wine_testing

October 18, 2022

1 CAPSTONE PROJECT

1.0.1 Summary:

For the wine tasting problem, I have dealth with multicolinearity through PCA and have trained Logistic regression, Random-forest and SVM. I have also run hyperparameter tuning with k-fold cross validation.

[1235]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings

warnings.filterwarnings('ignore')
```

[1236]:
```
#reading the data

df_entire = pd.read_csv('QualityPrediction.csv')
print(df_entire.shape)
df_entire.head(n=10)
```

```
(1599, 12)
```

[1236]:
```
   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
5            7.4              0.66         0.00             1.8      0.075
6            7.9              0.60         0.06             1.6      0.069
7            7.3              0.65         0.00             1.2      0.065
8            7.8              0.58         0.02             2.0      0.073
9            7.5              0.50         0.36             6.1      0.071

   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
5                13.0                   40.0   0.9978  3.51       0.56
6                15.0                   59.0   0.9964  3.30       0.46
7                15.0                   21.0   0.9946  3.39       0.47
8                 9.0                   18.0   0.9968  3.36       0.57
9                17.0                  102.0   0.9978  3.35       0.80

   alcohol  quality
0      9.4        5
1      9.8        5
2      9.8        5
3      9.8        6
4      9.4        5
5      9.4        5
6      9.4        5
7     10.0        7
8      9.5        7
9     10.5        5
```

1.1 1. Preprocessing & Exploratory data analysis

1. Treatment of missing values
2. Data format Handling
3. Data visualization
4. Treatment of outliers

1.1.1 1.1 Treatment of missing values

[1237]:
```
#percentage of missing values in each column

(df_entire.isnull().sum()/df_entire.isnull().count()).
↪sort_values(ascending=False)
```

[1237]:
```
quality                 0.0
alcohol                 0.0
sulphates               0.0
pH                      0.0
density                 0.0
total sulfur dioxide    0.0
free sulfur dioxide     0.0
chlorides               0.0
residual sugar          0.0
citric acid             0.0
volatile acidity        0.0
fixed acidity           0.0
```

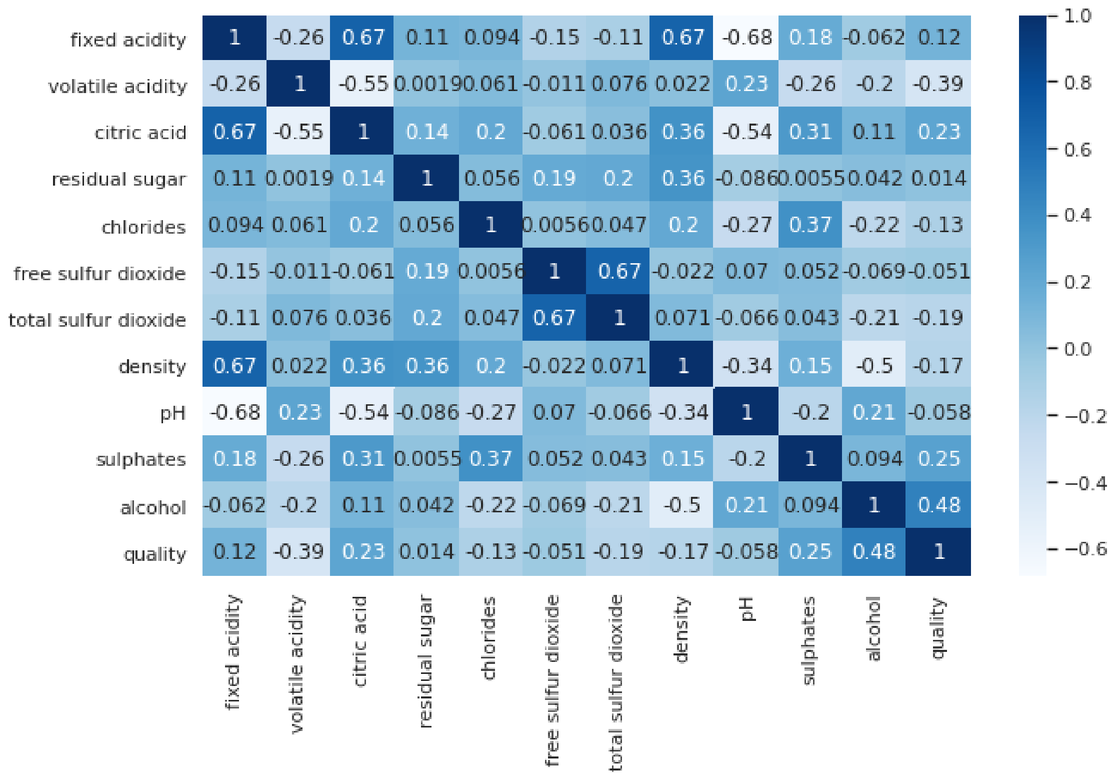
```
dtype: float64
```

Inference: There are no missing values in the data

1.1.2 1.2 Correlation coefficient measure

```
[1238]: corr= df_entire.corr()
        plt.figure(figsize=(10,6))
        sns.heatmap(corr, cmap='Blues', annot=True)
```

```
[1238]: <AxesSubplot:>
```



1.1.3 From the Correlation heatmap it is apparent that features have low-to-medium correlation. We will perform PCA little further before random forest classifier to handle multi-colinearity.

1.1.4 1.3 Data format handling

```
[1298]: df_entire.dtypes
```

```
[1298]: fixed acidity           float64
        volatile acidity        float64
```

```
citric acid             float64
residual sugar          float64
chlorides               float64
free sulfur dioxide     float64
total sulfur dioxide    float64
density                 float64
pH                      float64
sulphates               float64
alcohol                 float64
quality                   int64
dtype: object
```

No Data format handling required.

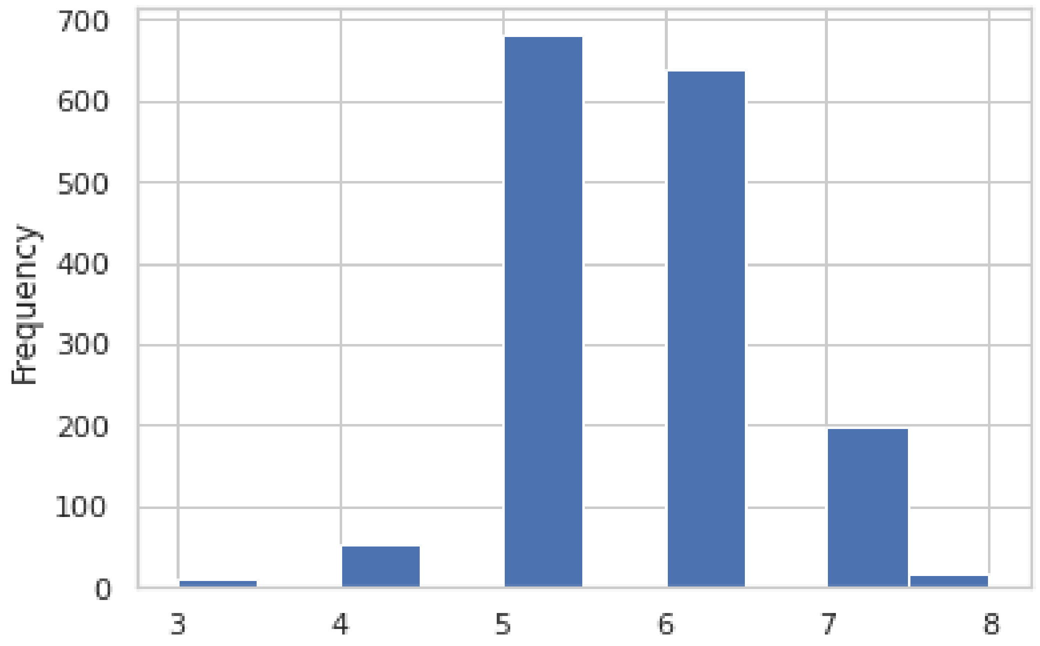
1.1.5 1.4 Data Visualization

```
[1299]: df_entire.columns
```

```
[1299]: Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
               'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
               'pH', 'sulphates', 'alcohol', 'quality'],
              dtype='object')
```

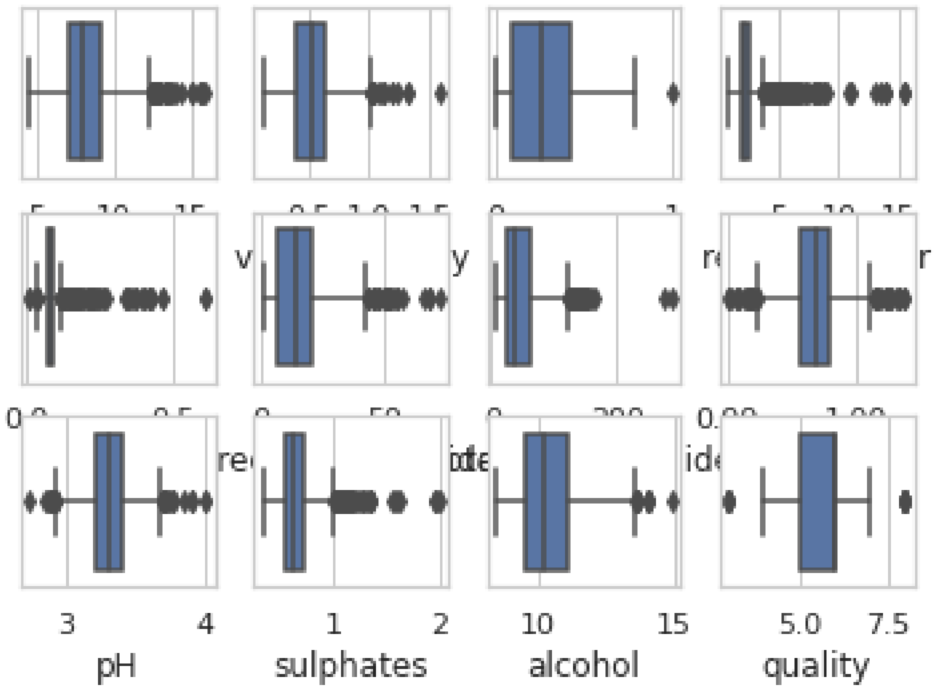
```
[1300]: df_entire['quality'].plot.hist()
```

```
[1300]: <AxesSubplot:ylabel='Frequency'>
```



```
[1301]: #sns.boxplot(x="variable", y="value", data=pd.melt(df_entire))
        #warnings.filterwarnings('ignore')

        f, axes = plt.subplots(3,4)
        sns.boxplot(df_entire['fixed acidity'], ax=axes[0,0])
        sns.boxplot(df_entire['volatile acidity'], ax=axes[0,1])
        sns.boxplot(df_entire['citric acid'], ax=axes[0,2])
        sns.boxplot(df_entire['residual sugar'], ax=axes[0,3])
        sns.boxplot(df_entire['chlorides'], ax=axes[1,0])
        sns.boxplot(df_entire['free sulfur dioxide'], ax=axes[1,1])
        sns.boxplot(df_entire['total sulfur dioxide'], ax=axes[1,2])
        sns.boxplot(df_entire['density'], ax=axes[1,3])
        sns.boxplot(df_entire['pH'], ax=axes[2,0])
        sns.boxplot(df_entire['sulphates'], ax=axes[2,1])
        sns.boxplot(df_entire['alcohol'], ax=axes[2,2])
        sns.boxplot(df_entire['quality'], ax=axes[2,3])
```

```
[1301]: <AxesSubplot:xlabel='quality'>
```



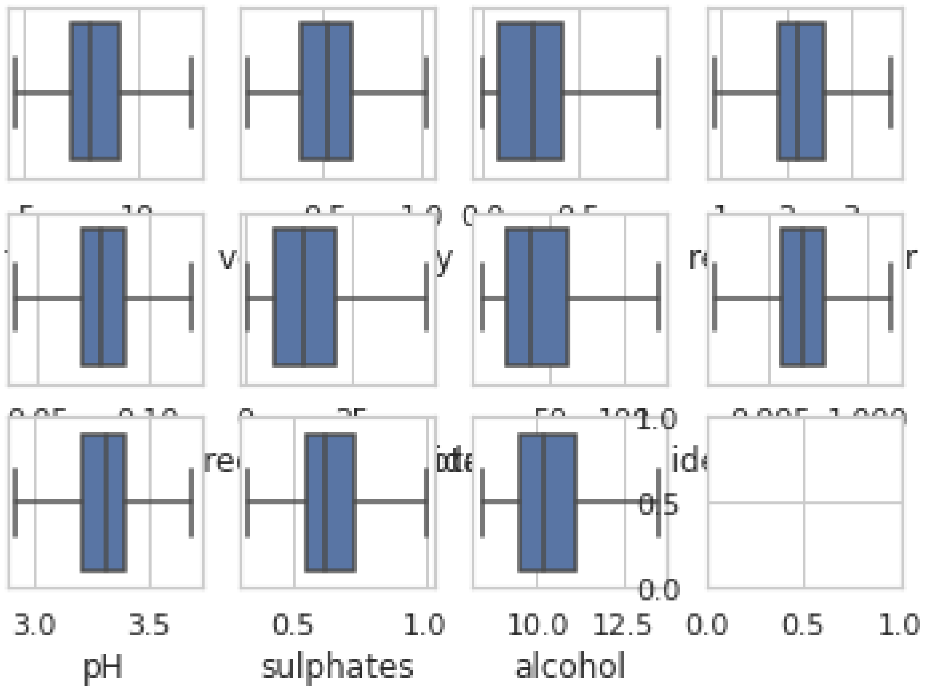
```
[1302]: df_X = df_entire.drop(columns=['quality'])
        df_Y = df_entire[['quality']]
        df_X, df_Y
```

```
[1302]: (      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
         …              …                …           …              …        …
         1594            6.2             0.600         0.08             2.0      0.090
         1595            5.9             0.550         0.10             2.2      0.062
         1596            6.3             0.510         0.13             2.3      0.076
         1597            5.9             0.645         0.12             2.0      0.075
         1598            6.0             0.310         0.47             3.6      0.067

               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
         …                   …                  …      …    …        …
         1594                 32.0                  44.0  0.99490  3.45       0.58
         1595                 39.0                  51.0  0.99512  3.52       0.76
         1596                 29.0                  40.0  0.99574  3.42       0.75
         1597                 32.0                  44.0  0.99547  3.57       0.71
         1598                 18.0                  42.0  0.99549  3.39       0.66

               alcohol
         0         9.4
         1         9.8
         2         9.8
         3         9.8
         4         9.4
         …        …
         1594     10.5
         1595     11.2
         1596     11.0
         1597     10.2
         1598     11.0

         [1599 rows x 11 columns],
               quality
         0           5
         1           5
         2           5
         3           6
         4           5
         …       …
         1594        5
         1595        6
         1596        6
         1597        5
         1598        6

         [1599 rows x 1 columns])
```

[1303]:
```
Q1 = df_X.quantile(0.25)
Q3 = df_X.quantile(0.75)
IQR = Q3 - Q1
lcut = Q1 - 1.5*IQR
rcut = Q3 + 1.5*IQR
L_columns = list(df_X.columns)

for col in L_columns:
    df_X[col].loc[df_X[col] < lcut[col]] = lcut[col] #np.median(df_X[col])
    df_X[col].loc[df_X[col] > rcut[col]] = rcut[col] #np.median(df_X[col])
```

[1304]:
```
f, axes = plt.subplots(3,4)
sns.boxplot(df_X['fixed acidity'], ax=axes[0,0])
sns.boxplot(df_X['volatile acidity'], ax=axes[0,1])
sns.boxplot(df_X['citric acid'], ax=axes[0,2])
sns.boxplot(df_X['residual sugar'], ax=axes[0,3])
sns.boxplot(df_X['chlorides'], ax=axes[1,0])
sns.boxplot(df_X['free sulfur dioxide'], ax=axes[1,1])
sns.boxplot(df_X['total sulfur dioxide'], ax=axes[1,2])
sns.boxplot(df_X['density'], ax=axes[1,3])
sns.boxplot(df_X['pH'], ax=axes[2,0])
sns.boxplot(df_X['sulphates'], ax=axes[2,1])
sns.boxplot(df_X['alcohol'], ax=axes[2,2])
```

[1304]:
```
<AxesSubplot:xlabel='alcohol'>
```



1.2 2. Modeling

1. Train-Test split
2. Scaling
3. Applying Logistic Regression
4. Applying Random-Forest Classifier
5. Applying SVM Classifier

1.2.1 2.1 Train-Test split

```
[1305]: from sklearn.model_selection import train_test_split
```

```
[1306]: df_X_revised = df_X.drop(columns=['pH','residual sugar', 'free sulfur␣
         ↪dioxide']) #, 'fixed acidity', 'citric acid', 'chlorides', 'density', 'total␣
         ↪sulfur dioxide'])
```

```
[1307]: x_train_unscaled, x_test_unscaled, y_train, y_test =␣
         ↪train_test_split(df_X_revised, df_Y, test_size=0.4, random_state=111,␣
         ↪stratify=df_Y)
```

1.3 2.2 Scaling

[1308]:
```
# Robust Scaler
## (x-Q1)/(Q3-Q1)
## Fit for data with outliers

from sklearn.preprocessing import RobustScaler

robust_scaler = RobustScaler()
x_train = robust_scaler.fit_transform(x_train_unscaled)
#df_X = pd.DataFrame(x_train, columns=['fixed acidity', 'volatile acidity',␣
↪'citric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total␣
↪sulfur dioxide', 'density', 'pH', 'sulphates', 'alcohol'])
#df_X.plot.kde()
```

[1309]:
```
x_test = robust_scaler.transform(x_test_unscaled)
```

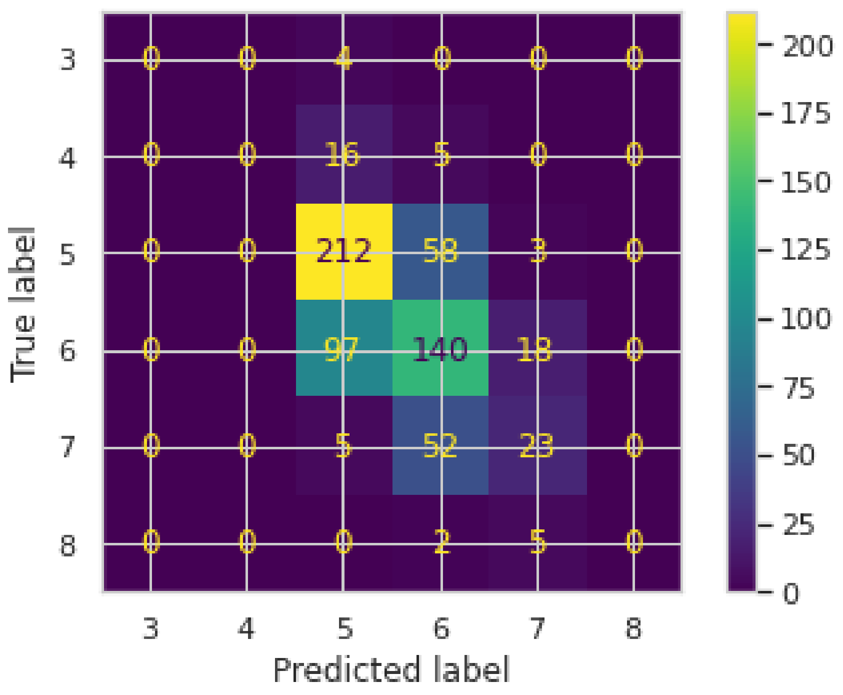
1.3.1 2.2 Applying Logistic Regression

[1310]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score, roc_auc_score,␣
↪plot_confusion_matrix, classification_report
```

[1311]:
```
logReg = LogisticRegression(random_state=111)
logReg.fit(x_train, y_train)
```

[1311]:
```
LogisticRegression(random_state=111)
```

[1312]:
```
#prediction and evaluation
y_pred = logReg.predict(x_test)
plot_confusion_matrix(logReg, x_test, y_test)
plt.show()
```



[1313]:
```
#accuracy score

print("accuracy score for test data", logReg.score(x_test, y_test))
print("accuracy score for training data", logReg.score(x_train, y_train))
```

```
accuracy score for test data 0.5859375
accuracy score for training data 0.5954118873826904
```

[1314]:
```
print(classification_report(y_test, y_pred))
```

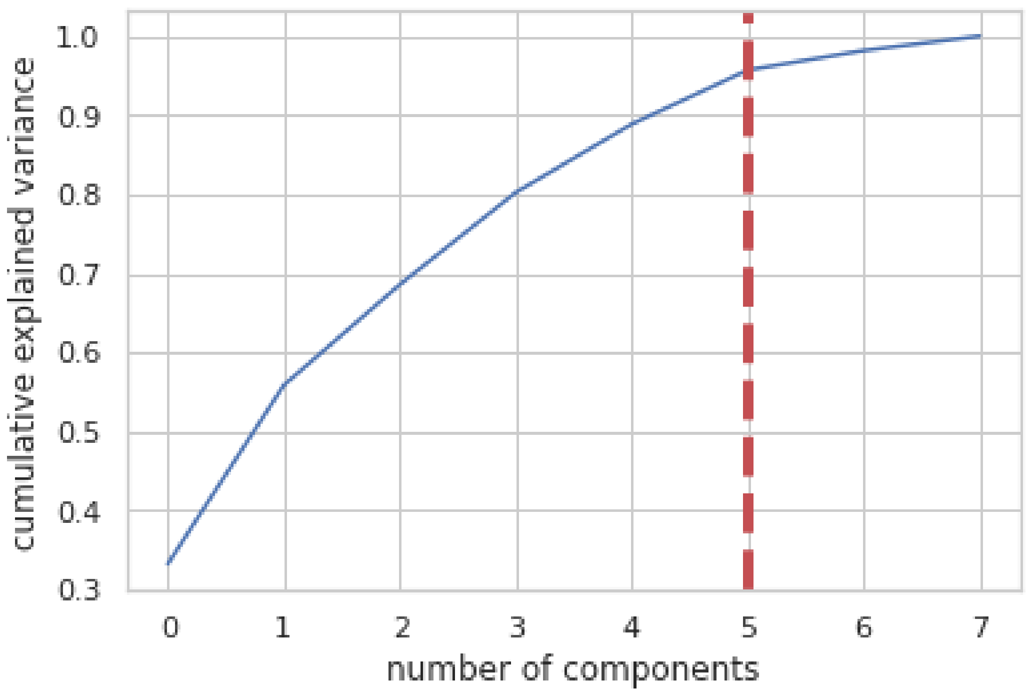
```
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         4
           4       0.00      0.00      0.00        21
           5       0.63      0.78      0.70       273
           6       0.54      0.55      0.55       255
           7       0.47      0.29      0.36        80
           8       0.00      0.00      0.00         7

    accuracy                           0.59       640
   macro avg       0.27      0.27      0.27       640
weighted avg       0.55      0.59      0.56       640
```

Inference: Using Logistic regression that the difference between accuracies of training data and test data is about 5% indicating no overfitting

1.4 Performing PCA to handle multicolinearity.

1.4.1 Principal components are used as features to train Random Forest.

[1315]:
```
#Applying PCA and apt number of principal components for the dataset.

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
pca_test = PCA(n_components=8)
pca_test.fit(x_train)
sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='r', linestyle = '--', x=5, ymin=0, ymax=1)
display(plt.show())
evr = pca_test.explained_variance_ratio_
cvr = np.cumsum(pca_test.explained_variance_ratio_)
pca_df = pd.DataFrame()
pca_df['Cumulative Variance Ratio'] = cvr
pca_df['Explained Variance Ratio'] = evr
display(pca_df.head(10))
```



```
None

   Cumulative Variance Ratio  Explained Variance Ratio
0                   0.331873                  0.331873
1                   0.557983                  0.226110
2                   0.685424                  0.127441
3                   0.802129                  0.116706
4                   0.888526                  0.086397
5                   0.956985                  0.068459
6                   0.981786                  0.024801
7                   1.000000                  0.018214
```

1.4.2 It can be seen from the PCA that with 5 components about 95% of the variance is covered. We will now use n=5 to get principle components for further training with Random forest classifier.

[1316]:
```
pca = PCA(n_components=5)
pca.fit(x_train)

x_train_pca = pca.transform(x_train)
x_test_pca = pca.transform(x_test)
```

1.4.3 2.3 Applying Random Forest Classifier

[1317]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import mean_squared_error
```

[1318]:
```
rf = RandomForestClassifier(random_state=111)
#rf.fit(x_train, y_train)
```

[1319]:
```
#hyperparameter tuning
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV

criterion = ['gini']
# Number of trees in random forest
n_estimators = [100]

# Number of features to consider at every split
max_features = [2,'auto']

# Maximum number of levels in tree
max_depth = range(4,10)
#max_depth.append(None)

# Minimum number of samples required to split a node
min_samples_split = range(2,6)
```

```
# Minimum number of samples required at each leaf node
min_samples_leaf = range(2,5)

# Method of selecting samples for training each tree
bootstrap = [True]

oob_score = [False]

# Create the random grid
random_grid = {'criterion': criterion,
               'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap,
               'oob_score': oob_score}
print(random_grid)
```

```
{'criterion': ['gini'], 'n_estimators': [100], 'max_features': [2, 'auto'],
'max_depth': range(4, 10), 'min_samples_split': range(2, 6), 'min_samples_leaf':
range(2, 5), 'bootstrap': [True], 'oob_score': [False]}
```

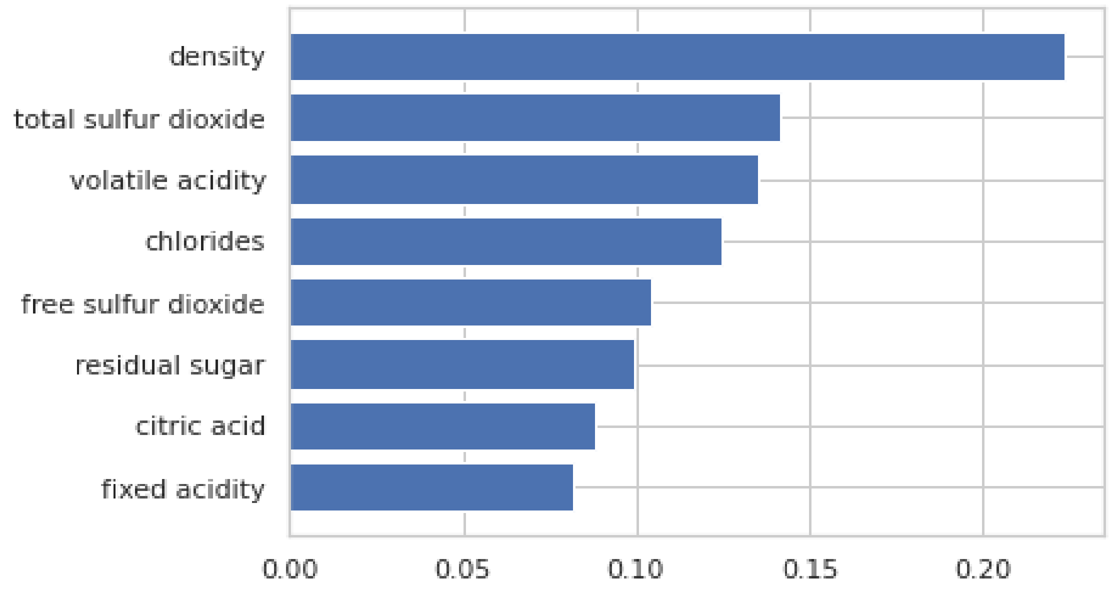
[1320]:
```
#rf_random = RandomizedSearchCV(estimator=rf, param_distributions=random_grid,␣
↪n_iter=100, cv=10, n_jobs=-1)
rf_random = GridSearchCV(estimator=rf, param_grid=random_grid, cv=3, n_jobs=-1,␣
↪verbose=2, scoring='accuracy', return_train_score=True)
rf_random.fit(x_train, y_train)
```

```
Fitting 3 folds for each of 144 candidates, totalling 432 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed:   10.4s
[Parallel(n_jobs=-1)]: Done 154 tasks      | elapsed:   21.1s
[Parallel(n_jobs=-1)]: Done 357 tasks      | elapsed:   41.5s
[Parallel(n_jobs=-1)]: Done 432 out of 432 | elapsed:   49.1s finished
```

[1320]:
```
GridSearchCV(cv=3, estimator=RandomForestClassifier(random_state=111),
             n_jobs=-1,
             param_grid={'bootstrap': [True], 'criterion': ['gini'],
                         'max_depth': range(4, 10), 'max_features': [2, 'auto'],
                         'min_samples_leaf': range(2, 5),
                         'min_samples_split': range(2, 6),
                         'n_estimators': [100], 'oob_score': [False]},
             return_train_score=True, scoring='accuracy', verbose=2)
```

[1321]:
```
sorted_idx = rf_random.best_estimator_.feature_importances_.argsort()
plt.barh(df_X.columns[sorted_idx],rf_random.best_estimator_.
 ↪feature_importances_[sorted_idx])
```

[1321]:
```
<BarContainer object of 8 artists>
```



[1322]:
```
rf_random.best_params_
```

[1322]:
```
{'bootstrap': True,
 'criterion': 'gini',
 'max_depth': 8,
 'max_features': 2,
 'min_samples_leaf': 2,
 'min_samples_split': 2,
 'n_estimators': 100,
 'oob_score': False}
```

[1327]:
```
#best model
rfc_best = RandomForestClassifier(random_state=11, max_depth=8, max_features=2,
                                  min_samples_leaf=2, min_samples_split=2,␣
 ↪bootstrap=True)
rfc_best.fit(x_train_pca, y_train)
print("Best RFC accuracy score on test data: ",rfc_best.score(x_test_pca,␣
 ↪y_test))
print("Best RFC accuracy score on training data: ",rfc_best.score(x_train_pca,␣
 ↪y_train))
print("Best RFC hyperparameters: ", rfc_best.get_params())
```

```
Best RFC accuracy score on test data:  0.603125
Best RFC accuracy score on training data:  0.8300312825860271
Best RFC hyperparameters:  {'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight':
None, 'criterion': 'gini', 'max_depth': 8, 'max_features': 2, 'max_leaf_nodes':
None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_impurity_split':
None, 'min_samples_leaf': 2, 'min_samples_split': 2, 'min_weight_fraction_leaf':
0.0, 'n_estimators': 100, 'n_jobs': None, 'oob_score': False, 'random_state':
11, 'verbose': 0, 'warm_start': False}
```

[1324]:
```
#prediction
#y_pred_rf = rf_random.best_estimator_.predict(x_test)
```

[1325]:
```
#base model
rfc = RandomForestClassifier(random_state=11)
rfc.fit(x_train_pca, y_train)
print("default RFC accuracy score on test data: ",rfc.score(x_test_pca, y_test))
print("default RFC accuracy score on training data: ",rfc.score(x_train_pca,␣
 ↪y_train))
print("default RFC hyperparameters: ", rfc.get_params())
```

```
default RFC accuracy score on test data:  0.6375
default RFC accuracy score on training data:  1.0
default RFC hyperparameters:  {'bootstrap': True, 'ccp_alpha': 0.0,
'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features':
'auto', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_decrease':
0.0, 'min_impurity_split': None, 'min_samples_leaf': 1, 'min_samples_split': 2,
'min_weight_fraction_leaf': 0.0, 'n_estimators': 100, 'n_jobs': None,
'oob_score': False, 'random_state': 11, 'verbose': 0, 'warm_start': False}
```

[1326]:
```
#print(classification_report(y_test, y_pred_rf))
```

Inference: Using RF classifier with PCA and with hyperparam tuning, the best accuracy obtained on test data is 60%. Notably, there is about 23% difference b/w accuracy score of test and training data indicating high variance.

Also note that with baseline Random Forest model, the testing and training accuracies are respectively 63.75% and 100%. This indicates overfitting or high-variance.

1.4.4 2.4 Applying SVM Classifier

[661]:
```
from sklearn.svm import SVC
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline

#pca = PCA(n_components=5, whiten=True, random_state=42)
svc = SVC(kernel='rbf', class_weight='balanced')
#model = make_pipeline(pca, svc)
```

```
model = make_pipeline(svc)
```

[662]:
```
from sklearn.model_selection import GridSearchCV
param_grid = {'svc__C': [1, 5, 10, 50],
              'svc__gamma': [0.0001, 0.0005, 0.001, 0.005, 1.0]}
grid = GridSearchCV(model, param_grid)

%timeit grid.fit(x_train, y_train)
print(grid.best_params_)
```

```
8.37 s ± 345 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
{'svc__C': 10, 'svc__gamma': 1.0}
```

[663]:
```
#prediction
model = grid.best_estimator_
y_pred_svc = model.predict(x_test)
```

[664]:
```
print("accuracy score for test data: ", model.score(x_test, y_test))
print("accuracy score for training data: ", model.score(x_train, y_train))
```

```
accuracy score for test data:  0.6125
accuracy score for training data:  0.9601250977326036
```

[665]:
```
print("classification report: ", classification_report(y_test, y_pred_svc))
```

```
classification report:                precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       0.09      0.09      0.09        11
           5       0.69      0.71      0.70       136
           6       0.60      0.59      0.60       128
           7       0.52      0.55      0.54        40
           8       0.00      0.00      0.00         3

    accuracy                           0.61       320
   macro avg       0.32      0.32      0.32       320
weighted avg       0.60      0.61      0.61       320
```

Inference: Using SVM classifier, the best accuracy obtained on the test data is about 57%. Notably, there is about 33% difference between the accuracy of test and training data indicating higher variance.

[ ]:

[ ]:

[ ]: