wine-quality-prediction-2

October 29, 2023

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

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn import metrics
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
```

[2]:
```
wine_quality= pd.read_csv("E:\Dataset\WineQT.csv")
```

[3]:
```
wine_quality
```

[3]:
```
      fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0               7.4             0.700         0.00             1.9      0.076
1               7.8             0.880         0.00             2.6      0.098
2               7.8             0.760         0.04             2.3      0.092
3              11.2             0.280         0.56             1.9      0.075
4               7.4             0.700         0.00             1.9      0.076
…                …                 …            …               …          …
1138            6.3             0.510         0.13             2.3      0.076
1139            6.8             0.620         0.08             1.9      0.068
1140            6.2             0.600         0.08             2.0      0.090
1141            5.9             0.550         0.10             2.2      0.062
1142            5.9             0.645         0.12             2.0      0.075

      free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                    11.0                  34.0  0.99780  3.51       0.56
1                    25.0                  67.0  0.99680  3.20       0.68
2                    15.0                  54.0  0.99700  3.26       0.65
3                    17.0                  60.0  0.99800  3.16       0.58
4                    11.0                  34.0  0.99780  3.51       0.56
…                       …                     …        …     …          …
1138                 29.0                  40.0  0.99574  3.42       0.75
```

```
1139                  28.0                  38.0  0.99651  3.42       0.82
1140                  32.0                  44.0  0.99490  3.45       0.58
1141                  39.0                  51.0  0.99512  3.52       0.76
1142                  32.0                  44.0  0.99547  3.57       0.71

      alcohol  quality    Id
0         9.4        5     0
1         9.8        5     1
2         9.8        5     2
3         9.8        6     3
4         9.4        5     4
…         …        …    …
1138     11.0        6  1592
1139      9.5        6  1593
1140     10.5        5  1594
1141     11.2        6  1595
1142     10.2        5  1597

[1143 rows x 13 columns]
```

[4]:
```python
wine_quality.shape
```

[4]: (1143, 13)

[5]:
```python
wine_quality.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1143 entries, 0 to 1142
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1143 non-null   float64
 1   volatile acidity      1143 non-null   float64
 2   citric acid           1143 non-null   float64
 3   residual sugar        1143 non-null   float64
 4   chlorides             1143 non-null   float64
 5   free sulfur dioxide   1143 non-null   float64
 6   total sulfur dioxide  1143 non-null   float64
 7   density               1143 non-null   float64
 8   pH                    1143 non-null   float64
 9   sulphates             1143 non-null   float64
 10  alcohol               1143 non-null   float64
 11  quality               1143 non-null   int64
 12  Id                    1143 non-null   int64
dtypes: float64(11), int64(2)
memory usage: 116.2 KB
```

[6]:
```
wine_quality.isna().sum()
```

[6]:
```
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
Id                      0
dtype: int64
```

[7]:
```
df = pd.DataFrame(wine_quality)
```

[8]:
```
df
```

[8]:
```
      fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0               7.4             0.700         0.00             1.9      0.076
1               7.8             0.880         0.00             2.6      0.098
2               7.8             0.760         0.04             2.3      0.092
3              11.2             0.280         0.56             1.9      0.075
4               7.4             0.700         0.00             1.9      0.076
…             …               …            …              …         …
1138            6.3             0.510         0.13             2.3      0.076
1139            6.8             0.620         0.08             1.9      0.068
1140            6.2             0.600         0.08             2.0      0.090
1141            5.9             0.550         0.10             2.2      0.062
1142            5.9             0.645         0.12             2.0      0.075

      free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                    11.0                  34.0  0.99780  3.51       0.56
1                    25.0                  67.0  0.99680  3.20       0.68
2                    15.0                  54.0  0.99700  3.26       0.65
3                    17.0                  60.0  0.99800  3.16       0.58
4                    11.0                  34.0  0.99780  3.51       0.56
…                   …                   …      …     …    …
1138                 29.0                  40.0  0.99574  3.42       0.75
1139                 28.0                  38.0  0.99651  3.42       0.82
1140                 32.0                  44.0  0.99490  3.45       0.58
1141                 39.0                  51.0  0.99512  3.52       0.76
1142                 32.0                  44.0  0.99547  3.57       0.71
```

```
      alcohol  quality    Id
0         9.4        5     0
1         9.8        5     1
2         9.8        5     2
3         9.8        6     3
4         9.4        5     4
…         …        …    …
1138     11.0        6  1592
1139      9.5        6  1593
1140     10.5        5  1594
1141     11.2        6  1595
1142     10.2        5  1597

[1143 rows x 13 columns]
```

[9]:

```
# calculate missing value count

for col in df.columns:
    if df[col].isnull().sum() > 0:
        df[col] = df[col].fillna(df[col].mean())

missing_values_count = df.isnull().sum().sum()

for col in df.columns:
    if df[col].isnull().sum() > 0:
        df[col] = df[col].fillna(df[col].mean())

missing_values_count = df.isnull().sum().sum()
```

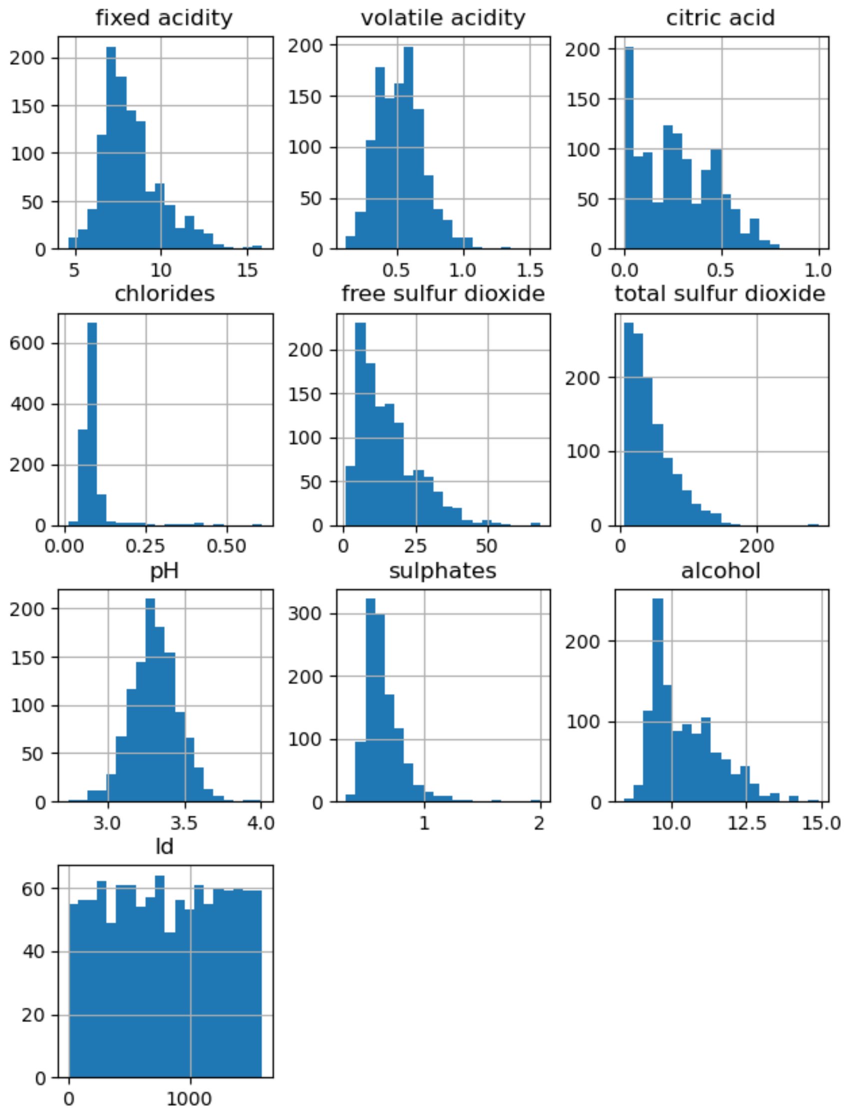
[10]:

```
# print missing value count

print(missing_values_count)
```

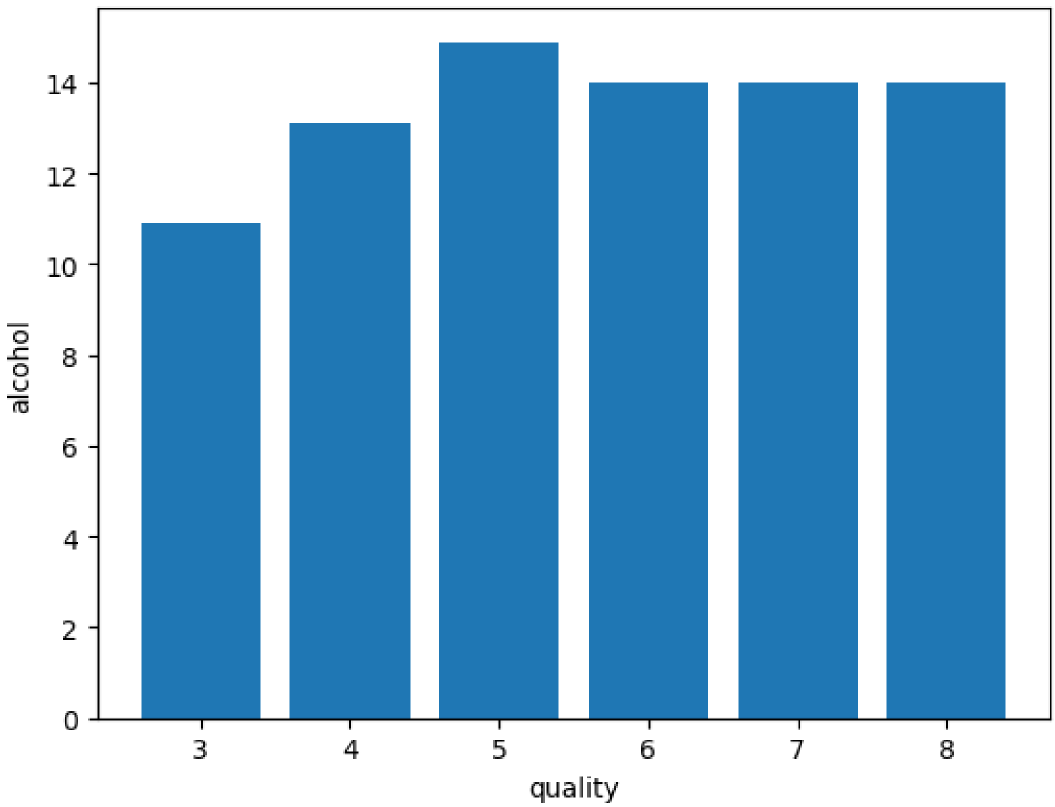
```
0
```

[11]:

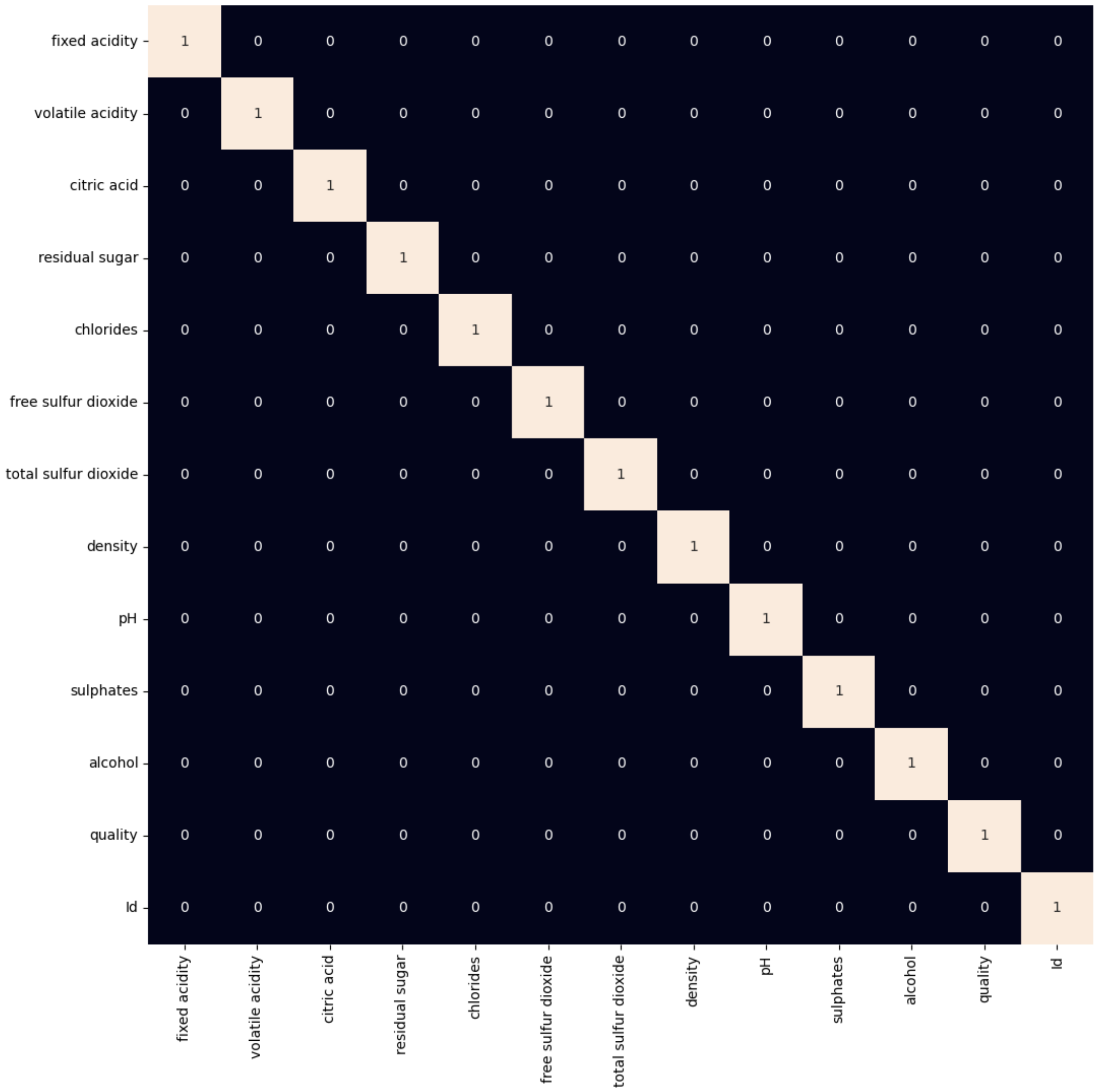
```
# draw the histogram to visualise the distribution of the data with continuous␣
 ↪values in the columns

df.hist(bins=20, figsize=(10, 10))
plt.show()
```



[12]:
```
# draw the count plot to visualise the number data for each quality of wine.

plt.bar(df['quality'], df['alcohol'])
plt.xlabel('quality')
plt.ylabel('alcohol')
plt.show()
```



[13]:
```
# the data provided to us contains redundant features they do not help with␣
↪increasing the model's performance that
# is why we remove them before using them to train our model.

plt.figure(figsize=(12, 12))
sb.heatmap(df.corr() > 0.7, annot=True, cbar=False)
plt.show()
```



[14]:
```
#  From the above heat map we can conclude that the 'total sulphur dioxide' and␣
↪'free sulphur dioxide
# 'are highly correlated features so, we will remove them.

df = df.drop('total sulfur dioxide', axis=1)
```

[15]:
```
df
```

[15]:
```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4             0.700         0.00             1.9      0.076
1            7.8             0.880         0.00             2.6      0.098
2            7.8             0.760         0.04             2.3      0.092
3           11.2             0.280         0.56             1.9      0.075
```

```
4              7.4            0.700        0.00              1.9      0.076
…             …              …           …             …       …
1138           6.3            0.510        0.13              2.3      0.076
1139           6.8            0.620        0.08              1.9      0.068
1140           6.2            0.600        0.08              2.0      0.090
1141           5.9            0.550        0.10              2.2      0.062
1142           5.9            0.645        0.12              2.0      0.075

      free sulfur dioxide  density    pH  sulphates  alcohol  quality    Id
0                    11.0  0.99780  3.51       0.56      9.4        5     0
1                    25.0  0.99680  3.20       0.68      9.8        5     1
2                    15.0  0.99700  3.26       0.65      9.8        5     2
3                    17.0  0.99800  3.16       0.58      9.8        6     3
4                    11.0  0.99780  3.51       0.56      9.4        5     4
…                   …      …    …          …        …      …   …
1138                 29.0  0.99574  3.42       0.75     11.0        6  1592
1139                 28.0  0.99651  3.42       0.82      9.5        6  1593
1140                 32.0  0.99490  3.45       0.58     10.5        5  1594
1141                 39.0  0.99512  3.52       0.76     11.2        6  1595
1142                 32.0  0.99547  3.57       0.71     10.2        5  1597

[1143 rows x 12 columns]
```

[16]:
```
df['best quality'] = [1 if x > 5 else 0 for x in df.quality]
```

[17]:
```
# We have a column with object data type as well let's replace it with the 0␣
↪and 1 as there are only two categories.

df.replace({'white': 1, 'red': 0}, inplace=True)
```

[18]:
```
# After segregating features and the target variable from the dataset we will␣
↪split it into 80:20 ratio for model selection.

features = df.drop(['quality', 'best quality'], axis=1)
target = df['best quality']

xtrain, xtest, ytrain, ytest = train_test_split(
    features, target, test_size=0.2, random_state=40)

xtrain.shape, xtest.shape
```

[18]: ((914, 11), (229, 11))

[19]:
```
# Normalising the data before training help us to achieve stable and fast␣
↪training of the model.

norm = MinMaxScaler()
```

```
xtrain = norm.fit_transform(xtrain)
xtest = norm.transform(xtest)
```

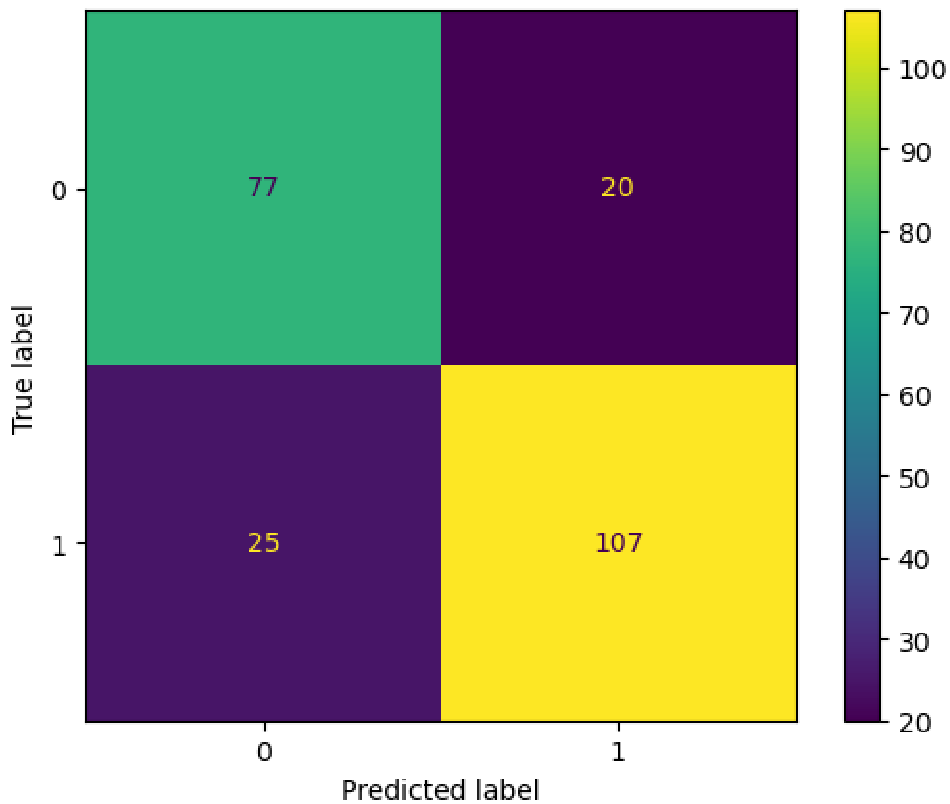
[20]:
```
# As the data has been prepared completely let's train some state of the art␣
↪machine learning model on it.

models = [LogisticRegression(), XGBClassifier(), SVC(kernel='rbf')]

for i in range(3):
    models[i].fit(xtrain, ytrain)

    print(f'{models[i]} : ')
    print('Training Accuracy : ', metrics.roc_auc_score(ytrain, models[i].
↪predict(xtrain)))
    print('Validation Accuracy : ', metrics.roc_auc_score(
        ytest, models[i].predict(xtest)))
    print()
```

```
LogisticRegression() :
Training Accuracy :  0.7546950559364851
Validation Accuracy :  0.7255154639175256

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, …) :
Training Accuracy :  1.0
Validation Accuracy :  0.8022102467978757

SVC() :
Training Accuracy :  0.7648213641284736
Validation Accuracy :  0.7358247422680412
```

[21]:
```
# From the above accuracies we can say that Logistic Regression and SVC()␣
↪classifier performing better on the validation
# data with less difference between the validation and training data. Let's␣
↪plot the confusion matrix as well for the
# validation data using the Logistic Regression model.
```

```
metrics.plot_confusion_matrix(models[1], xtest, ytest)
plt.show()
```

```
C:\Users\Pratik123\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87:
FutureWarning: Function plot_confusion_matrix is deprecated; Function
`plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one
of the class methods: ConfusionMatrixDisplay.from_predictions or
ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```



[22]:
```
# Let's also print the classification report for the best performing model.

print(metrics.classification_report(ytest,
                                    models[1].predict(xtest)))
```

```
              precision    recall  f1-score   support

           0       0.75      0.79      0.77        97
           1       0.84      0.81      0.83       132

    accuracy                           0.80       229
```

```
   macro avg       0.80      0.80      0.80       229
weighted avg       0.81      0.80      0.80       229
```

```
[ ]:
```