**CSC1001 Team Project**

1. **Data and Question:** Red Wine Quality Problem

We are interested in how to use Classification and Regression Tree (CART) to predict whether the quality score of the wine is above 6 or not. We use 1119 sample data of red wine containing 12 features of wine, which can be used to predict the quality of the wine. The features are shown as follows:

1. Fixed acidity: most acids involved with wine or fixed or nonvolatile (do not evaporate readily)

2. Volatile acidity: the amount of acetic acid in wine, which at too high of levels can lead to an unpleasant, vinegar taste

3. Citric acid: found in small quantities, citric acid can add 'freshness' and flavor to wines

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

5. Chlorides: the amount of salt in the wine

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

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

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

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

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

11. Alcohol: the percent alcohol content of the wine

12. Quality: output variable (based on sensory data, score between 0 and 10)

1. **Some of the approaches**
   1. **Save simple date:**

Each simple data is saved in dictionary. The keys of dictionary are features, and values are the value of each features. All simple data dictionarys are saved in list. In the simple data list, we can easily sort it by one feature, and find the value of one feature.

* 1. **Pre-pruning**

Pre-pruning is to improve the generalization performance of the decision tree by adjusting the bifurcation termination conditions

**2.2.1. General tree termination condition**

When the elements assigned to the same subset have the same tag ( Yes: the value of "quality" is larger than 6. No: the value of "quality" is small or equal to 6. ) , the bifurcation is terminated.

**2.2.2. Additional conditions to prevent overfitting**

In order to prevent overfitting with the training set, it is necessary to set conditions for the tree to terminate the bifurcation early. Here I added two conditions:

condition1: If the gini index calculated during the last bifurcation is less than a threshold, it means that the subset obtained by the last bifurcation is sufficiently similar, and can be regarded as having the same tag, and the bifurcation can be ended erarly here.

condition2: If the number of elements in the obtained subset is less than 5, and the tags are still not unified, it means continuing the bifurcation can only obtain the judgment logic of a special case, which is not suitable for generalization, so the bifurcation can be terminated early here.

( At this point I finished try\_prepruning\_ver1.py )

**2.2.3. Additional conditions for improving accuracy**

When only using the above conditions to build trees, it can achieve an accuracy rate of about 87.5%. In order to improve the accuracy, I added the condition for evaluating the bifurcation. The accuracy rate before bifurcation is compared with the accuracy rate after bifurcation. If the bifurcation can improve the accuracy, then the bifurcation will be made. If the bifurcation will reduce the accuracy, it will not be made.

The first attempt of this additional condition: I wanted to have a sufficiently rich data base, so I used the training set as both a training set and a validation set. I used the same data set ( data\_list ) for classification and evaluation accuracy.After modifying the code with this idea, the branching of the tree was reduced, but at the same time the accuracy dropped to 87.0%.

( At this point I finished try\_prepruning\_ver2.py )

The second attempt of this additional condition: After thinking again, I felt that this evaluation method can only improve the fit with the training set.Therefore, the generalization performance of the tree did not been improved, and the accuracy also decreased. So when I entered the data of train.csv, I divided the data into two parts, one part was stored in the training set ( data\_set ), and the other part was stored in the verification set. During the tree-building process, the validation set and the training set are divided into subsets according to the same criteria. The data of the training set is the basis of the tree bifurcation. The data of the verification set is used to judge the rationality of the bifurcation. After modifying the code with this idea, the branching of the tree was reduced again, but at the same time the accuracy increased to 87.7%.

( At this point I finished try\_prepruning\_ver3.py. This is also the final

version of the pre-prune code. It has relatively few branches and

relatively high accuracy.)

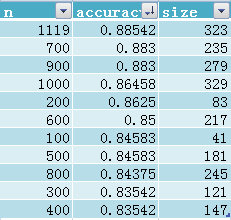
* 1. **Non-pruning**

Non-pruning allows the tree to perfectly classify the training set.

**2.3.1. Split simple data:**

Since simple data is large, we can use while loop to reduce the amount of data. It can save time when we test our program, and we can use different part of simple data to train Classification and Regression Tree.

Approaches (the amount of sample data is regarded as n, accuracy sorted in ascending order):



(All data can be found in try\_nonpruning\_ver2.py)

Those data shows that the noise of Red Wine Quality training sample data is very small.

**2.3.2. General tree termination condition**

If the amount of sample data is one, which means the accuracy of this type of sample data is 100%, the tree will stop growing at here.

**2.3.3. Update condition**

To save unnessary steps for pruning and prevent overfitting with the sample data, the general condition can be updated. This approach compromises the merits of pre-pruning method.

Update condition: If all sample data in parent tree can only be divide into one type (quality small or equal to 6, quality larger than 6), this small set of data are well designed.

**2.3.4. Without pruning**

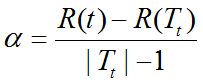
After all conditions mentioned above, each leaf node only contains one type value. It is very easy to cuase overfitting. But it seems like the data for testing our CART do not contain much noise. The accuracy for this version of Non-truning method is not bad, which is above 88.54%. And the number of leaf-node is 162.

(All data can be found in try\_nonpruning\_ver1.py).

Defeat: The number of node in CART is significantly large than other versions. It is very time consuming for building this tree. This method may not be applied for other data which has more noise.

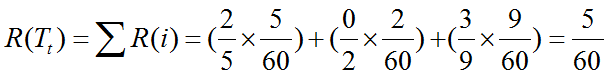
**2.3.5. Some methods I thought to improve the accuracy and avoid over-fitting**

**2.3.5.1**.Cost-Complexity Pruning:

 α: Gain Error Rate

: the number of leaf node in the subtree

 :the error cost of the node

 :the sum of the error cost of leaf node in the subtree

Step1: Compute α for each non-leaf node in the complete decision tree. Cut down the subtree which has smallest α by recursion and save all pruned trees.

Step2: Compute the accuracy of those pruned trees. The complete Post-pruning tree is the one with the highest accuracy.

**2.3.5.2.** Random Forest:

Random Forest is to build a lot of decision trees to form a "forest" of decision trees, and to make decisions by voting multiple trees. This method can effectively improve the classification accuracy of new samples.

Step 1: The sample data is sampled by putting back to get multiple sample sets. Specifically, n samples (including possible repeated samples) are randomly selected from the original N training samples.

Step 2: m features are randomly selected from the candidate features as the candidate features for decision-making under the current node, and the features of the training samples are selected from these features. Each sample set is used as the training sample to construct the decision tree. After a single decision tree generates a sample set and determines the characteristics, it uses the CART algorithm to calculate without pruning.

Step 3: After obtaining the required number of decision trees, the random forest method votes the output of these trees, and takes the class with the most votes as the decision-making of the random forest.

This is a very effective method. Since in this project, we are not allowed to use third-party packages, such as random, we can not use Rondom Forest.

**2.4. Post-pruning**

post-pruning is to prevent the decision tree from overfitting, improve the accuracy of classification.

**2.4.1the method used in post-pruning**

calculate the gini of nodes and compare it with the minGain that have already been set. If the gini of the nodes is smaller than the minGain, then merge the two nodes.

1. **Final approach**

The last version of pre-pruning program. (try\_prepruning\_ver3.py.).

1. **Summary and Why**

**4.1. Compare different versions and summary**

For the non-pruning version of the tree: The highest accuracy can be achieved but there are many nodes in the tree.

For the pre-pruning version of the tree: It greatly reduces the number of nodes in the tree and saves the time of training. But the accuracy is reduced slightly.

**4.2. Reason**

Although the highest accuracy can be achieved in the non-pruning version of the tree, this method is not suitable for other data with more noise. It will cause over-fitting**,** andthe accuracy will be reduced by the noise in the training sample data.

The pre-pruning version of the tree also has high accuracy and saves time. Although the accuracy is not the highest one, we can get more general results by applying to the different training samples. It is more suitable for solving practical problems.

1. **Principle and Logic**

1. Using variable "node" as a pointer, build a binary tree with a linked list.

2. The element of the node stores various information about the criteria, results, etc. of the last bifurcation. The left/right of the node store the reference to the subset bifurcate from this node.

3. The bifurcation criterion is the one with the smallest gini index among all bifurcation combinations.

4. When the elements in the subset are sufficiently similar, or continue to bifurcation will reduce the accuracy, the bifurcation is terminated early. This can both reduce the number of nodes and ensure accuracy**.**

1. **Summary of the results**

The program without pruning has the highest accuracy. The accuracy of the pre-pruning program is slightly lower than that of no pruning.

1. **Conclusions**

Pruning will reduce the accuracy rate, indicating that the noise in this set of data is very small. However, considering that noise may still exist, in order to ensure the generalization performance of the tree, the final decision was to use the pre-prune program.

1. **Contribution and Workload of each member**

石雯岚（119010265）：

report part: All about pre-pruning

code part: try\_prepruning\_ver1.py & try\_prepruning\_ver2.py & try\_prepruning\_ver3.py & main.py

王子洋（119010327）：

report part: All about Non-pruning, Data and Question, Save simple date

code part: try\_nonpruning\_ver1.py & try\_nonpruning\_ver2.py

卢怡宁（119010218）

report part:All about post-pruning

code part:pruning-ver1.py(This one is just programmed for showing the process of post- pruning, so I import third-party packages too quickly get the data. I did not use third-party packages when post-pruning)