# A Classification Problem on Craft Beer

Lars Keijzers[**]
Ruben Koelewijn
s1075816
s1105300

**Figure 1: Different types of craft beer**

## ABSTRACT

Craft beer enthusiasts often face the challenge of accurately identifying the style of beer they are drinking. This classification task is complicated by subjective human taste perceptions. To address this issue, we employ data mining techniques, specifically classification algorithms, to automate the categorization of craft beers. The project utilizes a dataset containing information on 3197 unique beers, including attributes such as Alcohol By Volume (ABV), bitterness, sweetness, and more. We explore the use of Decision Tree and Random Forest classification algorithms to predict beer styles. Through extensive tuning and evaluation, we achieve a mean accuracy of 0.756 for the Decision Tree and 0.812 for the Random Forest. While both models demonstrate accuracy, the Random Forest emerges as slightly more effective in our classification task.

[*]Both authors contributed equally to this research.

**ACM Reference Format:**
Lars Keijzers and Ruben Koelewijn. 2023. A Classification Problem on Craft Beer. In *Data Mining Project 2023*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

Craft beers, we all love them. However, we, Lars and Ruben, always struggle knowing what style of beer we are drinking without having to look like a fool and read the packaging or ask the bartender for the backstory of the beer. Because trying to do this yourself by tasting beers, which has the problem of a human being subjective, does not result in accurate classification, we will try to use the data mining technique of classification to do our work for us.

## 2 RELATED PREVIOUS WORK

We did not find research which aims to do exactly the same as we wish to accomplish, but we did manage to find two papers which both have some of the elements we want to accomplish.

### 2.1 Factors influencing the choice of beer: A review

The first article we want to cover is research on what flavours and variables make a beer drinker choose one beer over the other. The paper Factors influencing the choice of beer: A review by Maria Isabel Betancur in 2020 talks about how the sensory aspects of the beer affect choice, but also psychological, the packaging and environmental reasons why a consumer might prefer a certain beer. Environmental reasons would be the difference between buying a beer in the supermarket or buying a beer at a bar. Even though the study was not able to draw a clear cut conclusion on what drives a consumers beer choice, it did show what factors influence a choice, which might prove to be useful when we want to look at what classifies one beer as a particular style.

### 2.2 Decision Trees for Wine Classification

The research paper The Combination and Comparison of Neural Networks with Decision Trees for Wine Classification by researchers of the University of Fiji is on combining neural networks with decision trees to classify wine. In their research they start of by training the neural networks on the chemical data they were able to extract from the wine. Next they use rule-based classifiers in combination with their trained neural networks to predict from which of the three wine farmers the wine came. The researchers were able to do this with a stunningly high accuracy of 98.7 ± 1.2% for their neural networks, and their Decision Trees scored an accuracy of 96.8 ± 1.7%. Sadly they did not provide any other metrics except for the accuracy tests they provided, other metrics such as a ROC-curve or confusion matrices.

## 3 APPLICATION DOMAIN AND RESEARCH PROBLEM OVERVIEW

### 3.1 Application domain

In this project we focus on classifying craft beers on their attributes, making use of two different classification algorithms: Decision Tree Classification and Random Forest Classification. Our goal is to predict unseen beers to a certain accuracy.

### 3.2 Research Problem

Regardless of their popularity, easily classifying craft beers is not as it simple as it may seem. This is because their are different factors at play which could lead to subjectivity when a human is trying to classify it. We also need to figure out which classification algorithm we want to apply, and which one is more accurate for our specific data set.

The research problem is to use a classification algorithm which can predict craft beer styles.

## 4 DATA SET AND PREPROCESSING METHODS

In terms of finding a data set, we were mostly looking for a data set which contained different styles and clear information about every beer. So a data set with just three or four attributes about the beer type would not suffice, or a data set with reviews because that would be a subjective attribute and would not lead us to accurate predictions.

### 4.1 Data Set

The data set we choose to use is the Beer Profile and Ratings data set, which is a data set containing 3197 unique beers and their reviews and characteristics. We found this data set on www.kaggle.com. The data set contains beer and each of them has been given the following information: `Name, Style, Brewery, Beer Name (Full), Description, ABV, Min IBU, Max IBU, Astringency, Body, Alcohol, Bitter, Sweet, Sour, Salty, Fruits, Hoppy, Spices, Malty, review_aroma, review_appearance, review_palate, review_taste, review_overall, number_of_reviews`. For classification, not all of these attributes are relevant and some might even interfere with our classification process. SO we will mostly be looking at the attributes `ABV` until `Malty`, which all contain standardized numerical data.

### 4.2 Preprocessing Methods

If we were use the data set as is, our classification would be difficult. This is because there would be 111 different classes we would need to classify. Luckly these classes are smaller types of beer which can also be classed as a bigger class. For example, if we have 'Barleywine - American' and 'Barleywine - English', we simplify this into just 'Barleywine'. This leads us to have just have 43 classes instead of the original 111.

We also take out some of the columns which add nothing to our analysis. We take out columns 'Brewery', 'Beer Name (Full)', 'Description', which are just text types which describe our beer very nicely, but do not add much to our research.

Next we split the set into our `X` and `y`, where the data will be in the matrix `X` and the classes, in this case the simplified beer names, into a vector `y`.

## 5 APPROACH/METHODS AND RATIONALE

Before we are able to apply any of our algorithms to classify our set, we need to know what parameters give us the most accurate results. For both algorithms there a different ways to tune so we will cover the settings for each algorithm. We will be using the decision tree classifier from the `sklearn.tree.DecisionTreeClassifier` class and the random forest algorithm from the `sklearn.ensemble.RandomForestClassifier` class. We chose these specific parameters because in the documentation of both algorithms it is referrenced that these have the most impact on the accuracy. Every train/test split will be a 80/20 split, using the `sklearn.model_selection.train_test_split` class to ensure we get accurate training and test sets. We choose the 80/20 split because this results in enough data for a model to be trained accurately but also allowing the test set big enough so that the accuracy will be representative. For every iteration we run the model, we change the `random_state` which allows us to generate different training and test sets every time.

### 5.1 Decision Tree Classifier

For the Decision Tree Classifier there are many different settings we can play with, but we will only be looking at three, namely `max_depth`, `min_samples_split` and `ccp_alpha`. These variables do the following:

- `max_depth`: This variable sets a maximum depth on the decision tree, limiting it on how deep it can go. This can prevent overfitting and keep the decision tree simple.

- `min_samples_split`: This variable sets a minimum amount of samples there are needed before the tree create a split, which also helps against overfitting.
- `ccp_alpha`: This variable is different then the other two, it gives a value to the tree which prunes the tree. This is done to improve the accuracy.

To determine the `min_samples_split` and `max_depth`, we run the algorithm for multiple different values and different test sets. Then we take the average of those runs and see which value leads to the highest accuracy. When doing this the other variables will stay at their default settings.

#### 5.1.1 *Deciding* `max_depth`. .

For `max_depth` we will be running the algorithm from `max_depth=1` to `max_depth=25`. As you can see in Figure 2, the accuracy does not change significantly after a depth of 12 and has it's peak around 18. To prevent overfitting, we will pick our `max_depth` variable to be set to 17.

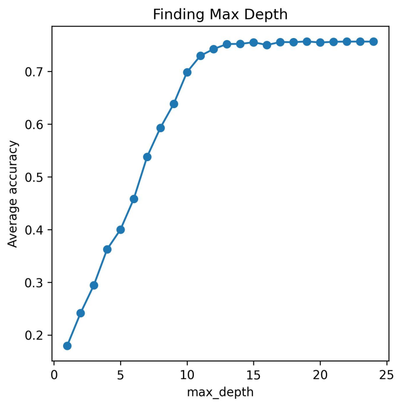

**Figure 2: Plot of the max depth and average accuracy.**

#### 5.1.2 *Deciding* `min_samples_split`. .

We will do the same for `min_samples_split` as we did for the depth, and run it from 2 to 25, it has to start from 2 because there is a minimun of 2 for the `min_samples_split` variable. As we can see in the plot in Figure 3, the accuracy climbs from 2, but takes a dip after 5. Even though the accuracy difference is not extreme, we will still be taking our value of 3 for the `min_samples_split`.

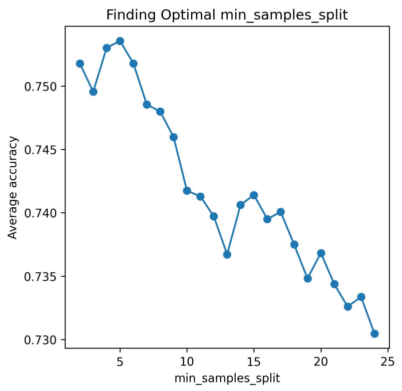

**Figure 3: Plot of `min_samples_split` and average accuracy.**

#### 5.1.3 *Deciding optimal texttccp_alpha.* .

For the `cpp_alpha` we take a different approach, becuase instead of just taking the default settings, we will be inputting the two values for `max_depth` and `min_samples_split` . This is because we will be pruning the tree and we want to prune our tree and not the default tree so the results will be better if we input our optimal settings.

After running the algorithm for the values of `ccp_alpha` we found, more information on how we found these are in Section A.3. We have found that an `ccp_alpha` of 0.00074 appears to be optimal for our tree.

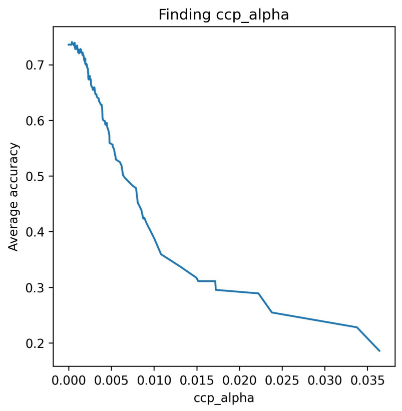

**Figure 4: Plot of `ccp_alpha` and average accuracy.**

### 5.2 Random Forest Classifier

The Random Forest Classifier has different attributes which are interesting for us to look at then the Decision Tree Classifier. For the forest we will be looking at `max_depth` and `bootstrap`. These variables do the following:

- `max_depth`: This variable sets a maximum depth on each tree of the random forest, limiting it on how deep it can go. This can prevent overfitting and keep the decision tree simple.

- `bootstrap`: This variable allows the use of bootstrapping while building the trees of the forest. If set to `False`, the whole dataset is used to build each tree.

Unlike with the Decision Tree Classifier, we will not prune the Random Forest Classifier, because that would lead to us having to prune every tree in the forest individually, which is not the aim of this project. We will run the forest with default settings except for the variable we want to examine multiple times and take the average of the accuracy score we receive.

### 5.2.1 *Deciding* `max_depth`. .

To find the optimal `max_depth`, we will check from 1 to 30, to find out which one gives the most accurate result. As with all the other variables, we run the classification multiple times and take the average accuracy to prevent overfitting. As we can see in Figure 5, around a depth of 15 the the accuracy stagnates and depth 20 gives us the best result. So we will decide to use `max_depth=17`.

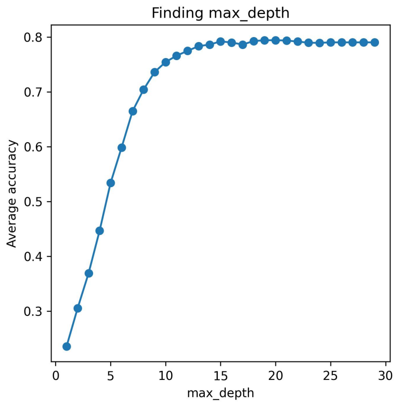

**Figure 5: Plot of the average accuracy and `max_depth`**

### 5.2.2 *Bootstrap vs No Bootstrap*. .

The `bootstrap` variable is a boolean, which means we only have to test for the values `True` and `False`. After running the forest 20 times for both values, we see that *without* bootstrapping, we get more accurate results. This is shown in Table 1. Despite the difference being minimal, we choose to use not bootstrapping.

| | `True` | `False` |
|---|---|---|
| Accuracy | 0.790 | 0.813 |

**Table 1: Mean Accuracy for Bootstrap and Non Bootstrap**

## 6 MAIN RESULTS AND INSIGHTS

Now that we are done with the fine tuning of our models, in section 5, we can use our models to predict our beer. After running both algorithms 50 times with different train/test sets, we received the plot shown in figure 6.

If we take all these different accuracy values, we can calculate the mean and the standard deviation:

The confidence interval of both algorithms, From 0.743 to 0.755 for the Decision Tree and from 0.807 to 0.817 for the Random Forest, is relatively narrow.

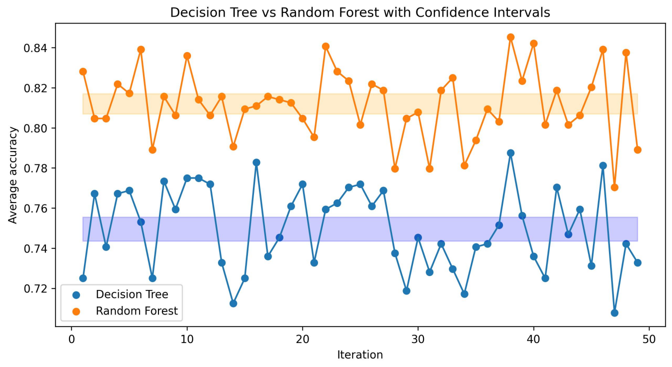

**Figure 6: Plot of the average accuracy and multiple iterations of predictions**

| | Mean | Standard Deviation | Confidence Interval |
|---|---|---|---|
| Decision Tree | 0.749 | 0.020 | (0.743, 0.755) |
| Random Forest | 0.812 | 0.017 | (0.807, 0.817) |

**Table 2: Mean Accuracy, Standard Deviation and Confidence Interval for Decision Tree and Random Forest**

The standard deviation of the Decision Tree, 0.020, appears to be slightly higher then that of the Random Forest, 0.017. This means that the values of the Decision are spread apart more.

Because we do not have a binary classification problem, we cannot use ROC-curves to determine our accuracy. We can however try to use confusion matrices:

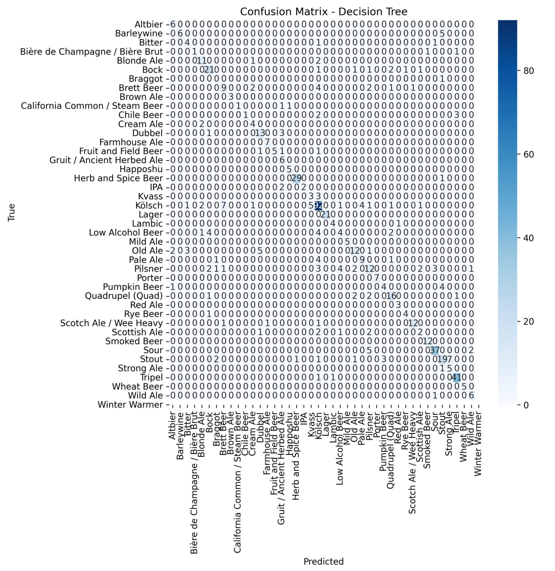

**Figure 7: Confusion Matrix for Decision Tree**

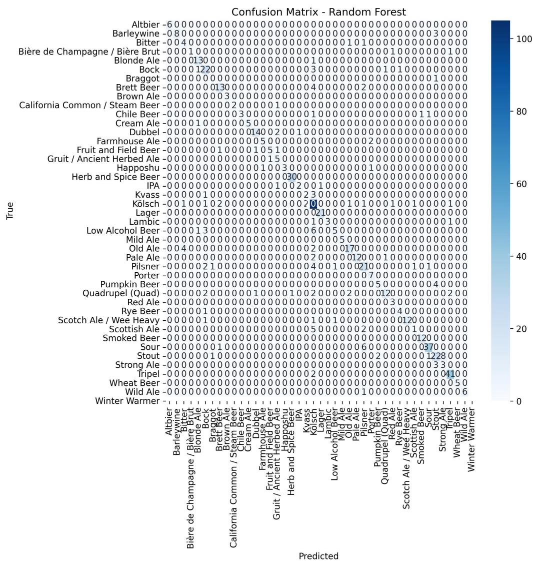

**Figure 8: Confusion Matrix for Random Forest**

## 7 ANALYSIS OF RESULTS AND EVALUATION

After looking at our results we have managed to gather in section 6, we can now compare the two algorithms.

### 7.1 Analysis of the Results

When we look at the mean of the Decision Tree accuracy (0.749) and the mean of the Random Forest accuracy (0.812), we see that the Random Forest has a better mean accuracy. Furthermore we can see that the confidence intervals drawn in figure 6 are narrow, which means that the algorithms are reliable which gives a high level of confidence in our mean accuracy values.

The standard deviation of the two algorithms only differs 0.03, as seen in table 2, which means that the accuracy values are spread out nearly the same, even though it does not look like that is the case in Figure 6. This means that both algorithms are almost equally sensitive to the random states of the train/test sets.

When looking at the confusion matrices in Figure 7, it looks like there are no abnormally big amount of False Positives or False Negatives for the Decision Tree algorithm. This might also be because we have a large amount of classes and a confusion matrix is not extremely effective when that is the case. The same holds for Figure 8, which is the confusion matrix for the Random Forest algorithm. So the confusion matrices do not help us determine which of the two algorithms is more accurate in our case.

### 7.2 Evaluation

Our metrics of calculating how efficient the algorithms were not extensive enough for us to be able draw a concise reason *why* one algorithm might perform better on this set then the other. We suspect this is because there are to many classes which might be to much for a classification problem and could have been handled by a (deep) neural network, as mentioned in the article in section 2.2.

## 8 CONCLUSION

Both algorithms have been accurate with predicting the style of beer on the given data set. Where the Decision Tree classifier had a lower mean accuracy then the mean accuracy of the Random Forest classifier , which leads us to conclude that the Random Forest classifier is slightly more accurate in our case. When trying to use other types of accuracy measures, we did not find any significant differences. So we managed to find algorithms to classify our craft beers, but there could be more work on still making them more accurate.

## 9 POSSIBLE FUTURE DIRECTIONS

In this project we compared two different types of classifiers, which both have proven to be fairly accurate for our data set, however in the future there is a possibility for us to use other classifiers such as kNearest Neighbours or neural networks which might have a higher accuracy and lead to an even more efficient algorithms of classifying craft beers.

Another possible future direction is to focus more on pre-pruning and post-pruning. Although we have done some post-pruning, there might be more ways for us to make our classifiers more accurate then they are now.

Lastly, there is also room for the model to not just classify beer, but also alcoholic drinks such as wine, liquor or even more specific styles of beer we did not cover because they might have been left out of the data set.

## A CODE INSIGHTS

In this section we will be explaining some of the import parts of Python code we used to generate results.

### A.1 Preprocessing our data

In section 4.2 we mentioned combining classes with the same prefix. The code used to do that is the following:

```
common_prefix = np.unique([style.split(' - ')[0]
                          for style in df['Style']])
style_mapping = {style: style.split(' - ')[0]
                    for style in df['Style']}
df['StyleSimple'] = df['Style'].map(style_mapping)

label_encoder = LabelEncoder()

y_encoded = label_encoder.fit_transform(df['StyleSimple'])
```

First we create a list of all the unique styles of beers, next we find which of them have a common prefix. After that we snip the suffix off and combine turn the old styles in to the simple styles.

### A.2 Finding `max_depth`

The process of calculating the `max_depth` we did in sections 5.2.1, 5.1.1 and other variables all follow a similar principle, so we will only discuss how we found the `max_depth` in section 5.1.1.

```
max_depth_list = []
for depth in range(1, 25):
    current_depths = []
    for i in range(1,15):
      X_train, X_test, y_train, y_test = train_test_split(X,
            y_encoded, test_size=0.2, random_state=i)
      dtc = tree.DecisionTreeClassifier(criterion="gini",
        max_depth=depth, random_state=1)

        dtc = dtc.fit(X_train, y_train)

        y_pred_test = dtc.predict(X_test)
      accuracy_test = accuracy_score(y_test, y_pred_test)
        current_depths.append(accuracy_test)
    max_depth_list.append(np.mean(current_depths))


print("Optimal depth with accuracy: ", max(max_depth_list),
max_depth_list.index(max(max_depth_list)))
```

This code tries depth 1 to 24, and for every depth it calculates the accuracy of the prediction 15 times, with 15 different train/test sets. After that we take the mean of the 15 runs and add it to our list `max_depth_list`. Then at the end we check the highest accuracy and at which depth this accuracy occurred.

## A.3 Finding possible values of `ccp_alpha`

As mentioned in section 5.1.3, we have found specific values of `ccp_alpha` to prune the Decision Tree Classifier.

First we run the algorithm with our settings without pruning, this is done to create a tree we want to prune and find which values we want to try.

```
split = 5
depth = 13
X_train, X_test, y_train, y_test = train_test_split(X,
    y_encoded, test_size=0.2, random_state=1)
dtc = tree.DecisionTreeClassifier(criterion="gini",
            min_samples_split=split, max_depth=depth,
            random_state=0)
dtc.fit(X_train, y_train)
score = dtc.score(X_test, y_test)

path = dtc.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
```

The method `cost_complexity_pruning_path` is part of the `scikit-learn` library and is used to compute the pruning path for decision trees. It returns a "path" of effective alphas for pruning, along with the corresponding total impurities at each alpha level.

Next we check the accuracy for all these possible values.

```
mean_alphas = []
for ccp_alpha in ccp_alphas:
    current_alphas = []
    for i in range(1,15):
        X_train, X_test, y_train, y_test =
          train_test_split(X, y_encoded, test_size=0.2,
            random_state=0)
      dtc = tree.DecisionTreeClassifier(criterion="gini"
            ,min_samples_split=split, max_depth=depth,
        ccp_alpha=ccp_alpha, random_state=0)
        dtc = dtc.fit(X_train, y_train)
        y_pred_test = dtc.predict(X_test)
      accuracy_test = accuracy_score(y_test, y_pred_test)
        current_alphas.append(accuracy_test)
    mean_alphas.append(np.mean(current_alphas))
```

This results a list with mean values of accuracy and then we will take the value of `ccp_alpha` which gave us the highest accuracy.

## A.4 Getting the results

After applying the same technique we applied in section A.2 for all the other variables, we have now found the optimal settings and can run our two algorithms.

```
    accuracy_dtc = []
accuracy_rfc = []
for state in range(1, 50):
   X_train, X_test, y_train, y_test = train_test_split(X,
    y_encoded, test_size=0.5, random_state=state)
    dtc = tree.DecisionTreeClassifier(criterion="gini",
            max_depth=depthDTC, random_state=0,
        min_samples_split=splitDTC, ccp_alpha=ccp_alpha)
    dtc = dtc.fit(X_train, y_train)

    rfc = RandomForestClassifier(criterion="gini",
        bootstrap=bootstrap, max_depth=depthRFC, random_state=0)
    rfc = rfc.fit(X_train, y_train)

    y_pred_test_dtc = dtc.predict(X_test)
   accuracy_test_dtc = accuracy_score(y_test, y_pred_test_dtc)
    accuracy_dtc.append(accuracy_test_dtc)

    y_pred_test_rfc = rfc.predict(X_test)
   accuracy_test_rfc = accuracy_score(y_test, y_pred_test_rfc)
    accuracy_rfc.append(accuracy_test_rfc)

meanT = np.mean(accuracy_dtc)
meanF = np.mean(accuracy_rfc)

confidence_dtc = stats.t.interval(0.95, len(accuracy_dtc) - 1,
loc=np.mean(accuracy_dtc), scale=stats.sem(accuracy_dtc))
confidence_rfc = stats.t.interval(0.95, len(accuracy_rfc) - 1,
loc=np.mean(accuracy_rfc), scale=stats.sem(accuracy_rfc))

std_dev_dtc = np.std(accuracy_dtc)
std_dev_rfc = np.std(accuracy_rfc)
```

We run both algorithms on the same test and every iteration we change the train/test sets. We do this for 50 iterations to prevent that a certain test set might be preferable for one algorithm in specific. After we have calculated the accuracy we add that to the accuracy list. Then we use those values to calculate the mean, standard deviation and the confidence interval.

All code can be found on our Github reposition, https://github.com/LarsDeLars/DataMiningProject

## REFERENCES

[1] Ruthgn. (2020). *Beer Profile and Ratings Data Set*. Kaggle. Retrieved from https://www.kaggle.com/datasets/ruthgn/beer-profile-and-ratings-data-set

[2] Betancur, M. I. (2020). *Factors influencing the choice of beer: A review*[4][4][5][5]. Food Research International, 137, 109527. https://doi.org/10.1016/j.foodres.2020.109527

[3] Chandra, R., Chaudhary, K., & Kumar, A. (2007). *The Combination and Comparison of Neural Networks with Decision Trees for Wine Classification*1. In Proceedings of the 2007 International Conference on Data Mining, Las Vegas, USA, pp. 10-17.