combined-algos

June 27, 2024

[ ]:
```
import numpy as np # linear algebra
import pandas as pd

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

[ ]:
```
data=pd.read_csv('/content/drive/MyDrive/ML_Team1/1sv21cs001/winequality-red.
↪csv')
```

[ ]:
```
data.head()
```

[ ]:
```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol  quality
0      9.4        5
1      9.8        5
2      9.8        5
3      9.8        6
```

```
4      9.4        5
```

```
[ ]: data.shape
```

```
[ ]: (1599, 12)
```

```
[ ]: feature_list = data.columns[:-1].values
     label = [data.columns[-1]]

     print ("Feature list:", feature_list)
     print ("Label:", label)
```

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

```
[ ]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

```
[ ]: data.describe()
```

```
[ ]:        fixed acidity  volatile acidity  citric acid  residual sugar  \
     count    1599.000000       1599.000000  1599.000000     1599.000000
     mean        8.319637          0.527821     0.270976        2.538806
     std         1.741096          0.179060     0.194801        1.409928
     min         4.600000          0.120000     0.000000        0.900000
     25%         7.100000          0.390000     0.090000        1.900000
```

```
50%         7.900000          0.520000     0.260000        2.200000
75%         9.200000          0.640000     0.420000        2.600000
max        15.900000          1.580000     1.000000       15.500000

         chlorides  free sulfur dioxide  total sulfur dioxide      density  \
count  1599.000000          1599.000000           1599.000000  1599.000000
mean      0.087467            15.874922             46.467792     0.996747
std       0.047065            10.460157             32.895324     0.001887
min       0.012000             1.000000              6.000000     0.990070
25%       0.070000             7.000000             22.000000     0.995600
50%       0.079000            14.000000             38.000000     0.996750
75%       0.090000            21.000000             62.000000     0.997835
max       0.611000            72.000000            289.000000     1.003690

                pH    sulphates      alcohol      quality
count  1599.000000  1599.000000  1599.000000  1599.000000
mean      3.311113     0.658149    10.422983     5.636023
std       0.154386     0.169507     1.065668     0.807569
min       2.740000     0.330000     8.400000     3.000000
25%       3.210000     0.550000     9.500000     5.000000
50%       3.310000     0.620000    10.200000     6.000000
75%       3.400000     0.730000    11.100000     6.000000
max       4.010000     2.000000    14.900000     8.000000
```

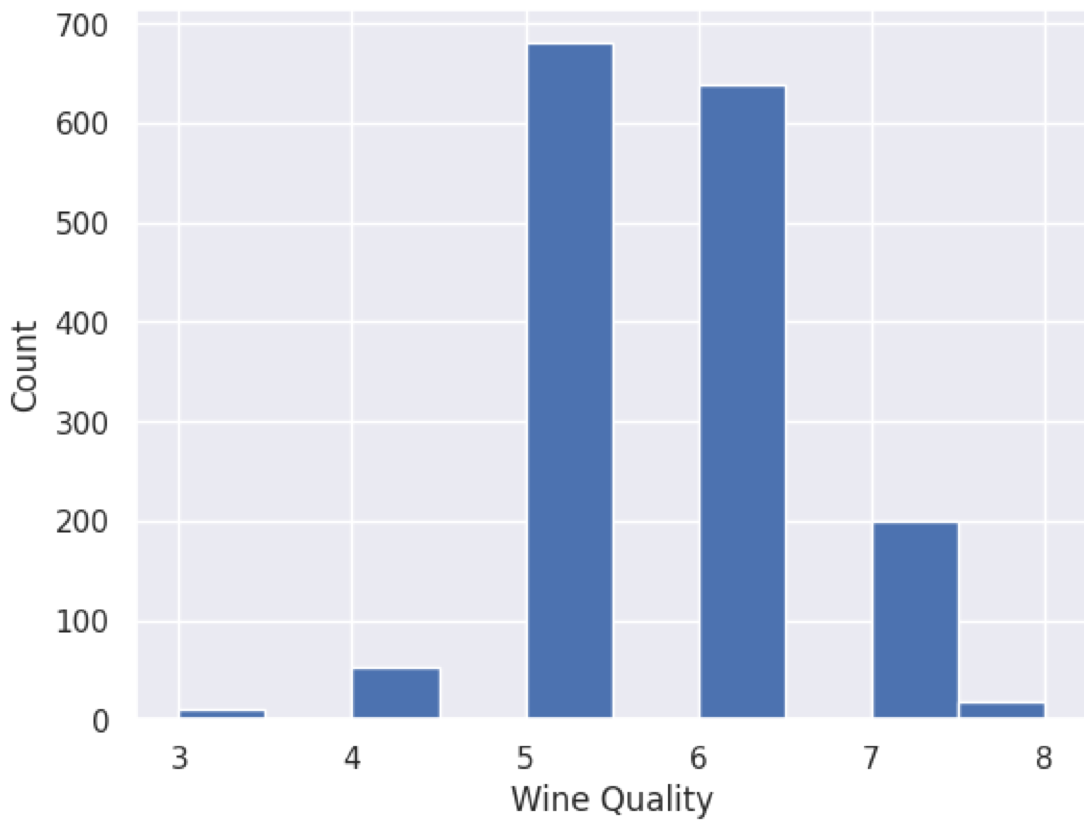
[ ]:
```python
data['quality'].value_counts()
```

[ ]:
```
quality
5    681
6    638
7    199
4     53
8     18
3     10
Name: count, dtype: int64
```

[ ]:
```python
sns.set()
```

[ ]:
```python
data.quality.hist()
plt.xlabel('Wine Quality')
plt.ylabel('Count')
```

[ ]:
```
Text(0, 0.5, 'Count')
```

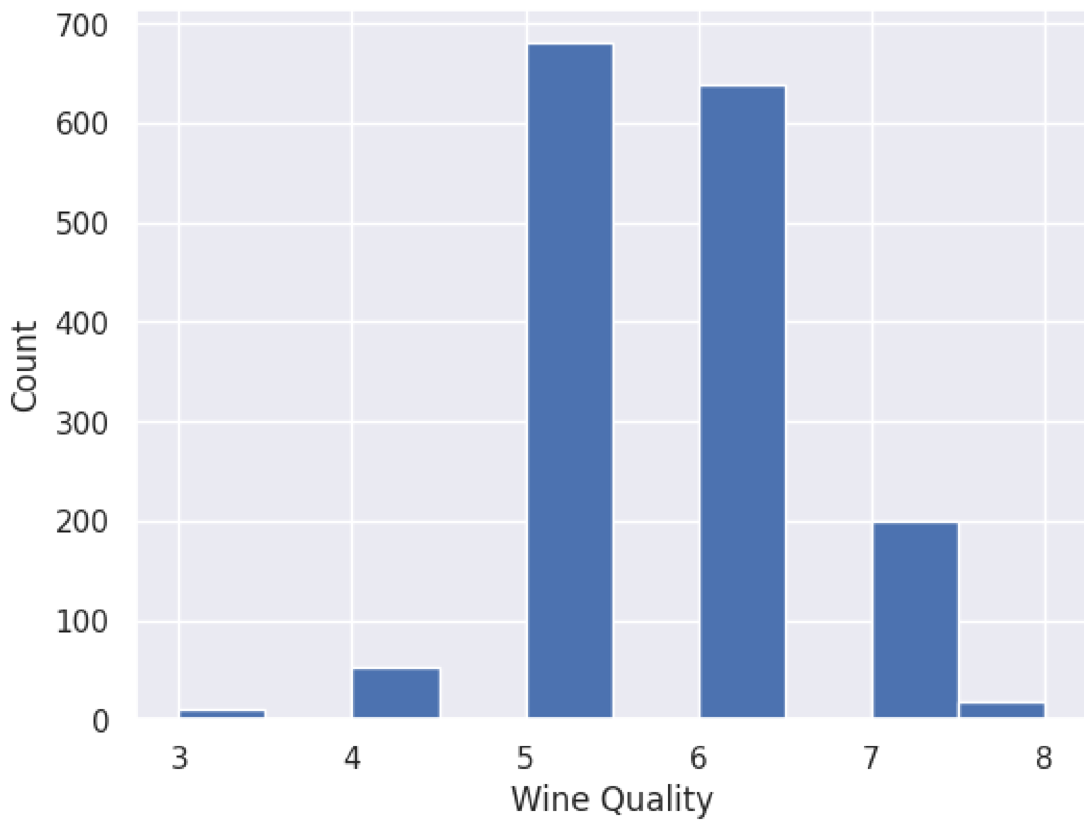



```
[ ]: from sklearn.model_selection import train_test_split
```

```
[ ]: train_set, test_set = train_test_split(data, test_size=0.2, random_state=42)
```

```
[ ]: data.quality.hist()
     plt.xlabel('Wine Quality')
     plt.ylabel('Count')
```

```
[ ]: Text(0, 0.5, 'Count')
```



[ ]:
```
from sklearn.model_selection import StratifiedShuffleSplit
sss = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in sss.split(data, data["quality"]):
  strat_train_set = data.loc[train_index]
  strat_test_set = data.loc[test_index]
```

[ ]:
```
train_index, test_index= next(sss.split(data, data["quality"]))
strat_train_set = data.loc[train_index]
strat_test_set = data.loc[test_index]
```

[ ]:
```
strat_train_set
```

[ ]:
```
      fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
1542            6.7             0.855         0.02            1.90      0.064
1558            6.9             0.630         0.33            6.70      0.235
344            11.9             0.570         0.50            2.60      0.082
924             8.6             0.470         0.27            2.30      0.055
971            10.4             0.260         0.48            1.90      0.066
…               …                 …            …               …         …
1056            8.9             0.480         0.53            4.00      0.101
```

```
1394           6.4          0.570       0.14           3.90       0.070
337            7.8          0.430       0.32           2.80       0.080
539           11.2          0.500       0.74           5.15       0.100
1083           8.7          0.420       0.45           2.40       0.072

      free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
1542                 29.0                  38.0  0.99472  3.30       0.56
1558                 66.0                 115.0  0.99787  3.22       0.56
344                   6.0                  32.0  1.00060  3.12       0.78
924                  14.0                  28.0  0.99516  3.18       0.80
971                   6.0                  10.0  0.99724  3.33       0.87
…                    …                    …      …        …      …
1056                  3.0                  10.0  0.99586  3.21       0.59
1394                 27.0                  73.0  0.99669  3.32       0.48
337                  29.0                  58.0  0.99740  3.31       0.64
539                   5.0                  17.0  0.99960  3.22       0.62
1083                 32.0                  59.0  0.99617  3.33       0.77

      alcohol  quality
1542    10.75        6
1558     9.50        5
344     10.70        6
924     11.20        5
971     10.90        6
…       …      …
1056    12.10        7
1394     9.20        5
337     10.30        5
539     11.20        5
1083    12.00        6

[1279 rows x 12 columns]
```

[ ]:
```python
random_dist = test_set["quality"].value_counts() / len(test_set)
random_dist
```

[ ]:
```
quality
6    0.412500
5    0.406250
7    0.131250
4    0.031250
8    0.015625
3    0.003125
Name: count, dtype: float64
```

[ ]:
```python
wine_features = strat_train_set.drop("quality", axis=1)
```

```
wine_labels = strat_train_set['quality'].copy()
```

[ ]:
```
wine_features.isna().sum()
```

[ ]:
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
dtype: int64
```

[ ]:
```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="median")
```

[ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsClassifier  # Import the classifier

# Assuming 'data' is your DataFrame and 'quality' is your target variable
wine_features = data.drop('quality', axis=1)
wine_labels = data['quality']

train_set, test_set = train_test_split(data, test_size=0.2, random_state=42)

wine_features_tr = train_set.drop('quality', axis=1)
wine_labels_tr = train_set['quality']

# Linear Regression (unchanged)
lin_reg = LinearRegression()
lin_reg.fit(wine_features_tr, wine_labels_tr)

# K-Nearest Neighbors Classifier
knn_classifier = KNeighborsClassifier(n_neighbors=5)  # Use the classifier
knn_classifier.fit(wine_features_tr, wine_labels_tr)

# Now you have a linear regression model and a k-NN classifier trained.
```

[ ]:
```
KNeighborsClassifier()
```

[ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier  # Import Decision Tree

# Assuming 'data' is your DataFrame and 'quality' is your target variable
wine_features = data.drop('quality', axis=1)
wine_labels = data['quality']

train_set, test_set = train_test_split(data, test_size=0.2, random_state=42)

wine_features_tr = train_set.drop('quality', axis=1)
wine_labels_tr = train_set['quality']

# Linear Regression (unchanged)
lin_reg = LinearRegression()
lin_reg.fit(wine_features_tr, wine_labels_tr)

# K-Nearest Neighbors Classifier (unchanged)
knn_classifier = KNeighborsClassifier(n_neighbors=5)
knn_classifier.fit(wine_features_tr, wine_labels_tr)

# Decision Tree Classifier
tree_classifier = DecisionTreeClassifier()  # Create the classifier
tree_classifier.fit(wine_features_tr, wine_labels_tr)  # Train the classifier

# Now you have linear regression, k-NN, and decision tree models trained.
```

[ ]: DecisionTreeClassifier()

[ ]:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier  # Import Random Forest

# Assuming 'data' is your DataFrame and 'quality' is your target variable
wine_features = data.drop('quality', axis=1)
wine_labels = data['quality']

train_set, test_set = train_test_split(data, test_size=0.2, random_state=42)

wine_features_tr = train_set.drop('quality', axis=1)
wine_labels_tr = train_set['quality']

# Linear Regression (unchanged)
lin_reg = LinearRegression()
lin_reg.fit(wine_features_tr, wine_labels_tr)
```

```
# K-Nearest Neighbors Classifier (unchanged)
knn_classifier = KNeighborsClassifier(n_neighbors=5)
knn_classifier.fit(wine_features_tr, wine_labels_tr)

# Decision Tree Classifier (unchanged)
tree_classifier = DecisionTreeClassifier()
tree_classifier.fit(wine_features_tr, wine_labels_tr)

# Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100)  # Create the␣
↪classifier
rf_classifier.fit(wine_features_tr, wine_labels_tr)  # Train the classifier

# Now you have linear regression, k-NN, decision tree, and random forest models.
```

[ ]: RandomForestClassifier()

[ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression  # Import␣
↪Logistic Regression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

# Assuming 'data' is your DataFrame and 'quality' is your target variable
wine_features = data.drop('quality', axis=1)
wine_labels = data['quality']

train_set, test_set = train_test_split(data, test_size=0.2, random_state=42)

wine_features_tr = train_set.drop('quality', axis=1)
wine_labels_tr = train_set['quality']

# Linear Regression (unchanged)
lin_reg = LinearRegression()
lin_reg.fit(wine_features_tr, wine_labels_tr)

# K-Nearest Neighbors Classifier (unchanged)
knn_classifier = KNeighborsClassifier(n_neighbors=5)
knn_classifier.fit(wine_features_tr, wine_labels_tr)

# Decision Tree Classifier (unchanged)
tree_classifier = DecisionTreeClassifier()
tree_classifier.fit(wine_features_tr, wine_labels_tr)

# Random Forest Classifier (unchanged)
rf_classifier = RandomForestClassifier(n_estimators=100)
```

```
rf_classifier.fit(wine_features_tr, wine_labels_tr)

# Logistic Regression Classifier
log_reg = LogisticRegression(max_iter=1000)  # Create the classifier
log_reg.fit(wine_features_tr, wine_labels_tr)  # Train the classifier

# Now you have linear regression, k-NN, decision tree, random forest, and␣
↪logistic regression.
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[ ]: LogisticRegression(max_iter=1000)

[ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import mean_squared_error, accuracy_score

# Assuming 'data' is your DataFrame and 'quality' is your target variable
wine_features = data.drop('quality', axis=1)
wine_labels = data['quality']

train_set, test_set = train_test_split(data, test_size=0.2, random_state=42)

wine_features_tr = train_set.drop('quality', axis=1)
wine_labels_tr = train_set['quality']
wine_features_te = test_set.drop('quality', axis=1)
wine_labels_te = test_set['quality']

# Dictionary to store models
models = {}

# Linear Regression
models['Linear Regression'] = LinearRegression()

# Logistic Regression
```

```python
models['Logistic Regression'] = LogisticRegression(max_iter=1000)

# K-Nearest Neighbors Classifier
models['K-Nearest Neighbors'] = KNeighborsClassifier(n_neighbors=5)

# Decision Tree Classifier
models['Decision Tree'] = DecisionTreeClassifier()

# Random Forest Classifier
models['Random Forest'] = RandomForestClassifier(n_estimators=100)

# Train and evaluate models
for name, model in models.items():
    model.fit(wine_features_tr, wine_labels_tr)

    if name == 'Linear Regression':
        predictions = model.predict(wine_features_te)
        mse = mean_squared_error(wine_labels_te, predictions)
        print(f"{name} - Mean Squared Error: {mse}")
    else:
        predictions = model.predict(wine_features_te)
        accuracy = accuracy_score(wine_labels_te, predictions)
        print(f"{name} - Accuracy: {accuracy}")
```

```
Linear Regression - Mean Squared Error: 0.390025143963954l6
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
Logistic Regression - Accuracy: 0.575
K-Nearest Neighbors - Accuracy: 0.45625
Decision Tree - Accuracy: 0.565625
Random Forest - Accuracy: 0.684375
```

[ ]:
```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import mean_squared_error, accuracy_score
```

```python
# Assuming 'data' is your DataFrame and 'quality' is your target variable
wine_features = data.drop('quality', axis=1)
wine_labels = data['quality']

train_set, test_set = train_test_split(data, test_size=0.2, random_state=42)

wine_features_tr = train_set.drop('quality', axis=1)
wine_labels_tr = train_set['quality']
wine_features_te = test_set.drop('quality', axis=1)
wine_labels_te = test_set['quality']

# Dictionary to store models
models = {}

# Linear Regression
models['Linear Regression'] = LinearRegression()

# Logistic Regression
models['Logistic Regression'] = LogisticRegression(max_iter=1000)

# K-Nearest Neighbors Classifier
models['K-Nearest Neighbors'] = KNeighborsClassifier(n_neighbors=5)

# Decision Tree Classifier
models['Decision Tree'] = DecisionTreeClassifier()

# Random Forest Classifier
models['Random Forest'] = RandomForestClassifier(n_estimators=100)

# Train and evaluate models
for name, model in models.items():
    model.fit(wine_features_tr, wine_labels_tr)

    if name == 'Linear Regression':
        predictions = model.predict(wine_features_te)
        mse = mean_squared_error(wine_labels_te, predictions)
        print(f"{name} - The average squared difference between predicted and␣
 ↪actual wine quality is {mse:.2f}.")
    else:
        predictions = model.predict(wine_features_te)
        accuracy = accuracy_score(wine_labels_te, predictions)
        print(f"{name} - Correctly predicted the quality of {accuracy * 100:.
 ↪2f}% of wines in the test set.")
```

```
Linear Regression - The average squared difference between predicted and actual
wine quality is 0.39.
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
Logistic Regression - Correctly predicted the quality of 57.50% of wines in the
test set.
K-Nearest Neighbors - Correctly predicted the quality of 45.62% of wines in the
test set.
Decision Tree - Correctly predicted the quality of 56.25% of wines in the test
set.
Random Forest - Correctly predicted the quality of 65.00% of wines in the test
set.
```