MATH 6350 Final Exam Part 1 Fall 2022
Applying Random Forest and Support Vector Machine to Dry Beans Dataset
Jake Zhu

Data Description

The set of data, Dry Bean Dataset, is obtained from University of California Irvine's machine learning repository and may be found in the following link:

https://archive.ics.uci.edu/ml/datasets/Dry+Bean+Dataset

There are a total of 13,511 cases of which we can classify into 7 different classes (types of beans).

The input variables (based on computer vision):

- 1 – area
- 2 – perimeter
- 3 – major axis length
- 4 – minor axis length
- 5 – aspect ratio
- 6 – eccentricity
- 7 – convex area
- 8 – equivalent diameter
- 9 – extent
- 10 – solidity
- 11 – roundness
- 12 – compactness
- 13 – ShapeFactor1
- 14 – ShapeFactor2
- 15 – ShapeFactor3
- 16 – ShapeFactor4

And the output variable in which we will attempt to classify:

- Class of the bean (Seker, Barbunya, Bombay, Cali, Dermosan, Horoz, Sira)

The only categorical variable is our output variable, in which we will encode during EDA.

After some initial EDA, we find the following:

- Dermason: 3546 cases
- Sira: 2636 cases
- Seker: 2027 cases
- Horoz: 1928 cases
- Cali: 1630 cases
- Barbunya: 1322 cases
- Bombay: 522 cases

Clearly, there are some imbalances in the dataset, so we will do the following to balance the dataset:

- We can omit the highest and lowest classes (Dermason and Bombay); this is because Bombay will need to be oversampled by a factor of more than 2.5 if we include Bombay and Dermason is our upper limiting factor in which the rest of the classes must need to be rebalanced.
    - This creates a case where Barbunya class would only need to be rebalanced by a factor of 2, less than the restricting limit of 2.5 that was recommended.
- If we follow the above rebalancing actions:
    - Sira: 2636 cases; does not need change
    - Seker: 2027 cases; oversample by a factor of 1.25
    - Horoz: 1928 cases; oversample by a factor of 1.3
    - Cali: 1630 cases; oversample by a factor of 1.55
    - Barbunya: 1322 cases; oversample by a factor of 2

Exploratory Data Analysis (EDA) and Pre-Processing

After eliminating the classes of Dermason and Bombay, as well as removing 68 duplicate records in our dataset, we arrive at the following distribution:

```
SIRA        2636
SEKER       2027
HOROZ       1860
CALI        1630
BARBUNYA    1322
Name: Class, dtype: int64
```

```
Cleaning Summary
-----------------------------------
Total records: 9543
Removed 68 duplicate rows

Missing Value Summary Below
-----------------------------------

Dry Beans Dataset with DERMASON and BOMBAY removed
---------------
Area               0
Perimeter          0
MajorAxisLength    0
MinorAxisLength    0
AspectRation       0
Eccentricity       0
ConvexArea         0
EquivDiameter      0
Extent             0
Solidity           0
roundness          0
Compactness        0
ShapeFactor1       0
ShapeFactor2       0
ShapeFactor3       0
ShapeFactor4       0
Class              0
y                  0
dtype: int64
```

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiameter	Extent	Solidity	roundness	Compactne
0	28395	610.291	208.178117	173.888747	1.197191	0.549812	28715	190.141097	0.763923	0.988856	0.958027	0.913
1	28734	638.018	200.524796	182.734419	1.097356	0.411785	29172	191.272750	0.783968	0.984986	0.887034	0.953
2	29380	624.110	212.826130	175.931143	1.209713	0.562727	29690	193.410904	0.778113	0.989559	0.947849	0.908
3	30008	645.884	210.557999	182.516516	1.153638	0.498616	30724	195.467062	0.782681	0.976696	0.903936	0.928
4	30140	620.134	201.847882	190.279279	1.060798	0.333680	30417	195.896503	0.773098	0.990893	0.984877	0.970
...	...	...	...	...	...	...	...	...	...	...	...	
10060	58063	941.882	366.689523	202.790970	1.808214	0.833160	58936	271.897237	0.735012	0.985187	0.822463	0.741
10061	58074	910.115	351.958861	210.417832	1.672667	0.801610	58609	271.922992	0.777648	0.990872	0.881047	0.772
10062	59431	956.785	390.489073	194.564645	2.006989	0.867028	60276	275.081623	0.689839	0.985981	0.815820	0.704
10063	60493	931.321	363.814243	212.613752	1.711151	0.811464	61239	277.528521	0.791814	0.987818	0.876428	0.762
10064	63612	984.282	400.931467	204.347808	1.962005	0.860362	64581	284.593243	0.815664	0.984996	0.825106	0.709

9475 rows × 18 columns

Table 1: Preview of cleaned dry beans data frame

The categorical column was encoded with label encoder and named as column 'y'. There are a total of 16 input variables, 1 output variable (ground truth classes), and 1 column of 'y' that we created. A total of 9475 cases are observed for 5 different classes of dry beans (sira, seker, horoz, cali, barbunya).

The correlation heatmap of all 16 predictors and the response class can be seen below:

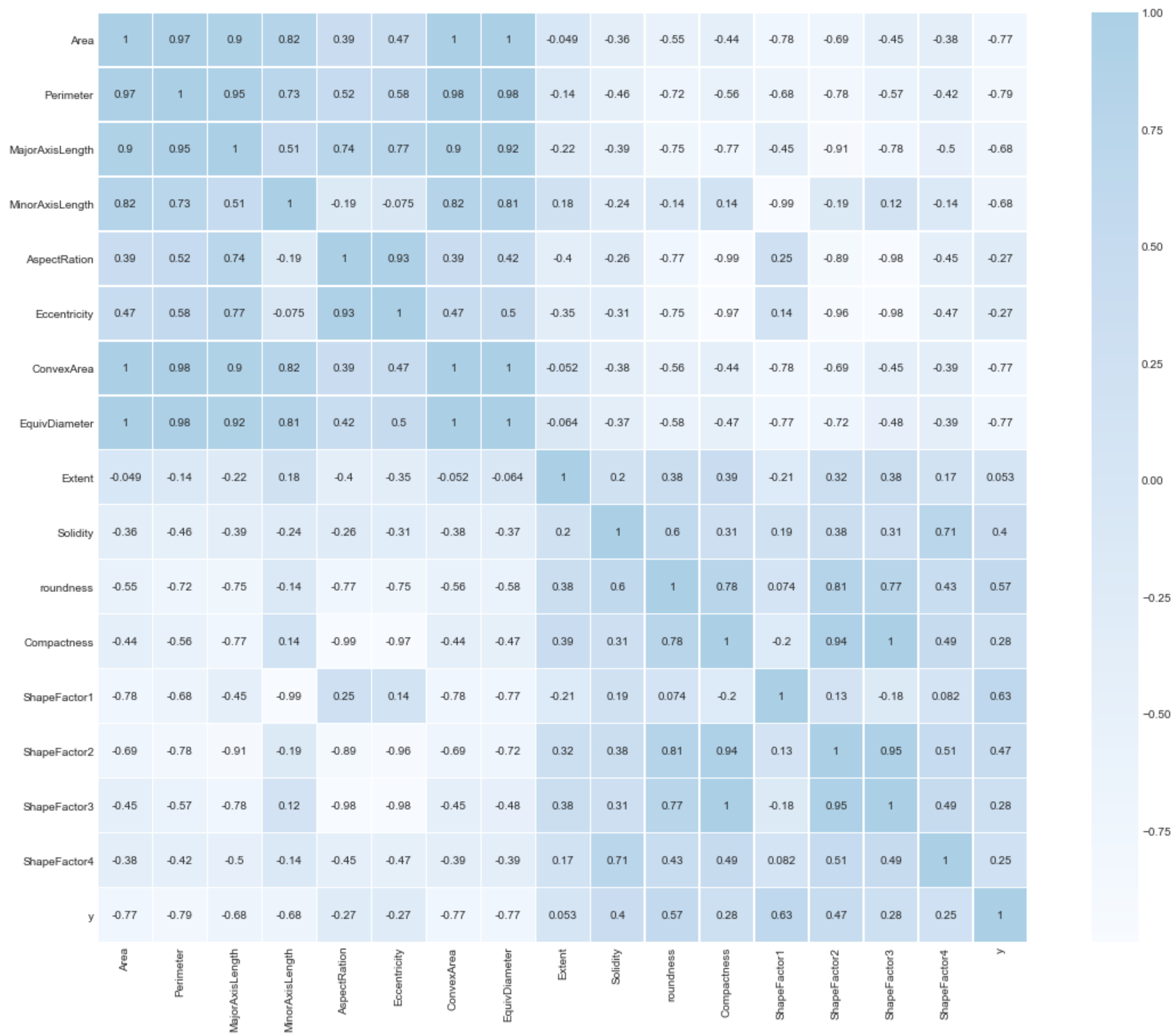

Figure 1: Correlation heatmap of cleaned data frame of dry beans

We can observe there is quite a bit of correlation between a majority of variables. We can handle these correlations in two ways:

1. Drop the highly correlated features
2. Leave them as is because tree-based models are not impacted by correlated features; when tree-based models decide to split, the tree will choose only one of the perfectly correlated features

We can choose the second method here because we are indeed using a tree-based model to evaluate our dry beans dataset.

Next, we can apply feature scaling to our dataset using the standard scaler from sklearn package. Only scaling the input variables and adding on the categorical variable before train and test split (we will use 85% to train and 15% to test), then applying SMOTE after to only the training set (this is because we do not want to have synthetic data in our test set).

Using the parameter sampling_strategy = 'not majority' makes it so the class with the highest number of cases do not get resampled and the rest of the classes will have equal number of cases compared to our highest class of .

In [268]:
```
X_resample_train
```
executed in 19ms, finished 02:12:29 2022-12-13

Out[268]:

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiameter	Extent	Solidity	roundness	Compa
0	2.363153	2.146785	1.932493	1.918112	0.564118	0.671029	2.354376	2.195473	0.151482	-0.363008	-0.991318	-0.
1	-0.976761	-1.198724	-1.322773	-0.212079	-1.347502	-1.540274	-0.984228	-1.018643	0.714800	0.983872	1.678336	1.
2	-1.098313	-1.271461	-1.420657	-0.387292	-1.357641	-1.564047	-1.098769	-1.167461	0.542890	0.541679	1.393947	1.
3	-1.546387	-1.752301	-1.894733	-0.913612	-1.612402	-2.264913	-1.540094	-1.750599	0.463594	0.330827	1.455770	1.
4	-1.385062	-1.663287	-1.751262	-0.657475	-1.587402	-2.185727	-1.390136	-1.533812	0.433505	1.231749	2.026584	1.
...	...	...	...	...	...	...	...	...	...	...	...	
11265	-0.707833	-0.825856	-1.005287	-0.033602	-1.099265	-1.030379	-0.711717	-0.701364	0.162627	0.501424	1.016525	1.
11266	-1.110245	-1.333152	-1.621619	-0.013794	-1.783501	-2.893009	-1.113843	-1.182549	0.581650	0.806200	1.698589	2.
11267	-1.177092	-1.406146	-1.461983	-0.555991	-1.305801	-1.446116	-1.185259	-1.265886	0.090480	1.251420	1.737624	1.
11268	-1.492828	-1.681538	-1.830203	-0.855166	-1.567748	-2.125345	-1.489410	-1.677938	0.958224	0.557398	1.375588	1.
11269	-1.023969	-1.248818	-1.393785	-0.215412	-1.425115	-1.729489	-1.032806	-1.076028	0.822040	1.137521	1.703728	1.

11270 rows × 16 columns

Table 2: Preview of our resampled training set

In [270]:
```
Y_resample_train.value_counts()
```
executed in 13ms, finished 02:12:35 2022-12-13

Out[270]:
```
y
0    2254
1    2254
2    2254
3    2254
4    2254
dtype: int64
```

The result is we have a training set with 11270 rows, each class has 2254 cases of the 5 dry bean classes.

Application of Random Forest and Results

Using the random forest classifier from sklearn package, we can train different classifiers by changing the hyperparameters. Specifically, we can tune the following:

1. The number of trees built by the classifier: 100, 200, 300, 400, 500
2. Minimum impurity decrease, a node will be split if this split induces a decrease of the impurity greater than or equal to this value: 0, 0.02, 0.04, 0.06, 0.08, 0.1
3. The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches: 1, 2, 6, 10

	n_estimators	min_impurity_decrease	min_samples_leaf	OOB Acc	Test Acc
72	400	0.0	1	1.000	0.942
96	500	0.0	1	1.000	0.941
0	100	0.0	1	1.000	0.940
74	400	0.0	6	0.973	0.940
24	200	0.0	1	1.000	0.940
...	...	...	...	...	...
119	500	0.1	10	0.773	0.821
20	100	0.1	1	0.772	0.819
21	100	0.1	2	0.772	0.819
22	100	0.1	6	0.772	0.819
23	100	0.1	10	0.772	0.819

120 rows × 5 columns

Table 3: Hyperparameter tuning of random forest classification

From our hyperparameter tuning, we can choose several models that would be deemed viable for our primary model. Arguments could be made for the top 3 choices in the table above due to computational efficiency as well as the top choice having the best test accuracy. I have chosen the first model with 400 trees, 0.00% min impurity decrease, and minimum number of samples at a leaf node to be 1. Our out of bag accuracy turns out to be 100% and our test set accuracy is 94.2%.

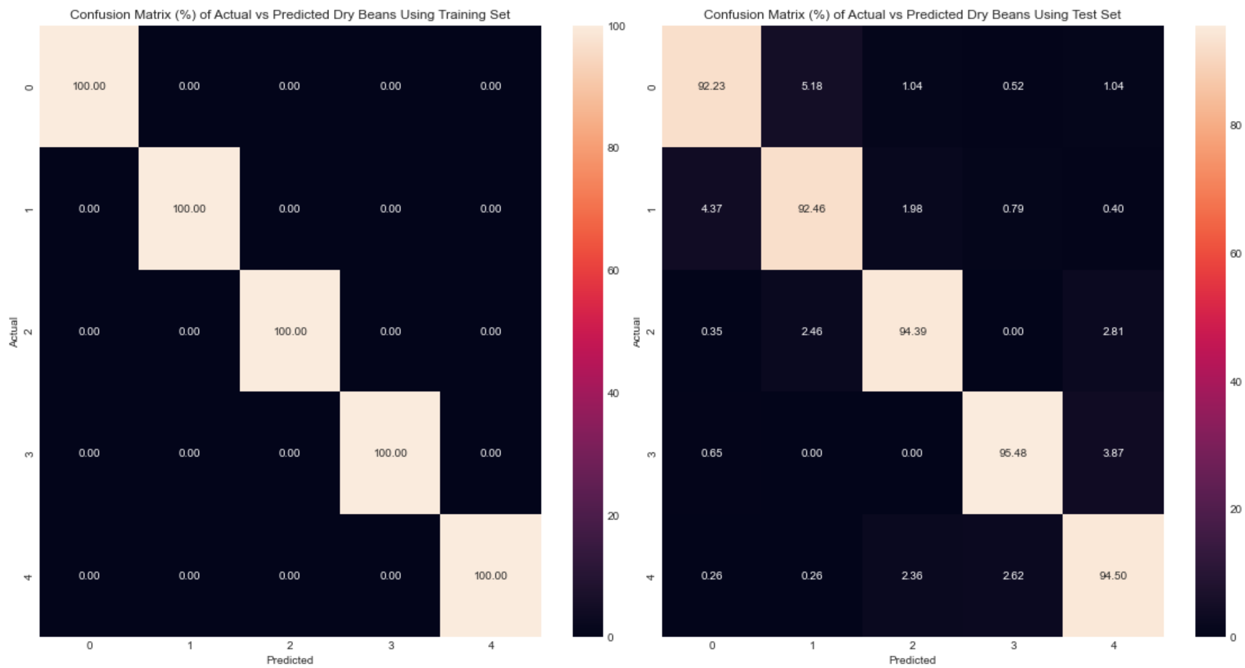

Figure 2: Confusion matrix of training (left figure) and test (right right) sets

Class N	CL0 - BARBUNYA	CL1 - CALI	CL2 - HOROZ	CL3 - SEKER	CL4 - SIRA
OOB Training Accuracy (%)	100%	100%	100%	100%	100%
Test Accuracy (%)	92.23%	92.46%	94.39%	95.48%	94.50%

Table 4: Summary of training and testing accuracy of our classes

From our chosen model, we can see that classes CL3 and CL4 resulted in the highest accuracy on the test set. Hence, we will choose those two classes for our support vector machine application with radial kernel.

Overall, our random forest model has shown accurate classification prediction accuracy above 90%. The procedure of SMOTE has also helped us to balance classes that were imbalanced beforehand.

Application of Radial Kernel Support Vector Machine and Results

We can now implement a radial kernel SVM for our two highest accuracy classes. There are 2 main hyper-parameters used for tuning the radial SVM model, mainly the cost, 'C', and gamma, 'g'. A large value of C gives us low bias and high variance while a small value of C gives us higher bias and lower variance. A large gamma will give us higher bias and low variance while a small gamma will give us low bias and high variance.

We have employed an iterative technique of calculating Hilbert distances for specific values of gamma in order uncover a meaningful range of values from which to choose from. See the below function that was applied to find the gamma range (note: tested for gamma values of 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 25, 50, 100). The resulting was that gamma value of 1 resulted in a convergence of Hilbert's distance values, thus, our max gamma value should be close to 1.

```
def hilb_hist(k,g):
    dis = [] #empty list to store distances
    k = k
    g = g
    index_hilb = random.sample(range(2250),k)
    CL3 = X_resample_train[(Y_resample_train['y']==3)].iloc[index_hilb]
    CL4 = X_resample_train[(Y_resample_train['y']==4)].iloc[index_hilb]
    for i in range(k):
        for j in range(k):
            hil = math.sqrt(2 - (2*math.exp(-g*pow(abs(math.dist(CL3.iloc[i],CL4.iloc[j])),2))))
            dis.append(hil)
    return (dis)
executed in 6ms, finished 03:57:42 2022-12-14
```

Image 1: Function used to find appropriate gamma range

For the values of C, I have chosen the values of: 0.1, 1, 4, 10, 20, 30, 50. We typically do not want such a large value for C as we are attempting to minimize C while having good accuracy.

Next, we implemented a grid search function to find the optimal value of gamma and cost as shown below.

```
# hyperparameter tuning
rad_svm = []
for gamma in (0.01, 0.05, 0.1, 0.5, 1):
    for cost in (0.1, 1, 5, 10, 20, 30, 50):
        radial_svm = SVC(kernel='rbf', gamma=gamma, C=cost, decision_function_shape='ovo',
                         random_state=1).fit(X_train_svm,np.ravel(Y_train_svm))

        radial_svm_train_pred = radial_svm.predict(X_train_svm)
        radial_svm_test_pred = radial_svm.predict(X_test_svm)

        acc_train = round(radial_svm.score(X_train_svm, Y_train_svm),3)

        acc_test = round(radial_svm.score(X_test_svm, Y_test_svm),3)

        rad_svm.append([gamma,cost, acc_train,acc_test])

results_2 = pd.DataFrame(columns=['Gamma', 'Cost', 'Train Acc', 'Test Acc'], data=rad_svm)
executed in 16.3s, finished 03:50:37 2022-12-14
```

Image 2: Function used to find optimal values of gamma and cost

From our hyperparameter tuning, we can choose our top model to be the one with gamma = 0.5, cost = 0.1, with training accuracy at 97.4% and test accuracy at 97.5%.

```
In [361]: results_2.sort_values(by='Test Acc', ascending=False)
```

executed in 31ms, finished 03:50:37 2022-12-14

Out[361]:

	Gamma	Cost	Train Acc	Test Acc
7	0.05	0.1	0.974	0.975
1	0.01	1.0	0.974	0.973
14	0.10	0.1	0.974	0.973
0	0.01	0.1	0.969	0.970
2	0.01	5.0	0.977	0.970
8	0.05	1.0	0.978	0.970
26	0.50	30.0	0.991	0.970
21	0.50	0.1	0.975	0.970
16	0.10	5.0	0.984	0.968

Table 4: Top 9 results of hyperparameter tuning of radial SVM

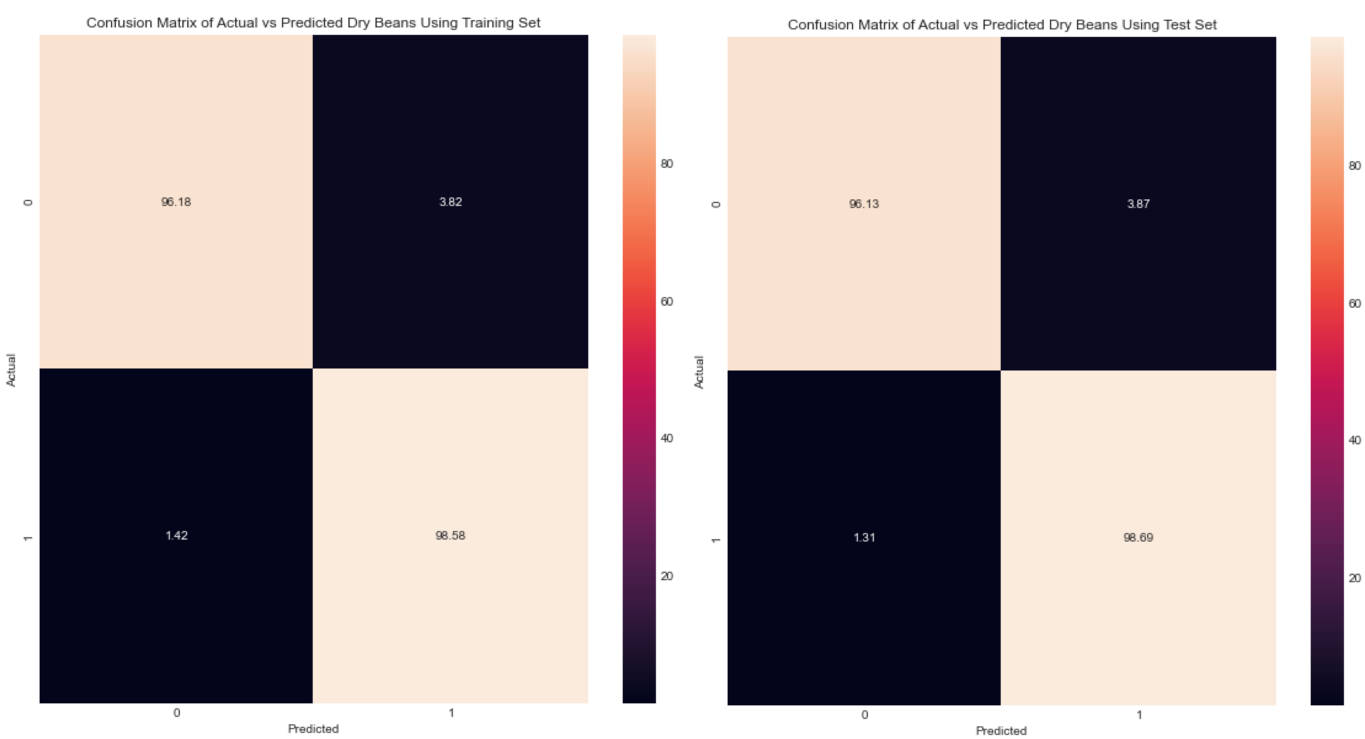

Figure 3: Confusion matrix of training (left figure) and test (right right) sets for our radial SVM model

Appendix A: All code for this report

```
import pandas as pd
import numpy as np
import random
import math
import sklearn
import imblearn
from imblearn.over_sampling import SMOTE
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, LabelEncoder
import os
from scipy import stats
import seaborn as sns
import itertools
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from matplotlib.ticker import PercentFormatter
from sklearn import svm
%matplotlib inline
plt.style.use("seaborn-whitegrid")
import warnings
warnings.filterwarnings("ignore")
```

```
# import the data
data = pd.read_excel('Dry_Bean_Dataset.xlsx')
# data.head()
df = data.copy()
df.head(6)
```

```
# omitting Dermason and Bombay

newdf_1 = (df[df['Class'] != 'DERMASON'])
newdf = (newdf_1[newdf_1['Class'] != 'BOMBAY'])
```

executed in 14ms, finished 04:44:54 2022-12-14

```
newdf.info()
```

executed in 15ms, finished 04:44:54 2022-12-14

```
newdf['Class'] = newdf['Class'].astype('category')
newdf.info()
```

executed in 15ms, finished 04:44:54 2022-12-14

```
# creating instance of labelencoder
labelencoder = LabelEncoder()
# Assigning numerical values and storing in another column
newdf['y'] = labelencoder.fit_transform(newdf['Class'])
newdf
```

executed in 31ms, finished 04:44:54 2022-12-14

```
# This section prints out the following:
# whether there are exact duplicate rows in the dataframe and remove the duplicated row
# how many null values are in each column of newdf dataframe

print("\nCleaning Summary\n{}".format("-"*35))
print("Total records:", newdf.shape[0])
duplicate_rows = newdf.duplicated()
if True in duplicate_rows:
    newdf = newdf[~duplicate_rows]
print("Removed {} duplicate rows".format(np.where(duplicate_rows==True)[0].size))

print("\nMissing Value Summary Below\n{}".format("-"*35))
print("\nDry Beans Dataset with DERMASON and BOMBAY removed\n{}".format("-"*15))

print(newdf.isnull().sum(axis = 0))
```

```
newdf['Class'].value_counts()
#for col in newdf:
 #   print(newdf[col].unique())

# after removing 68 rows of dupcliated values, we arrive at the following spread of classes
# encoded labels
# SIRA = 4
# SEKER = 3
# HOROZ = 2
# CALI = 1
# BARBUNYA = 0
```

```
corr = newdf.corr()
f,axes = plt.subplots(1,1,figsize = (20,15))
sns.heatmap(corr, square=True, annot = True, linewidth = .5, center = 2, ax = axes, cmap='Blues')
```

executed in 2.00s, finished 04:45:01 2022-12-14

```
data_df = newdf
data_df
```

executed in 30ms, finished 04:45:01 2022-12-14

```
scaler = StandardScaler()
```
executed in 14ms, finished 04:45:01 2022-12-14

```
scaler.fit(data_df.iloc[:,:16])
```
executed in 61ms, finished 04:45:01 2022-12-14

```
▾ StandardScaler
StandardScaler()
```

```
X_features = scaler.transform(data_df.iloc[:,:16])
X_features
```
executed in 15ms, finished 04:45:01 2022-12-14

```
X_features_df = pd.DataFrame(X_features, columns=['Area','Perimeter', 'MajorAxisLength', 'MinorAxisLength', 'AspectRation',
                                                  'Eccentricity', 'ConvexArea', 'EquivDiameter', 'Extent', 'Solidity',
                                                  'roundness', 'Compactness', 'ShapeFactor1', 'ShapeFactor2',
                                                  'ShapeFactor3', 'ShapeFactor4'])
X_features_df
```
executed in 31ms, finished 04:45:01 2022-12-14

```
Y_true_class = data_df.iloc[:,17:18]
Y_true_class
```
executed in 15ms, finished 04:45:01 2022-12-14

```
Y_true_class = Y_true_class.reset_index(drop=True)
```
executed in 15ms, finished 04:45:01 2022-12-14

```
Y_true_class.info()
```
executed in 14ms, finished 04:45:01 2022-12-14

```
data_df_final = X_features_df.join(Y_true_class, how='left').reset_index()
data_df_final = data_df_final.iloc[:, 1:]
data_df_final
```
executed in 44ms, finished 04:45:01 2022-12-14

```
data_df_final['y'].astype('category')
```
executed in 13ms, finished 04:45:01 2022-12-14

```
data_df_final.iloc[:, :16]
```
executed in 30ms, finished 04:45:01 2022-12-14

```
X_train, X_test, y_train, y_test = train_test_split(data_df_final.iloc[:, :16], data_df_final.iloc[:, 16:17],
                                                    test_size=0.15, random_state=42)
```
executed in 15ms, finished 04:45:01 2022-12-14

```python
#SMOTE to give new balanced data sets
seed = 123
neighbors = 5
sm = SMOTE(sampling_strategy='not majority', k_neighbors=neighbors, random_state=seed)
X_resample_train, Y_resample_train = sm.fit_resample(X_train, y_train)
```

executed in 679ms, finished 04:45:02 2022-12-14

```python
data = []
for bags in (100,200,300,400,500):
    for pur in (0.00,0.02,0.04,0.06,0.08,0.10):
        for leafs in (1,2,6,10):
            rf_model = RandomForestClassifier(n_estimators=bags, criterion='gini', max_depth=None,
                                              min_samples_split=2, min_samples_leaf=leafs, min_weight_fraction_leaf=0.0,
                                              max_features='sqrt', max_leaf_nodes=None, min_impurity_decrease=pur,
                                              bootstrap=True, oob_score=True, n_jobs=-1, random_state=1, verbose=0,
                                              warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None)
            rf_model_fit = rf_model.fit(X_resample_train,np.ravel(Y_resample_train)) #training the model

            rf_model_fit_train = rf_model_fit.predict(X_resample_train) #predict the training model results

            oob_acc = round(accuracy_score(Y_resample_train, rf_model_fit_train),3) #accuracy of the traininng set

            rf_model_fit_test = rf_model_fit.predict(X_test) #predict the test model results

            test_acc = round(accuracy_score(y_test, rf_model_fit_test),3) #accuracy of the test set

            data.append([bags,pur,leafs, oob_acc, test_acc])

results_1 = pd.DataFrame(columns=['n_estimators', 'min_impurity_decrease','min_samples_leaf', 'OOB Acc', 'Test Acc'], data=data)
```

executed in 3m 42s, finished 04:48:44 2022-12-14

```python
# RUNNNING THE BEST PARAMETERS
rf_model = RandomForestClassifier(n_estimators=500, criterion='gini', max_depth=None,
                                  min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
                                  max_features='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0,
                                  bootstrap=True, oob_score=True, n_jobs=-1, random_state=2, verbose=0,
                                  warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None)

rf_model_fit = rf_model.fit(X_resample_train,np.ravel(Y_resample_train)) #training the model

rf_model_fit_train = rf_model_fit.predict(X_resample_train) #predict the training model results

print(accuracy_score(Y_resample_train, rf_model_fit_train)) #accuracy of the traininng set

rf_model_fit_test = rf_model_fit.predict(X_test) #predict the test model results

print(accuracy_score(y_test, rf_model_fit_test)) #accuracy of the test set

train_cm_1 = confusion_matrix(Y_resample_train, rf_model_fit_train)
test_cm_1 = confusion_matrix(y_test, rf_model_fit_test)
```

executed in 5.97s, finished 04:48:50 2022-12-14

```python
cm_y_test = confusion_matrix(y_test, rf_model_fit_test)
cm_y_test
```

executed in 15ms, finished 04:48:50 2022-12-14

```
array([[178,  10,   2,   1,   2],
       [ 11, 233,   5,   2,   1],
       [  1,   7, 269,   0,   8],
       [  2,   0,   0, 296,  12],
       [  1,   1,   9,  10, 361]], dtype=int64)
```

```python
cm_y_train = confusion_matrix(Y_resample_train, rf_model_fit_train)
cm_y_train
```

executed in 14ms, finished 04:48:50 2022-12-14

```
array([[2254,    0,    0,    0,    0],
       [   0, 2254,    0,    0,    0],
       [   0,    0, 2254,    0,    0],
       [   0,    0,    0, 2254,    0],
       [   0,    0,    0,    0, 2254]], dtype=int64)
```

```python
results_1.sort_values(by='Test Acc', ascending=False)
```

executed in 29ms, finished 04:48:50 2022-12-14

```python
# Normalise
cmn = cm_y_test.astype('float') / cm_y_test.sum(axis=1)[:, np.newaxis]
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(cmn, annot=True, fmt='.2f')
plt.title("Confusion Matrix of Actual vs Predicted Dry Beans Using Test Set")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show(block=False)
```

executed in 233ms, finished 04:48:50 2022-12-14

```python
X_train_svm = X_resample_train[(Y_resample_train['y']==3) | (Y_resample_train['y']==4)]
Y_train_svm = Y_resample_train[(Y_resample_train['y']==3) | (Y_resample_train['y']==4)]
X_test_svm = X_test[(y_test['y']==3) | (y_test['y']==4)]
Y_test_svm = y_test[(y_test['y']==3) | (y_test['y']==4)]
```

executed in 18ms, finished 04:50:17 2022-12-14

```python
def hilb_hist(k,g):
    dis = [] #empty list to store distances
    k = k
    g = g
    index_hilb = random.sample(range(850),k)
    CL4 = X_res_train[(Y_resample_train[(Y_resample_train['y']==4)].iloc[index_hilb]
    CL3 = X_res_train[(Y_resample_train[(Y_resample_train['y']==3)].iloc[index_hilb]
    for i in range(k):
        for j in range(k):
            hil = math.sqrt(2 - (2*math.exp(-g*pow(abs(math.dist(CL3.iloc[i],CL4.iloc[j])),2))))
            dis.append(hil)
    return (dis)

np.mean(hilb_hist(k=300, g=1))
plt.hist(hilb_hist(k=300, g=1))
sns.histplot(hilb_hist(k=300, g=0.0001), stat='probability')
```

executed in 16ms, finished 04:50:18 2022-12-14

```python
rad_svm = []
for gamma in (0.01, 0.05, 0.1, 0.5, 1):
    for cost in (0.1, 1, 5, 10, 20, 30, 50):
        radial_svm = svm.SVC(kernel='rbf', gamma=gamma, C=cost, decision_function_shape='ovo',
                             random_state=1).fit(X_train_svm,np.ravel(Y_train_svm))

        radial_svm_train_pred = radial_svm.predict(X_train_svm)
        radial_svm_test_pred = radial_svm.predict(X_test_svm)

        acc_train = round(radial_svm.score(X_train_svm, Y_train_svm),3)
        acc_test = round(radial_svm.score(X_test_svm, Y_test_svm),3)

        rad_svm.append([gamma,cost, acc_train,acc_test])

results_2 = pd.DataFrame(columns=['Gamma', 'Cost', 'Train Acc', 'Test Acc'], data=rad_svm)
```

executed in 16.0s, finished 04:50:38 2022-12-14

```python
results_2.sort_values(by='Test Acc', ascending=False)
```

executed in 27ms, finished 04:55:54 2022-12-14

```python
radial_svm = svm.SVC(kernel='rbf', gamma=0.05, C=0.1, decision_function_shape='ovo', random_state=1).fit(X_train_svm,np.ravel(Y_t
radial_svm_train_pred = radial_svm.predict(X_train_svm)
radial_svm_test_pred = radial_svm.predict(X_test_svm)
```

executed in 358ms, finished 04:51:02 2022-12-14

```python
round(radial_svm.score(X_train_svm, Y_train_svm),3)
```

executed in 223ms, finished 04:55:28 2022-12-14

```
0.974
```

```python
round(radial_svm.score(X_test_svm, Y_test_svm),3)
```

executed in 39ms, finished 04:55:29 2022-12-14

```
0.975
```

```python
cm_y_test_svm = confusion_matrix(Y_test_svm, radial_svm_test_pred)
cm_y_test_svm
```

executed in 9ms, finished 04:57:14 2022-12-14

```
array([[298,  12],
       [  5, 377]], dtype=int64)
```

```python
cm_y_train_svm = confusion_matrix(Y_train_svm, radial_svm_train_pred)
cm_y_train_svm
```

executed in 23ms, finished 04:57:38 2022-12-14

```
array([[2168,   86],
       [  32, 2222]], dtype=int64)
```

```python
# Normalise
cmn = cm_y_test_svm.astype('float')*100 / cm_y_test_svm.sum(axis=1)[:, np.newaxis]
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(cmn, annot=True, fmt='.2f')
plt.title("Confusion Matrix of Actual vs Predicted Dry Beans Using Test Set")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show(block=False)
```

executed in 140ms, finished 04:57:55 2022-12-14

```python
# Normalise
cmn = cm_y_train_svm.astype('float')*100 / cm_y_train_svm.sum(axis=1)[:, np.newaxis]
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(cmn, annot=True, fmt='.2f')
plt.title("Confusion Matrix of Actual vs Predicted Dry Beans Using Training Set")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show(block=False)
```

executed in 128ms, finished 04:58:15 2022-12-14