randomforest-hyperparameter-tuning

November 29, 2023

Day 54: Random Forest Hyperparameter Tuning By: Loga Aswin

[1]:
```
# import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
```

[29]:
```
# load datasets
df= pd.read_csv('/content/winequality-red (1).csv')
```

Exploratory Data Analysis(EDA):

[30]:
```
df.head()
```

[30]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

```
4      9.4        5
```

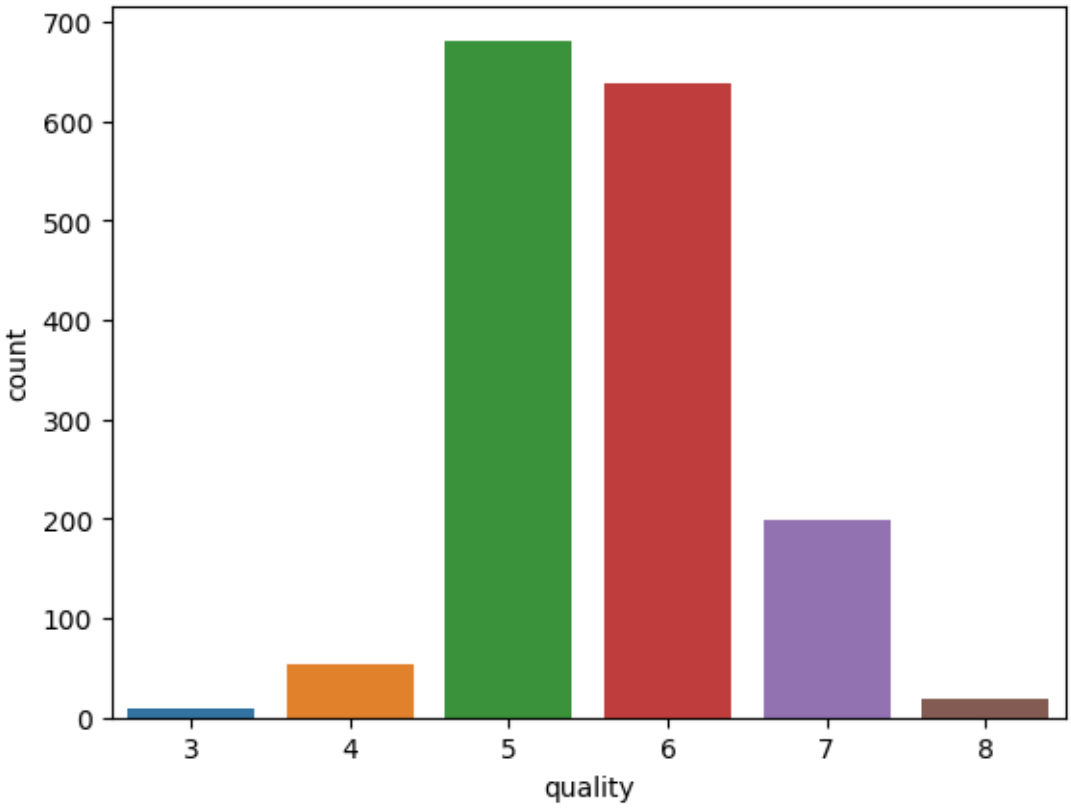
[3]:
```
# checking missing values
df.isnull().sum()
```

[3]:
```
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
quality                 0
dtype: int64
```

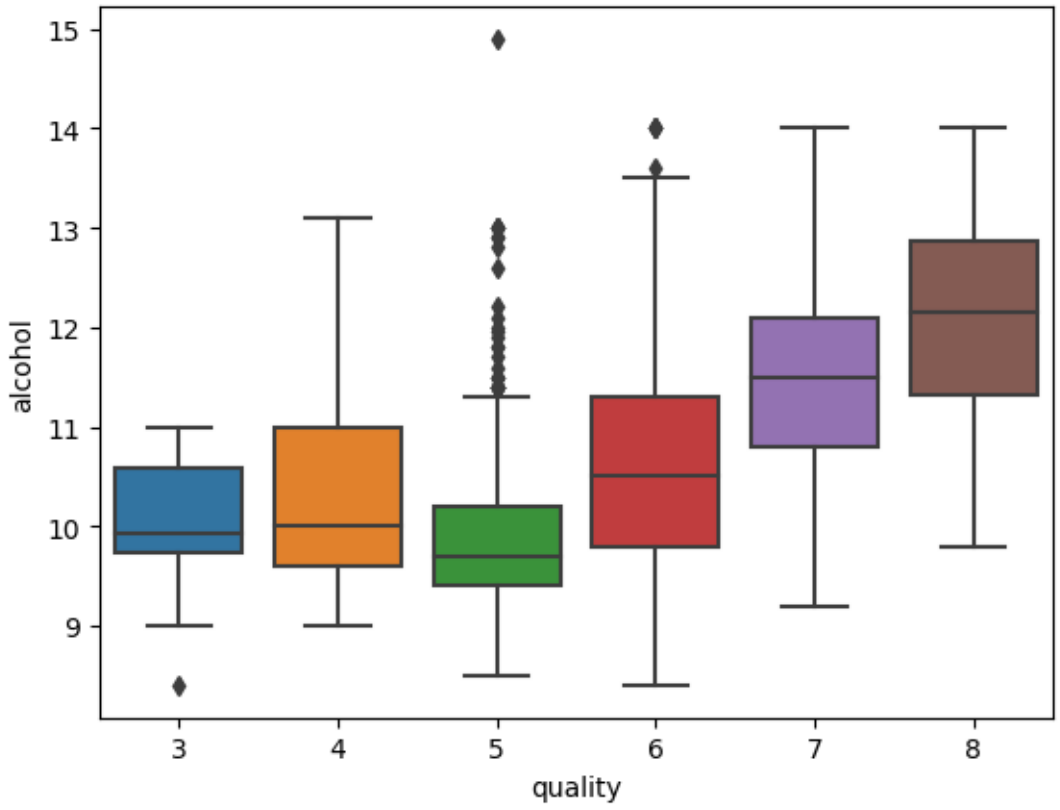
[26]:
```
df['quality'].value_counts
```

[26]:
```
<bound method IndexOpsMixin.value_counts of 0       5
1       5
2       5
3       6
4       5
       ..
1594    5
1595    6
1596    6
1597    5
1598    6
Name: quality, Length: 1599, dtype: int64>
```

[4]:
```
sns.countplot(x='quality', data=df)
plt.show()
```



[5]:
```
sns.boxplot(x='quality', y='alcohol', data=df)
plt.show()
```



[6]:
```
# target variable
X = df.drop('quality', axis=1)
y = df['quality']
```

[7]:
```
df.head()
```

[7]:

	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
```

Splitting into train and test split:

[8]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)
```

[9]:
```
X_train.shape, X_test.shape
```

[9]:
```
((1279, 11), (320, 11))
```

Using RandomForestClassifier Model:

[10]:
```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=100)
model.fit(X_train,y_train)
```

[10]:
```
RandomForestClassifier()
```

Predict Test Results:

[11]:
```
y_pred = model.predict(X_test)
```

Model Evaluation Metrics:

[12]:
```
from sklearn import metrics

print('Accuracy: ', metrics.accuracy_score(y_test,y_pred))
```

```
Accuracy:  0.678125
```

[ ]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

# Defining evaluate function
def evaluate(model, X_test, y_test):
    predictions = model.predict(X_test)
    errors = abs(predictions - y_test)
    mape = 100 * np.mean(errors / y_test)
    accuracy = 100 - mape
    print('Model Performance')
    print('Average Error: {:0.4f} degrees.'.format(np.mean(errors)))
    print('Accuracy = {:0.2f}%.'.format(accuracy))

    return accuracy
```

[ ]:
```
# Define parameter grid
param_grid = {
    'bootstrap': [True],
    'max_depth': [8, 10, 12, 14],
    'max_features': [2, 3],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300, 1000]
}
```

[25]:
```
# Create a base model
rf = RandomForestRegressor()

# Instantiate the grid search model
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid,
                           cv=3, n_jobs=-1, verbose=2)
```

[21]:
```
# Fit grid search to data
grid_search.fit(X_train, y_train)

# Get best parameters by grid search
best_grid = grid_search.best_estimator_

# Model Evaluate with from grid search
grid_accuracy = evaluate(best_grid, X_test, y_test)
```

```
Fitting 3 folds for each of 288 candidates, totalling 864 fits
Model Performance
Average Error: 0.4630 degrees.
Accuracy = 91.47%.
```