Problem 1

1.1. Problem statement

1.1.1. Dataset

The dataset is red variant of the Portuguese "Vinho Verde" wine appraisal samples. The goal is to model wine quality based on physicochemical tests.

```
Data size:  (1599, 12)
```

11 Features:

```
['fixed acidity' 'volatile acidity' 'citric acid' 'residual sugar'
 'chlorides' 'free sulfur dioxide' 'total sulfur dioxide' 'density' 'pH'
 'sulphates' 'alcohol']
```

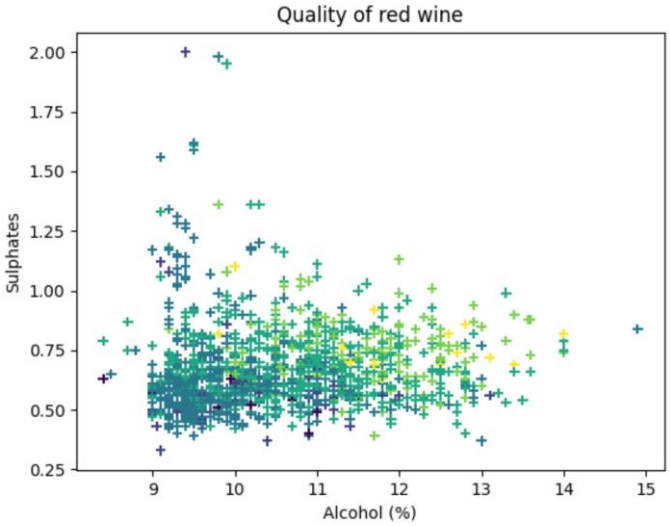
Target: quality (score between 0 and 10), only 3 to 8 samples in this dataset

What interests me is that this dataset can be viewed as both multi-classification and binary-classification problems in order to satisfy different requirements. I might get some interesting findings comparing the results of this two problems. Moreover, the classes are not balanced, there are much more normal wines than excellent or poor ones. I'm curious will or how this unbalance affects the result.

Problem 1 is the multi-classification model. I'll do the latter in Problem 2.

1.1.2. Visualize

```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality
```



I used two dominant features, which I got from decision tree, to visualize the dataset. The boundaries are not clear. Obviously, it will be a tough task.

1.1.3. Split training / test set

I randomly split the dataset with 0.3 of it as test set and keep the same distribution in both sets.

1.2. k-Nearest Neighbors

1.2.1. Default model

I don't know exactly which algorithm will benefit from scaling. So I decided to test it myself.

I used the default model (parameters) to fit my training set at first:

```
Training score:0.634
Test score:0.490
Training score (scaled):0.689
Test score (scaled):0.579
```

From the results above, I take the answer is yes for KNN

1.2.2. Tune hyper-parameters

I used the Grid search cross validation to find the optimal k:

```
parameters = {'knn__n_neighbors': np.arange(1, 50)}
```

```
Best parameters {'knn__n_neighbors': 40}
Best score 0.590703074951954
```

1.2.3. Optimal model

1. Predict

   ```
   Tuned training score:0.605
   Tuned test score:0.562
   ```

   Using the optimal parameter, I got the training/test score above. Since the k-40 is larger than default, it is normal to see lower training score. While test score getting lower too, it is a sign of underfitting.

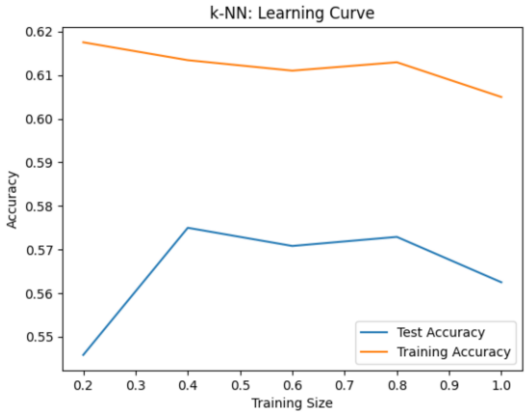
2. Confusion matrix

```
Confusion Matrix
[[  0   0   3   0   0   0]
 [  0   0  12   3   1   0]
 [  0   0 134  70   0   0]
 [  0   0  57 120  15   0]
 [  0   0   5  39  16   0]
 [  0   0   0   5   0   0]]
```

	precision	recall	f1-score	support
3	0.00	0.00	0.00	3
4	0.00	0.00	0.00	16
5	0.64	0.66	0.65	204
6	0.51	0.62	0.56	192
7	0.50	0.27	0.35	60
8	0.00	0.00	0.00	5
accuracy			0.56	480
macro avg	0.27	0.26	0.26	480
weighted avg	0.53	0.56	0.54	480

   The reason why KNN performed badly is that class 3, 4, 8 don't have sufficient data (training set has the same distribution as test set). And this is the prove to support unbalaced classification indeed affects the result, espacialy for KNN.

3. Learning curve



The X axis is the portion of training set I used to plot the learning curve.

It might because of the underfitting that test accuracy did not benefits from larger training set.

1.3. Decision Trees

1.3.1. Default model

```
Training score:1.000
Test score:0.606
Training score (scaled):1.000
Test score (scaled):0.604
```

This result shows the characteristic of decision trees: good at considering all possibilities but tend to overfit data. And there is no need to scale.

1.3.2. Tune hyper-parameters

```
param_grid = {'criterion': ['gini', 'entropy'],
              'max_depth': np.arange(3, 20, 1),
              'min_samples_split': np.arange(2, 10, 1),
              'min_samples_leaf': np.arange(1, 10, 1)}
```

```
Best parameters {'criterion': 'entropy', 'max_depth': 8, 'min_samples_leaf': 4, 'min_samples_split': 9}
Best score 0.5791039397821909
```

1.3.3. Optimal model

1. Predict

```
Tuned training score:0.751
Tuned test score:0.594
```

After pruning, overfitting has been alleviated a little.

2. Confusion matrix

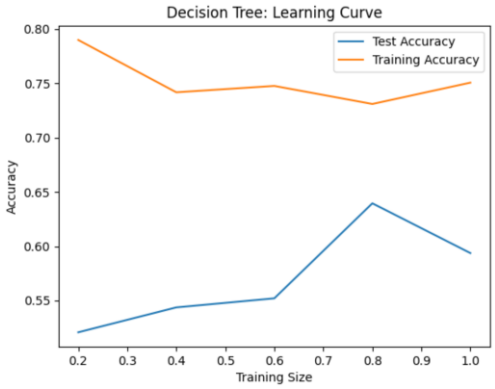
Confusion Matrix

```
[[  0   1   2   0   0   0]
 [  0   2   7   4   3   0]
 [  2   5 124  72   1   0]
 [  0   4  38 138  12   0]
 [  1   1   3  34  21   0]
 [  0   0   0   3   2   0]]
```

	precision	recall	f1-score	support
3	0.00	0.00	0.00	3
4	0.15	0.12	0.14	16
5	0.71	0.61	0.66	204
6	0.55	0.72	0.62	192
7	0.54	0.35	0.42	60
8	0.00	0.00	0.00	5
accuracy			0.59	480
macro avg	0.33	0.30	0.31	480
weighted avg	0.60	0.59	0.59	480

See the score for class 4, the reason why Decision tree got a better result than KNN is that it can make predictions for small classes.

3. Learning curve



1.4. Boosting

1.4.1. Default model

I used Gradient boosting classifier:

```
Training score:0.908
Test score:0.675
```

Since the weak learner Gradient boosting used is Decision tree. It has Decision tree's advantage and disadvantage. By combining multiple weak learners, variance and bias will be reduced.

And there is no need to scale as well.

1.4.2. Tune hyper-parameters

Procedures:

1. Fix learning_rate and Decision tree parameters to search for optimal n_estimators

```
param_test1 = {'n_estimators': range(20, 81, 10)}
gb = GradientBoostingClassifier(learning_rate=0.1,
                                subsample=0.8,
                                max_depth=5,
                                min_samples_leaf=4,
                                min_samples_split=5,
                                random_state=SEED)
Best parameters {'n_estimators': 60}
Best score 0.6318225496476617
```

2. Fix learning_rate and n_estimators to search for optimal Decision tree parameters

```
param_test2 = {'max_depth': range(3, 14, 2), 'min_samples_split': range(2, 20, 2)}
Best parameters {'max_depth': 9, 'min_samples_split': 18}
Best score 0.6451913837283791
```

Fix max_depth

```
param_test3 = {'min_samples_split': range(18, 30, 2), 'min_samples_leaf': range(1, 10, 1)}
Best parameters {'min_samples_leaf': 4, 'min_samples_split': 18}
Best score 0.6451913837283791
```

3. Lower learning_rate and increase n_estimators to make the model more robust

```
learning_rate = 0.1, n_estimators = 60:
Tuned training score: 1.0
Tuned test score: 0.6916666666666667
learning_rate = 0.05, n_estimators = 120:
Tuned training score: 1.0
Tuned test score: 0.6979166666666666
learning_rate = 0.01, n_estimators = 600:
Tuned training score: 1.0
Tuned test score: 0.7020833333333333
```

The result is significantly better than Decision tree. While as I used larger number of estimators, the training time increased significantly as well.

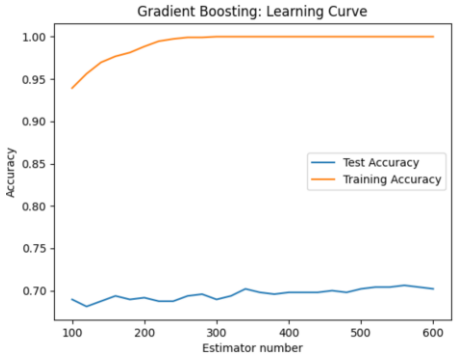
1.4.3. Optimal model

1. Confusion matrix

```
Confusion Matrix
 [[  0   0   3   0   0   0]
 [  0   1   8   6   1   0]
 [  0   5 161  38   0   0]
 [  1   0  33 145  13   0]
 [  0   0   4  27  29   0]
 [  0   0   0   3   1   1]]
```

	precision	recall	f1-score	support
3	0.00	0.00	0.00	3
4	0.17	0.06	0.09	16
5	0.77	0.79	0.78	204
6	0.66	0.76	0.71	192
7	0.66	0.48	0.56	60
8	1.00	0.20	0.33	5
accuracy			0.70	480
macro avg	0.54	0.38	0.41	480
weighted avg	0.69	0.70	0.69	480

2. Learning curves



Just a little bit improvement for test accuracy.

1.5. Neural Networks

1.5.1. Default model

Training score:0.602

Test score:0.600

Training score (scaled):0.694

Test score (scaled):0.602

Since the scaled model has not converged completely, if I set a larger max_iter, it will have an even better score. As a result, Neural networks will benefit from scaling.

1.5.2. Tune hyper-parameters

```
param_test1 = {'mlpc__solver': ['lbfgs', 'adam'],
               'mlpc__alpha': [0.00001, 0.0001, 0.001, 0.01],
               'mlpc__hidden_layer_sizes': [(10,), (20,), (100,), (5, 2), (100, 30), (50, 50, 50), (50, 100, 50)],
               'mlpc__activation': ['tanh', 'relu']}
```

```
Best parameters {'mlpc__activation': 'relu', 'mlpc__alpha': 0.01, 'mlpc__hidden_layer_sizes': (50, 100, 50), 'mlpc__solver': 'adam'}
Best score 0.6130725496476617
```

1.5.3. Optimal model

1. Predict

   Tuned training score:0.993

   Tuned test score:0.633

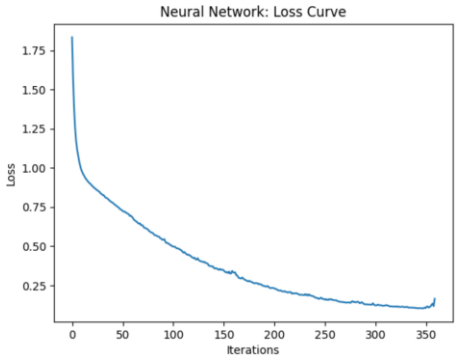
2. Confusion matrix

Confusion Matrix

```
[[  0   0   3   0   0   0]
 [  0   1  11   3   1   0]
 [  1   5 147  48   3   0]
 [  0   1  44 123  23   1]
 [  0   0   5  22  33   0]
 [  0   0   1   2   2   0]]
```

	precision	recall	f1-score	support
3	0.00	0.00	0.00	3
4	0.14	0.06	0.09	16
5	0.70	0.72	0.71	204
6	0.62	0.64	0.63	192
7	0.53	0.55	0.54	60
8	0.00	0.00	0.00	5
accuracy			0.63	480
macro avg	0.33	0.33	0.33	480
weighted avg	0.62	0.63	0.62	480

3. Learning curves



1.6. Support Vector Machines

1.6.1. Default model

I tried three kernels:

1. Radial base function kernel

```
Training score:0.510
Test score:0.494
Training score (scaled):0.668
Test score (scaled):0.646
```

It's clear that SVM need scaling.

2. Linear kernel

```
Training score (scaled):0.587
Test score (scaled):0.583
```

There is no hyper-parameter for linear kernel. This is the best it can get. So I skipped it.

3. Polynomial kernel

```
Training score (scaled):0.684
Test score (scaled):0.600
```

Just a little worse than RBF, I chose it as candidate.

1.6.2. Tune hyper-parameters

```
param_test1 = {'SVM__kernel': ['rbf', 'poly', 'sigmoid'],
               'SVM__C': [0.1, 1, 10, 100],
               'SVM__gamma': [1, 0.1, 0.01, 0.001]}
```

```
Best parameters {'SVM__C': 1, 'SVM__gamma': 0.1, 'SVM__kernel': 'rbf'}
Best score 0.6058656310057656
```

Choose RBF kernel:

```
param_test2 = {'SVM__C': [1, 2, 3],
               'SVM__gamma': [0.1, 0.2, 0.3, 0.4]}
```

```
Best parameters {'SVM__C': 2, 'SVM__gamma': 0.3}
Best score 0.6210762331838564
```

1.6.3. Optimal model

1. Predict

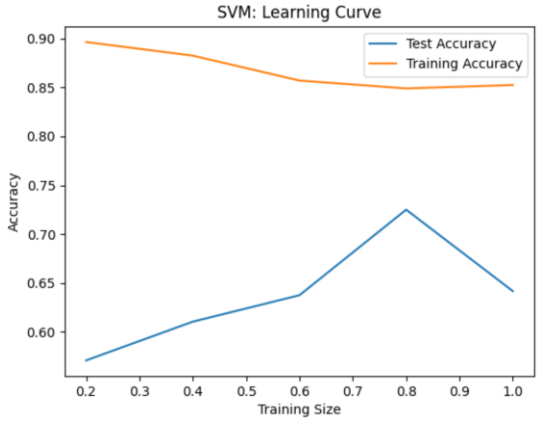
```
Tuned training score:0.853
Tuned test score:0.642
```

2. Confusion matrix

```
Confusion Matrix
[[  0   0   2   1   0   0]
 [  0   1  11   3   1   0]
 [  0   1 146  56   1   0]
 [  0   0  45 137  10   0]
 [  0   0   4  32  24   0]
 [  0   0   0   3   2   0]]
```

	precision	recall	f1-score	support
3	0.00	0.00	0.00	3
4	0.50	0.06	0.11	16
5	0.70	0.72	0.71	204
6	0.59	0.71	0.65	192
7	0.63	0.40	0.49	60
8	0.00	0.00	0.00	5
accuracy			0.64	480
macro avg	0.40	0.32	0.33	480
weighted avg	0.63	0.64	0.62	480

3. Learning curves



1.7. In conclusion

Algorithms	Training accuracy	Test accuracy	Training time (s)	Scaling	Linearity
KNN	0.605	0.563	0.016	Yes	Non-linear
Decision Tree	0.751	0.594	0.016	No	Non-linear
Gradient Boosting	1	0.702	20.946	No	Non-linear
Neural Network	0.993	0.633	6.742	Yes	Non-linear
SVM	0.853	0.642	0.098	Yes	Linear

Boosting got the best score, while since it uses a chain of estimators, the training time is the longest.

Advanced algorithms like Neural networks and SVM performed better. Because Neural networks use an iterative approach, the training time is longer than SVM.

KNN being affected the most by unbalanced classes.

Problem 2

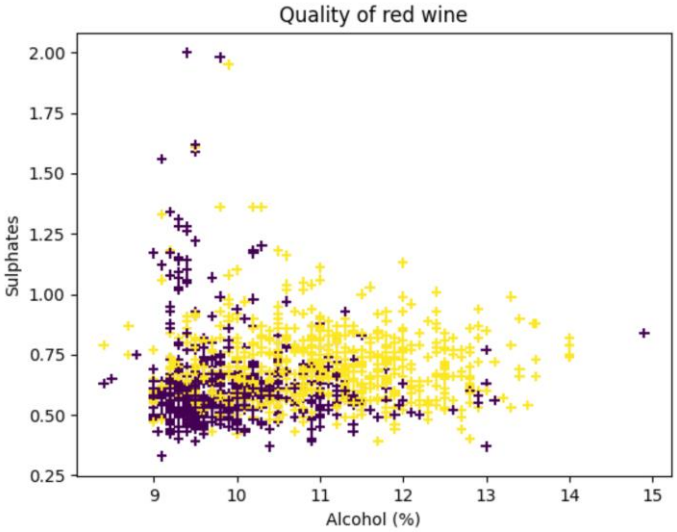
2.1. Problem statement

2.1.1. Dataset

I used the same dataset as Problem 1. Instead, I turned the 10 point scale into dichotome variable: quality that 6 or higher as good/1, and the rest as not good/0.

2.1.2. Visualize

```
1    855
0    744
Name: quality
```



2.1.3. Split training / test set

I split the dataset using the same strategy as in Problem 1.

2.2. k-Nearest Neighbors

2.2.1. Default model

```
Training score (scaled):0.806
Test score (scaled):0.738
```

Much better than the results of multi-classification problem.

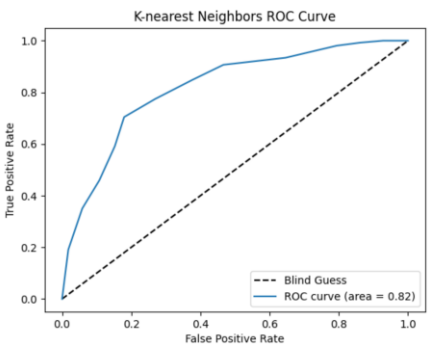
2.2.2. Optimal model

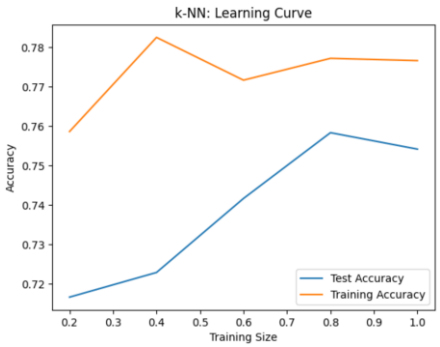
1. Predict

   ```
   Tuned training score:0.777
   Tuned test score:0.754
   ```

   Although training score decreased with a larger K - 12 used, the ability to generalize increased. Compared to Problem 1, KNN performed much better when the classes are balanced.

2. ROC curve and learning curve





We now can see the trend that the model learns better with larger training set.

2.3. Decision Trees

2.3.1. Default model

```
Training score:1.000
Test score:0.717
```

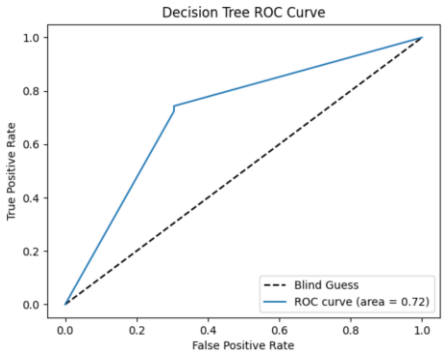
Overfitting as usual if we don't prune it.

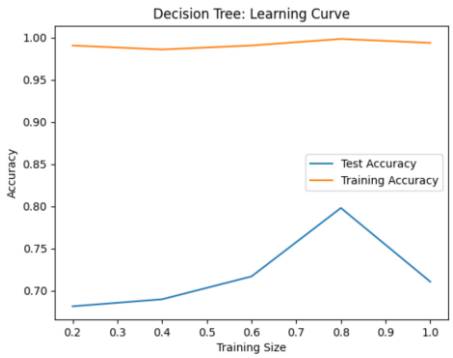
2.3.2. Optimal model

1. Predict

```
Tuned training score:0.994
Tuned test score:0.710
```

2. ROC curve and learning curve





2.4. Boosting

2.4.1. Default model

I used Gradient boosting as well:

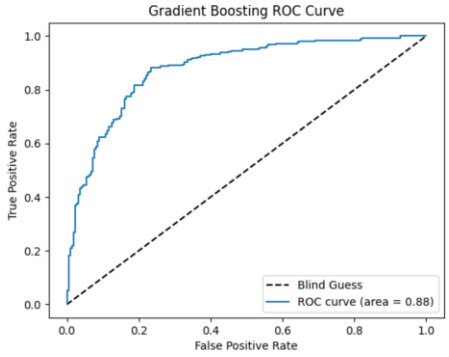
```
Training score:0.876
Test score:0.767
```

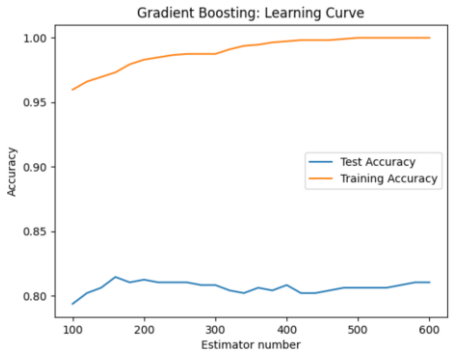
2.4.2. Optimal model

1. Predict

```
learning_rate = 0.05, n_estimators = 160:
Tuned training score: 1.0
Tuned test score: 0.81875
```

2. ROC curve and learning curve





2.5. Neural Networks

2.5.1. Default model

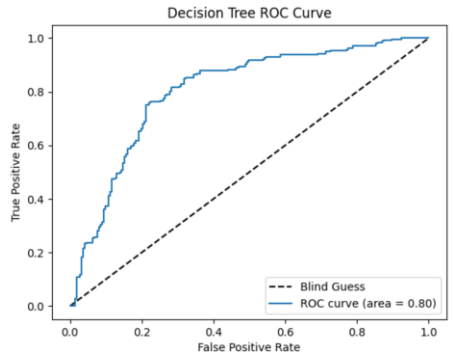
```
Training score (scaled):0.809
Test score (scaled):0.756
```

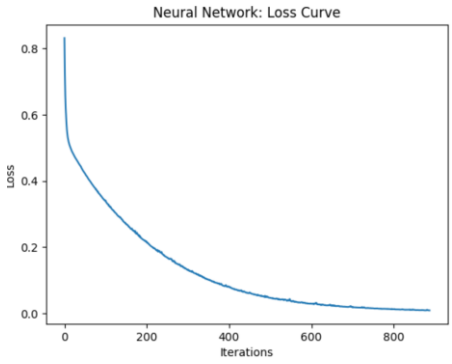
2.5.2. Optimal model

1. Predict

```
Tuned training score:1.000
Tuned test score:0.767
```

2. ROC curve and learning curve





2.6. Support Vector Machines

2.6.1. Default model

I tried three kernel as well:

1. Radial base function kernel

```
Training score (scaled):0.802
Test score (scaled):0.752
```

2. Linear kernel

```
Training score (scaled):0.739
Test score (scaled):0.740
```

It seems that a two-dimensional linear model is not suitable for this problem.

3. Polynomial kernel

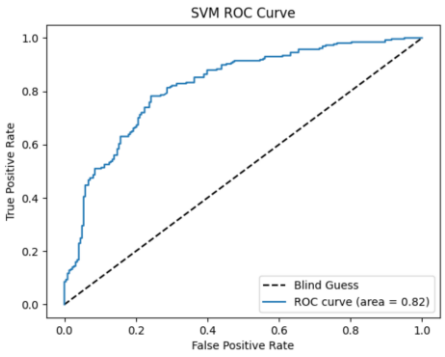
```
Training score (scaled):0.808
Test score (scaled):0.746
```

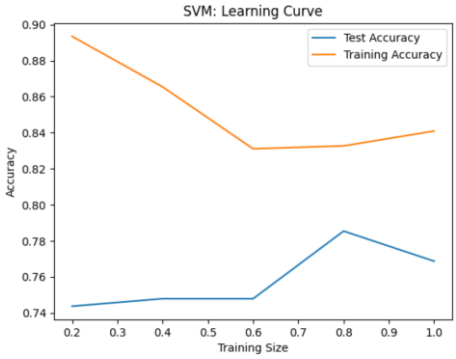
2.6.2. Optimal model

1. Predict

```
Tuned training score:0.841
Tuned test score:0.769
```

2. ROC curve and learning curve





2.7. In conclusion

Algorithms	Training accuracy	Test accuracy	Test AUC	Training time (s)
KNN	0.777	0.754	0.82	0.016
Decision Tree	0.994	0.71	0.72	0.016
Gradient Boosting	1	0.804	0.88	0.86
Neural Network	1	0.767	0.8	8.421
SVM	0.841	0.769	0.82	0.282

Compared to the results of Problem 1, unbalanced classes indeed affect the accuracy. However, scores of Problem 2 are still not good enough. It might because there are only physicochemical features are available in the dataset. We probably need more relevant features like grape types, wine brand, etc.

Reference

P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. Modeling wine preferences by data mining from physicochemical properties.
In Decision Support Systems, Elsevier, 47(4):547-553, 2009.