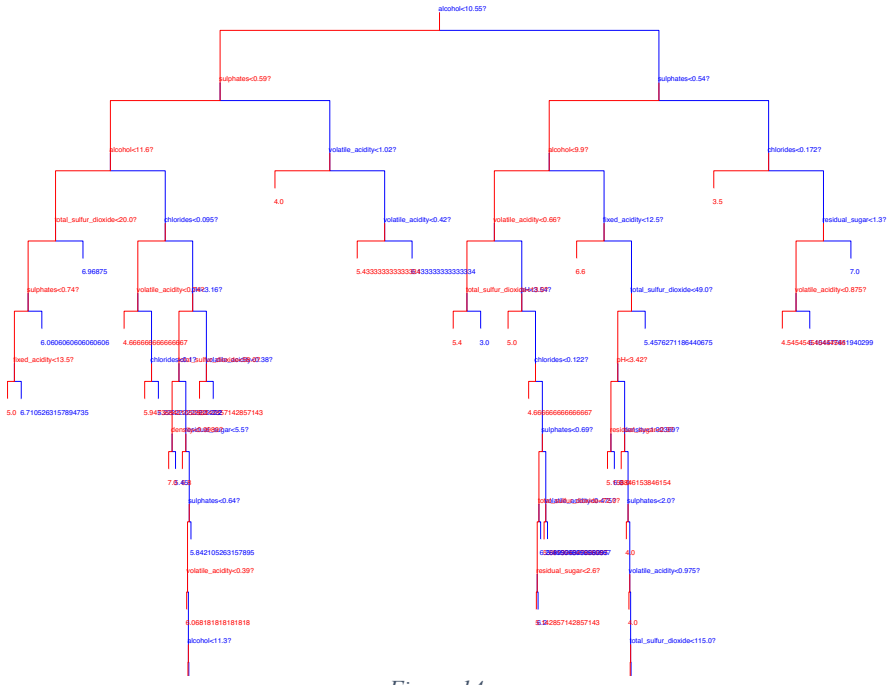


Figure 14

```
def classifier(data_list):
    fixed_acidity = data_list[0]
    volatile_acidity = data_list[1]
    citric_acid = data_list[2]
    residual_sugar = data_list[3]
    chlorides = data_list[4]
    free_sulfur_dioxide = data_list[5]
    total_sulfur_dioxide = data_list[6]
    density = data_list[7]
    pH = data_list[8]
    sulphates = data_list[9]
    alcohol = data_list[10]
    if alcohol<10.55:
        if sulphates<0.54:
            if chlorides<0.172:
                if residual_sugar<1.3:
                    return 7.0
                else:
                    if volatile_acidity<0.875:
                        return 5.104477611940299
                    else:
                        return 4.545454545454546
            else:
                return 3.5
        else:
            if alcohol<9.9:
                if fixed_acidity<12.5:
                    if total_sulfur_dioxide<49.0:
                        return 5.4576271186440675
                    else:
                        if pH<3.42:
                            if density<1.00369:
                                if sulphates<2.0:
                                    if volatile_acidity<0.975:
                                        if total_sulfur_dioxide<115.0:
                                            if fixed_acidity<7.9:
                                                return 5.05
                                            else:
                                                if fixed_acidity<8.3:
                                                    return 5.714285714285714
                                                else:
                                                    return 5.138888888888889
                                        else:
                                            return 4.958333333333333
                                    else:
                                        return 4.0
                                else:
                                    return 4.0
                            else:
                                return 6.0
                        else:
                            if residual_sugar<2.3:
                                return 6.0
                            else:
                                return 5.153846153846154
                else:
                    if volatile_acidity<0.66:
                        if pH<3.54:
                            if chlorides<0.122:
                                if sulphates<0.69:
                                    if volatile_acidity<0.475:
                                        return 5.466666666666667
                                    else:
                                        return 5.891304347826087
                                else:
                                    if total_sulfur_dioxide<72.0:
                                        return 6.2682926829268295
                                    else:
                                        if residual_sugar<2.6:
                                            return 6.2
                                        else:
                                            return 5.142857142857143
                            else:
                                return 4.666666666666667
                        else:
                            return 5.0
                    else:
                        if total_sulfur_dioxide<13.0:
                            return 3.0
                        else:
                            return 6.6
            else:
                if volatile_acidity<0.66:
                    if pH<3.54:
                        if chlorides<0.122:
                            if sulphates<0.69:
                                if volatile_acidity<0.475:
                                    return 5.466666666666667
                                else:
                                    return 5.891304347826087
                            else:
                                if total_sulfur_dioxide<72.0:
                                    return 6.2682926829268295
                                else:
                                    if residual_sugar<2.6:
                                        return 6.2
                                    else:
                                        return 5.142857142857143
                        else:
                            return 4.666666666666667
                    else:
                        return 5.0
                else:
                    if total_sulfur_dioxide<13.0:
                        return 3.0
```

```
            else:
                return 5.4
        else:
            if sulphates<0.59:
                if volatile_acidity<1.02:
                    if volatile_acidity<0.42:
                        return 6.133333333333334
                    else:
                        return 5.433333333333334
                else:
                    return 4.0
            else:
                if alcohol<11.6:
                    if chlorides<0.095:
                        if pH<3.16:
                            if volatile_acidity<0.38:
                                return 7.2
                            else:
                                return 6.142857142857143
                        else:
                            if total_sulfur_dioxide<58.0:
                                if residual_sugar<5.5:
                                    if sulphates<0.64:
                                        return 5.842105263157895
                                    else:
                                        if volatile_acidity<0.39:
                                            if alcohol<11.3:
                                                return 6.258064516129032
                                            else:
                                                return 7.333333333333333
                                        else:
                                            return 6.068181818181818
                                else:
                                    return 6.8
                            else:
                                if density<0.9989:
                                    return 5.45
                                else:
                                    return 7.0
                    else:
                        if volatile_acidity<0.74:
                            if chlorides<0.1:
                                return 5.222222222222222
                            else:
                                return 5.947368421052632
                        else:
                            return 4.666666666666667
                else:
                    if total_sulfur_dioxide<20.0:
                        return 6.96875
                    else:
                        if sulphates<0.74:
                            return 6.0606060606060606
                        else:
                            if fixed_acidity<13.5:
                                return 6.7105263157894735
                            else:
                                return 5.0
```

Use it over testing data:

551th tree is the optimal tree, MSE is 0.48498296954827563

This data is considerable, and acceptable.

Secondly, we use **Multiple Variables Linear Regression**.
We write out the target function, and we consider a 11-dimensional vector:

$$\boldsymbol{Y} = \beta_1 + \beta_1 \boldsymbol{X_1} + \beta_1 \boldsymbol{X_1} + \cdots + \beta_{12} \boldsymbol{X_{12}} + \boldsymbol{u}$$

Where $\boldsymbol{u}$ is the error. Our mission is to minimize the error $\boldsymbol{u}$.

$$\boldsymbol{u} = \boldsymbol{Y} - (\beta_1 + \beta_1 \boldsymbol{X_1} + \beta_1 \boldsymbol{X_1} + \cdots + \beta_{12} \boldsymbol{X_{12}})$$

Moreover, we calculate the RSS:

$$RSS = \sum \boldsymbol{u^2} = \boldsymbol{u'u} = \boldsymbol{Y'Y} - \boldsymbol{\beta' X' Y}$$

Then we calculate the derivative of RSS, and we can find its minimum:

$$\boldsymbol{\beta} = (\boldsymbol{X'X})^{-1} \boldsymbol{X'Y}$$

So, we use some function to deal with those matrixes. Then we can find the optimized $\boldsymbol{\beta}$ very easily.
After we finish this module, we have to optimize it. For example, to avoid some extreme fitting like:

*Score = -2304.0 + 140.0 * fixed_acidity + 106.0 * residual_sugar + -2560.0 * chlorides + -18.0 * free_sulfur_dioxide + 14.0 * total_sulfur_dioxide + 81.0 * density + -56.0 * pH + 256.0 * sulphates + 2.5 * alcohol*

This is not general and may cause a huge error.
We should prevent it from happening. The reason for that is simple: some variables do not have a strong correlation with the score value. So, we only pick those pointers whose relative coefficients are greater than 0.3. In practice, we got good results.

And we find that the accuracy of this model increases as the number of samples increase. So we should consider apply this algorithm near the root node.

So, we can apply this algorithm to those trees generated by Weakest Link Pruning. **We use a range of trees and apply this algorithm to them.**

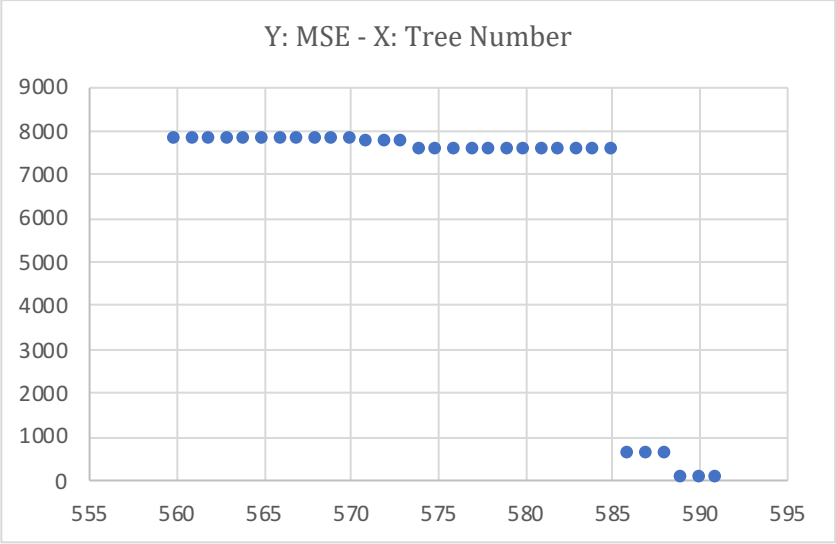


Figure 15

As we can see from the graph, as N increases, the MSE will drop significantly at some point. So it is possible to find a considerably small MSE as we iterate over those trees.

M5 algorithm.
General Idea: change the node.result (only for leaf node) into a function than can calculate the M5 value. Then, use the left train-data to calculate the MSE and figure out the best one.

The **Predictor class** has the following attributes:
1. Item (list): store the 11 indexes.
2. Tree: point to the tree we have grown.
3. Data_dict: the statistic of all wines' 11 indexes.
4. Predict_value: store predict value.

Function:
1. Yuce (i): "i" represents the ID of a wine. Given a wine, yuce () can return the predicted value by calling the filter function.

2. Filter (pointer, i): "I" means the list of 11 indexes' statistic. According to the condition in each node, each wine will flow into a leaf and has a predicted score. Then, store its predicted value into self.predicted_ value. Based on the format of the node.condition (alcohol<10.5), find out the matching index of the wine and get that particular index of the wine. If wine's value is smaller than the condition value, call the filter(a.left). If larger or equal to the standard, call the filter(a.right). Repeat the process until reaching a leaf. Finally, return the self.predict_value.

3. **Regress(): Different from two dimensional linear regression, when we deal with higher dimensional regression, we calculate based on matrix. According to the formula defined above, we have to calculate the transpose of matrix, the multiplication of matrix and the inversion of matrix.**

$$\boldsymbol{\beta} = (\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{Y}$$

Using this formula, we plug all 11 attributes of all wine into matrix *X*. Then we plug all score value y into the matrix Y. The last step is simple. We first calculate the transpose of X, and then multiply it by X. The third step is to calculate the inversion of that product. The fourth step is to calculate the product of X' and Y. The final step is to calculate the product of A and B. By now we have known the value of $\boldsymbol{\beta 1}, \boldsymbol{\beta 2}, \dots, \boldsymbol{\beta 12}$.

The accuracy of the regression is better than CART in generalization. For example, we generate a set of β using all data of our training data. The MSE is roughly 0.40 for test data set. What a high accuracy!

Then, how about combining M5 and CART together? We tried it out. The results are amazing. We use 80% of our training data set, and use the remaining 20% of data to validate our model. To our surprise,

We find that:

589th M5 tree is the optimized tree with MSE of 0.34714662814824765
(based on 20% remaining training data).

To our surprise, MSE dropped by **19%** in comparison with the optimized tree entirely generated by Weakest Link Pruning algorithm.

Then we use this tree as our final model.

We test it by using our **testing data**.

By using M5 algorithm, the final MSE is 0.4363702808129912
(based on test data)

So, eventually we have made final MSE decreased by (0.4849 – 0.4363) / 0.4849 = 10%.

This is a huge leap of our model.

Therefore, our final MSE is: 0.4363

Here we provide the graph of optimized M5 Regression Tree:

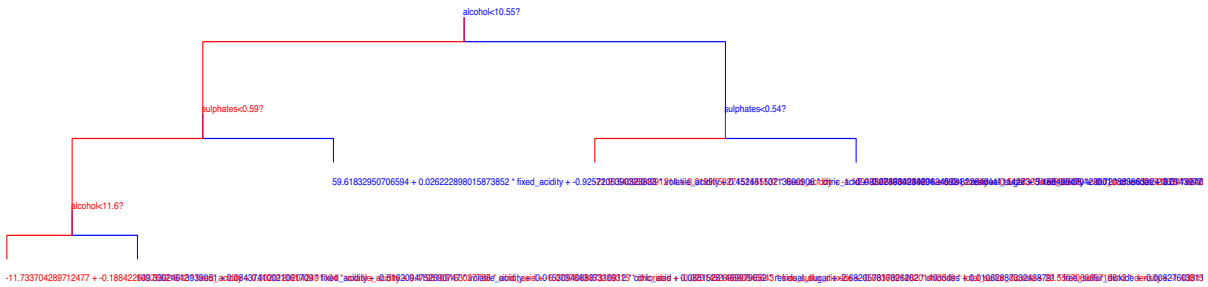


Figure 16

The program script is a lot simpler than CART:

```
def classifier(data_list):
    fixed_acidity = data_list[0]
    volatile_acidity = data_list[1]
    citric_acid = data_list[2]
    residual_sugar = data_list[3]
    chlorides = data_list[4]
    free_sulfur_dioxide = data_list[5]
    total_sulfur_dioxide = data_list[6]
    density = data_list[7]
    pH = data_list[8]
    sulphates = data_list[9]
    alcohol = data_list[10]
    if alcohol<10.55:
        if sulphates<0.54:
            return -2.933273680944694 + 0.01822646441114273 * fixed_acidity + -0.7208838633241825 *
volatile_acidity + -0.5203821999160141 * citric_acid + -6.657339719968945 * chlorides +
0.00091079135148453 * free_sulfur_dioxide + 0.0005793860249294069 * total_sulfur_dioxide +
9.481423352379352 * density + -0.7646153903338586 * pH + 2.2274997011806477 * sulphates +
0.0890950943021380 2 * alcohol
        else:
            return -21.915409804991214 + 0.019975027151545532 * fixed_acidity + -1.1001064667442844 *
volatile_acidity + -0.5116363406863638 * citric_acid + 0.015236347624075108 * residual_sugar + -
1.2281389275228207 * chlorides + 0.001241668697254554 * free_sulfur_dioxide + -
0.0037677370701494906 * total_sulfur_dioxide + 27.295827815687517 * density + -0.3712818548675614 *
pH + 0.3211683764309896 * sulphates + 0.20420135737668943 * alcohol
    else:
        if sulphates<0.59:
            return 59.61832950706594 + 0.026222898015873852 * fixed_acidity + -0.9257205390323833 *
volatile_acidity + 0.4524511071369 0906 * citric_acid + -0.024934239206245934 * residual_sugar +
5.168499470423001 * chlorides + 0.014397766870685835 * free_sulfur_dioxide + 0.0014394099297018181
* total_sulfur_dioxide + -56.99638553155819 * density + 0.00389109151996081 * pH + -
0.1152621685992532 * sulphates + 0.20994446749688223 * alcohol
        else:
            if alcohol<11.6:
                return 109.33024613939051 + 0.08437410021061709 * fixed_acidity + -0.5162094752590747 *
volatile_acidity + -0.015309408837310912 * citric_acid + 0.08815281469979652 * residual_sugar + -
2.682057817826262 * chlorides + 0.01062887332438791 * free_sulfur_dioxide + -0.008276033115767939 *
```

*total_sulfur_dioxide + -103.22758416703437 * density + -0.19995124981176104 * pH + 0.23634288730825403 * sulphates + -0.024381593530279133 * alcohol*

else:

*return -11.733704289712477 + -0.18842264376621642 * fixed_acidity + 0.30201202042491104 * volatile_acidity + 0.4198109153027758 * citric_acid + -6.0737843453289315 * chlorides + -0.023145269890845643 * free_sulfur_dioxide + -0.003609140201403349 * total_sulfur_dioxide + 22.550706969719613 * density + -1.4381538745844864 * pH + 1.3796576458834267 * sulphates + 0.15114036736264325 * alcohol*

Conclusions:

CART algorithm performs just so-so on this training dataset, because the data-set is not evenly distributed and the sample are far too small, so CART doesn't give a good prediction due to the existence of extreme value.

Given that we cannot tell the extreme value in theory, however, we can use some method to minimize the error caused by the majority wine. So, we use M5' algorithm to further improve the performance of our model. Its nature is to replace those constant prediction value with a smooth curve(actually it is in higher dimension and hard to visualize). By combing CART and M5 algorithm, we can make 10~20% improvement on the performance of our model.

However, the running time is a big restriction. In our experiment, we firstly grow a complete regression tree. And then we prune the tree. The third step is to apply M5' algorithm. After we carry out the experiment, we can make a difference:

1. **Grow the regression tree with a max depth of 4 or less.**
2. **Every time a new node is added, save the tree as a copy.**
3. **Apply M5' algorithm to the first 100 candidate trees.**
4. **Select the tree with lowest MSE based on remaining train dataset.**

By using this procedure, we can save 90% of time.

We also find an interesting fact:

The CART algorithm perform well on its training data, using with MSE less than 0.2 but with MSE of roughly 0.6~0.8 on testing data! Note that if we only guess the score without any principle, the MSE will be 0.67! So, it is amazing.

The Linear Regression algorithm perform equally on both training data and testing data, with MSE of roughly 0.4. It did well in generalization.

In conclusion, we got a fairly good model based on this kind of terrible data-set.

References

[1] https://en.wikipedia.org/wiki/Decision_tree_learning

[2] http://www.stats.ox.ac.uk/~flaxman/HT17_lecture13.pdf

[3] http://www.math.snu.ac.kr/~hichoi/machinelearning/lecturenotes/CART.pdf

[4] Breiman, L., Friedman, J., Stone, C. J., & Olshen, R. A. (1984). Classification and regression trees. CRC press.