iml554-classification-assignment04

September 15, 2024

1 Case:

Accurate prediction of fire occurrence can help in timely interventions and resource allocation. In this case study, we use a dataset containing various features related to forest fires to build and evaluate Support Vector Machine (SVM) models for classification.

Objective

The goal is to apply different SVM kernels (linear, polynomial, and radial basis function (RBF)) to classify forest fire occurrences and determine which kernel performs best based on accuracy Text, e.g., 'small', 'large')

2 Data Description

Metadata for dataset forestfires.csv :

Columns:

month: Categorical ("jan" to "dec")

day: Categorical ("mon" to "sun")

FFMC: Numeric (Fine Fuel Moisture Code, range typically 18.7 to 96.20)

DMC: Numeric (Duff Moisture Code, range typically 1.1 to 291.3)

DC: Numeric (Drought Code, range typically 7.9 to 860.6)

ISI: Numeric (Initial Spread Index, range typically 0.0 to 56.10)

temp: Numeric (Temperature in Celsius, range typically 2.2 to 33.30)

RH: Numeric (Relative Humidity, range typically 15.0 to 100%)

wind: Numeric (Wind Speed in km/h, range typically 0.40 to 9.40)

rain: Numeric (Rainfall in mm/m2, range typically 0.0 to 6.4)

area: Numeric (Area burned in hectares, continuous variable; 0.00 to 1090.8)

dayfri, daymon, daysat, daysun, daythu, daytue, daywed: Binary (One-hot encoded days of the week)

monthapr, monthaug, monthdec, monthfeb, monthjan, monthjul, monthjun, monthmar, monthmay, monthnov, monthoct, monthsep: Binary (One-hot encoded months)

size_category: Categorical (Text, e.g., 'small', 'large')

3 Q1) Write a Python code snippet to load the dataset forestfires.csv from a given path and display the first 5 rows of the dataset. (1 M)

[431]:
```
# Importing required packages
import numpy as np
import pandas as pd
import warnings as war
war.filterwarnings("ignore")
```

[432]:
```
# Defining dataset csv Path
dataSetPath="C:\\Users\\ASUS\\jupyterworkspace\\Assignment & Mini␣
 ↪Project\\Module_03_Classification\\Assignment\\04_Support Vector␣
 ↪Machine\\forestfires(1).csv"
# Loading dataSet
dataSetRead=pd.read_csv(dataSetPath)
```

[433]:
```
# Displaying first 5 records to confirming data loading
print("**************************************************Displaying below␣
 ↪first 5 records************************************************************")
dataSetRead.head()
```

```
**************************************************Displaying below first 5
records************************************************************
```

[433]:

	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	…	monthfeb	\
0	mar	fri	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	…	0	
1	oct	tue	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	…	0	
2	oct	sat	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	…	0	
3	mar	fri	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	…	0	
4	mar	sun	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	…	0	

	monthjan	monthjul	monthjun	monthmar	monthmay	monthnov	monthoct	\
0	0	0	0	1	0	0	0	
1	0	0	0	0	0	0	1	
2	0	0	0	0	0	0	1	
3	0	0	0	1	0	0	0	
4	0	0	0	1	0	0	0	

	monthsep	size_category
0	0	small
1	0	small
2	0	small
3	0	small

```
4         0          small

[5 rows x 31 columns]
```

[434]:
```
# Displaying last 5 records to confirming data loading
print("**************************************************Displaying below␣
  ↪last 5 records************************************************************")
dataSetRead.tail()
```

```
**************************************************Displaying below last 5
records************************************************************
```

[434]:

	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	…	monthfeb	\
512	aug	sun	81.6	56.7	665.6	1.9	27.8	32	2.7	0.0	…	0	
513	aug	sun	81.6	56.7	665.6	1.9	21.9	71	5.8	0.0	…	0	
514	aug	sun	81.6	56.7	665.6	1.9	21.2	70	6.7	0.0	…	0	
515	aug	sat	94.4	146.0	614.7	11.3	25.6	42	4.0	0.0	…	0	
516	nov	tue	79.5	3.0	106.7	1.1	11.8	31	4.5	0.0	…	0	

	monthjan	monthjul	monthjun	monthmar	monthmay	monthnov	monthoct	\
512	0	0	0	0	0	0	0	
513	0	0	0	0	0	0	0	
514	0	0	0	0	0	0	0	
515	0	0	0	0	0	0	0	
516	0	0	0	0	0	1	0	

	monthsep	size_category
512	0	large
513	0	large
514	0	large
515	0	small
516	0	small

[5 rows x 31 columns]

[435]:
```
# Displaying dimension of dataSet
print("Dimention of Dataset:- {}".format(dataSetRead.shape[0:2]))
print("Total number of rows in Dataset:- {}".format(dataSetRead.shape[0]))
print("Total number of columns in Dataset:- {}".format(dataSetRead.shape[1]))
```

```
Dimention of Dataset:- (517, 31)
Total number of rows in Dataset:- 517
Total number of columns in Dataset:- 31
```

[436]:
```
# Displaying description & statistcal summary of the dataSet
dataSetRead.describe().T
```

[436]:

	count	mean	std	min	25%	50%	75%	max
FFMC	517.0	90.644681	5.520111	18.7	90.2	91.60	92.90	96.20
DMC	517.0	110.872340	64.046482	1.1	68.6	108.30	142.40	291.30
DC	517.0	547.940039	248.066192	7.9	437.7	664.20	713.90	860.60
ISI	517.0	9.021663	4.559477	0.0	6.5	8.40	10.80	56.10
temp	517.0	18.889168	5.806625	2.2	15.5	19.30	22.80	33.30
RH	517.0	44.288201	16.317469	15.0	33.0	42.00	53.00	100.00
wind	517.0	4.017602	1.791653	0.4	2.7	4.00	4.90	9.40
rain	517.0	0.021663	0.295959	0.0	0.0	0.00	0.00	6.40
area	517.0	12.847292	63.655818	0.0	0.0	0.52	6.57	1090.84
dayfri	517.0	0.164410	0.371006	0.0	0.0	0.00	0.00	1.00
daymon	517.0	0.143133	0.350548	0.0	0.0	0.00	0.00	1.00
daysat	517.0	0.162476	0.369244	0.0	0.0	0.00	0.00	1.00
daysun	517.0	0.183752	0.387657	0.0	0.0	0.00	0.00	1.00
daythu	517.0	0.117988	0.322907	0.0	0.0	0.00	0.00	1.00
daytue	517.0	0.123791	0.329662	0.0	0.0	0.00	0.00	1.00
daywed	517.0	0.104449	0.306138	0.0	0.0	0.00	0.00	1.00
monthapr	517.0	0.017408	0.130913	0.0	0.0	0.00	0.00	1.00
monthaug	517.0	0.355899	0.479249	0.0	0.0	0.00	1.00	1.00
monthdec	517.0	0.017408	0.130913	0.0	0.0	0.00	0.00	1.00
monthfeb	517.0	0.038685	0.193029	0.0	0.0	0.00	0.00	1.00
monthjan	517.0	0.003868	0.062137	0.0	0.0	0.00	0.00	1.00
monthjul	517.0	0.061896	0.241199	0.0	0.0	0.00	0.00	1.00
monthjun	517.0	0.032882	0.178500	0.0	0.0	0.00	0.00	1.00
monthmar	517.0	0.104449	0.306138	0.0	0.0	0.00	0.00	1.00
monthmay	517.0	0.003868	0.062137	0.0	0.0	0.00	0.00	1.00
monthnov	517.0	0.001934	0.043980	0.0	0.0	0.00	0.00	1.00
monthoct	517.0	0.029014	0.168007	0.0	0.0	0.00	0.00	1.00
monthsep	517.0	0.332689	0.471632	0.0	0.0	0.00	1.00	1.00

[437]:
```
# Displaying the columns and their respective data types
dataSetRead.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 517 entries, 0 to 516
Data columns (total 31 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   month           517 non-null    object
 1   day             517 non-null    object
 2   FFMC            517 non-null    float64
 3   DMC             517 non-null    float64
 4   DC              517 non-null    float64
 5   ISI             517 non-null    float64
 6   temp            517 non-null    float64
 7   RH              517 non-null    int64
 8   wind            517 non-null    float64
```

```
 9   rain           517 non-null    float64
 10  area           517 non-null    float64
 11  dayfri         517 non-null    int64
 12  daymon         517 non-null    int64
 13  daysat         517 non-null    int64
 14  daysun         517 non-null    int64
 15  daythu         517 non-null    int64
 16  daytue         517 non-null    int64
 17  daywed         517 non-null    int64
 18  monthapr       517 non-null    int64
 19  monthaug       517 non-null    int64
 20  monthdec       517 non-null    int64
 21  monthfeb       517 non-null    int64
 22  monthjan       517 non-null    int64
 23  monthjul       517 non-null    int64
 24  monthjun       517 non-null    int64
 25  monthmar       517 non-null    int64
 26  monthmay       517 non-null    int64
 27  monthnov       517 non-null    int64
 28  monthoct       517 non-null    int64
 29  monthsep       517 non-null    int64
 30  size_category  517 non-null    object
dtypes: float64(8), int64(20), object(3)
memory usage: 125.3+ KB
```

[438]:
```
# Checking total no. of NULL values for attributes specific
missingValue_Count=dataSetRead_New.isnull().sum()
print(missingValue_Count)
```

```
FFMC        0
DMC         0
DC          0
ISI         0
temp        0
RH          0
wind        0
rain        0
area        0
dayfri      0
daymon      0
daysat      0
daysun      0
daythu      0
daytue      0
daywed      0
monthapr    0
monthaug    0
monthdec    0
```

```
monthfeb    0
monthjan    0
monthjul    0
monthjun    0
monthmar    0
monthmay    0
monthnov    0
monthoct    0
monthsep    0
large       0
small       0
large       0
small       0
dtype: int64
```

[439]:
```
# Checking duplicate values in dataSet
duplicatevalue_Count=dataSetRead_New.duplicated().sum()
print("Total duplicates values in dataSet:- {}".format(duplicatevalue_Count))
```

```
Total duplicates values in dataSet:- 8
```

[440]:
```
# Removing duplicate rows in dataSet
dataSetRead = dataSetRead.drop_duplicates()
```

[441]:
```
# Verifing the duplicates values after removing
duplicate_values = dataSetRead.duplicated().sum()
print(f'Total duplicates values in dataSet after removing: {duplicate_values}')
```

```
Total duplicates values in dataSet after removing: 0
```

4 Q2) Drop the unnecessary columns and apply min-max normalization to scale the feature columns (1.5 M)

[442]:
```
# dropping the unnecessary columns from data set
dataSetRead=dataSetRead.drop(columns=['dayfri', 'daymon', 'daysat', 'daysun',␣
 ↪'daythu','daytue', 'daywed', 'monthapr', 'monthaug', 'monthdec',
                 'monthfeb','monthjan', 'monthjul', 'monthjun', 'monthmar',␣
 ↪'monthmay', 'monthnov','monthoct','monthsep'],axis=1)
```

[443]:
```
# Displaying all records from dataSet after dropping unnecessary columns
dataSetRead
```

[443]:

	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area
0	mar	fri	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	0.00
1	oct	tue	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	0.00
2	oct	sat	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	0.00

```
3     mar  fri  91.7   33.3   77.5   9.0   8.3  97   4.0   0.2   0.00
4     mar  sun  89.3   51.3  102.2   9.6  11.4  99   1.8   0.0   0.00
..    …  …   …    …    …   …   …  ..   …   …    …
512   aug  sun  81.6   56.7  665.6   1.9  27.8  32   2.7   0.0   6.44
513   aug  sun  81.6   56.7  665.6   1.9  21.9  71   5.8   0.0  54.29
514   aug  sun  81.6   56.7  665.6   1.9  21.2  70   6.7   0.0  11.16
515   aug  sat  94.4  146.0  614.7  11.3  25.6  42   4.0   0.0   0.00
516   nov  tue  79.5    3.0  106.7   1.1  11.8  31   4.5   0.0   0.00

    size_category
0           small
1           small
2           small
3           small
4           small
..            …
512         large
513         large
514         large
515         small
516         small

[509 rows x 12 columns]
```

[444]:
```
# importing required package
# Performing LabelEncoder for 'month' & 'day' attribute
from sklearn.preprocessing import LabelEncoder
label_Encoder = LabelEncoder()
dataSetRead['month'] = label_Encoder.fit_transform(dataSetRead['month'])
dataSetRead['day'] = label_Encoder.fit_transform(dataSetRead['day'])
```

[445]:
```
# Performing LabelEncoder for 'size_category' target attribute
dataSetRead.size_category.replace(('small', 'large'), (0, 1), inplace = True)
```

[446]:
```
# displaying all records in data set after LabelEncoding
dataSetRead
```

[446]:
```
     month  day  FFMC    DMC     DC   ISI  temp  RH  wind  rain   area  \
0        7    0  86.2   26.2   94.3   5.1   8.2  51   6.7   0.0   0.00
1       10    5  90.6   35.4  669.1   6.7  18.0  33   0.9   0.0   0.00
2       10    2  90.6   43.7  686.9   6.7  14.6  33   1.3   0.0   0.00
3        7    0  91.7   33.3   77.5   9.0   8.3  97   4.0   0.2   0.00
4        7    3  89.3   51.3  102.2   9.6  11.4  99   1.8   0.0   0.00
..     …  …   …    …    …   …   …  ..   …   …    …
512      1    3  81.6   56.7  665.6   1.9  27.8  32   2.7   0.0   6.44
513      1    3  81.6   56.7  665.6   1.9  21.9  71   5.8   0.0  54.29
514      1    3  81.6   56.7  665.6   1.9  21.2  70   6.7   0.0  11.16
```

```
515      1    2  94.4  146.0  614.7  11.3  25.6  42   4.0   0.0   0.00
516      9    5  79.5    3.0  106.7   1.1  11.8  31   4.5   0.0   0.00

     size_category
0                0
1                0
2                0
3                0
4                0
..             …
512              1
513              1
514              1
515              0
516              0

[509 rows x 12 columns]
```

[447]:
```
# Calculating Pearson correlation matrix for the entire DataFrame
corr_matrix = dataSetRead.corr()

# Displaying the correlation matrix
corr_matrix
```

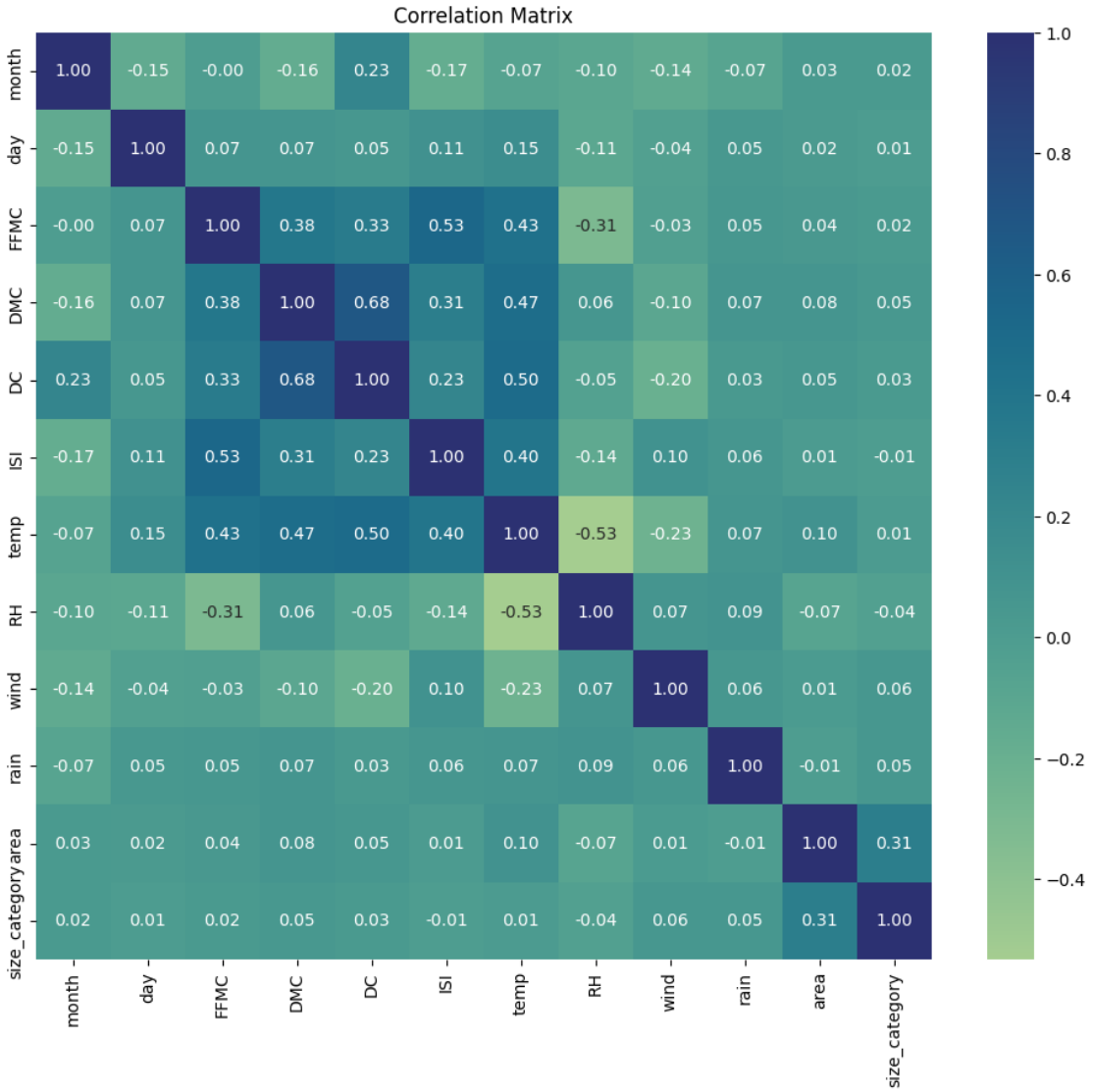
[447]:
```
                 month       day      FFMC       DMC        DC       ISI  \
month         1.000000 -0.149053 -0.003122 -0.161662  0.231234 -0.172492
day          -0.149053  1.000000  0.073244  0.068438  0.053815  0.113982
FFMC         -0.003122  0.073244  1.000000  0.382925  0.331956  0.531926
DMC          -0.161662  0.068438  0.382925  1.000000  0.681446  0.309459
DC            0.231234  0.053815  0.331956  0.681446  1.000000  0.229757
ISI          -0.172492  0.113982  0.531926  0.309459  0.229757  1.000000
temp         -0.069237  0.151230  0.431744  0.470875  0.496608  0.395858
RH           -0.097232 -0.109650 -0.306262  0.062772 -0.047971 -0.135955
wind         -0.140640 -0.035736 -0.030409 -0.104563 -0.202856  0.104834
rain         -0.072934  0.047466  0.052280  0.071017  0.033772  0.062132
area          0.028569  0.019449  0.040998  0.076932  0.051542  0.008429
size_category 0.024288  0.005360  0.024292  0.046598  0.028372 -0.007867

                  temp        RH      wind      rain      area  size_category
month         -0.069237 -0.097232 -0.140640 -0.072934  0.028569       0.024288
day            0.151230 -0.109650 -0.035736  0.047466  0.019449       0.005360
FFMC           0.431744 -0.306262 -0.030409  0.052280  0.040998       0.024292
DMC            0.470875  0.062772 -0.104563  0.071017  0.076932       0.046598
DC             0.496608 -0.047971 -0.202856  0.033772  0.051542       0.028372
ISI            0.395858 -0.135955  0.104834  0.062132  0.008429      -0.007867
temp           1.000000 -0.532503 -0.227754  0.067911  0.099173       0.009976
RH            -0.532503  1.000000  0.070721  0.094343 -0.074554      -0.039730
```

```
wind           -0.227754  0.070721  1.000000  0.059332  0.012623       0.059513
rain            0.067911  0.094343  0.059332  1.000000 -0.006511       0.053463
area            0.099173 -0.074554  0.012623 -0.006511  1.000000       0.310561
size_category   0.009976 -0.039730  0.059513  0.053463  0.310561       1.000000
```

[448]:
```
# visualizing the correlation matrix with a heatmap
# importing required packages
import seaborn as sbn
import matplotlib.pyplot as plt
```

[449]:
```
# Using Seaborn to create a heatmap
plt.figure(figsize=(12, 10))
sbn.heatmap(corr_matrix, annot=True, cmap='crest', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```



4.1 Analysis:-

Feature Correlation with size_category

month: 0.024288 (very low)

day: 0.005360 (very low)

FFMC: 0.024292 (very low)

DMC: 0.046598 (very low)

DC: 0.028372 (very low)

ISI: -0.007867 (very low)

temp: 0.009976 (very low)

RH: -0.039730 (very low)

wind: 0.059513 (low)

rain: 0.053463 (low)

area: 0.310561 (moderate)

4.1.1 Keep Features with Moderate/Low Correlation:

area: Shows a moderate correlation with the size_category. This feature might be more useful for the model.

wind:- Shows a low correlation with the size_category. This feature might be more useful for the model.

rain:- Shows a low correlation with the size_category. This feature might be more useful for the model.

[450]:
```
# Droping irrelevant features based on correlation matrix analysis due to very␣
↪low coreelation with target value
dataSetRead = dataSetRead.drop(columns=['month','day','FFMC','DMC','DC',␣
↪'ISI','temp','RH'],axis=1)
```

4.2 Data scaling

[451]:
```
# importing required package
# MinMax Scaler is used to perform feature scalling
from sklearn.preprocessing import MinMaxScaler
scalling=MinMaxScaler()
```

[452]:
```
# displaying list numerical columns toi scale
numerical_columns = ['wind', 'rain', 'area']
```

```
# performing Min-Max Scaling to the numerical columns
dataSetRead[numerical_columns] = scalling.
 ↪fit_transform(dataSetRead[numerical_columns])
```

[453]:
```
# displaying all records in data set after Minmac scalling
dataSetRead
```

[453]:
```
          wind     rain      area  size_category
0     0.700000  0.00000  0.000000              0
1     0.055556  0.00000  0.000000              0
2     0.100000  0.00000  0.000000              0
3     0.400000  0.03125  0.000000              0
4     0.155556  0.00000  0.000000              0
..         …      …        …             …
512   0.255556  0.00000  0.005904              1
513   0.600000  0.00000  0.049769              1
514   0.700000  0.00000  0.010231              1
515   0.400000  0.00000  0.000000              0
516   0.455556  0.00000  0.000000              0

[509 rows x 4 columns]
```

5 Q3) Write the code to split the dataset into training and testing sets with a 75-25 split. Hint: Consider target as size_category (1 M)

5.1 Split data in to features & target

[454]:
```
# Dropping target variable
X=dataSetRead.drop('size_category',axis=1)
# Taking target variavle
y=dataSetRead['size_category']
```

[455]:
```
# Checking percentagewise distiribution for "size_category" target variable
dataSetRead['size_category'].value_counts(normalize=True).mul(100).round(2)
```

[455]:
```
size_category
0    72.89
1    27.11
Name: proportion, dtype: float64
```

5.2 Spliting the Data into Training and Testing Sets¶

[456]:
```
# Importing train_test_split package
from sklearn.model_selection import train_test_split
# Splitting train & test data
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.
 ↪25,random_state=42,stratify=y)
```

[457]:
```
# Displaying dimenstion of train & test dataset
print('Shape of X_train = ', X_train.shape)
print('Shape of y_train = ', y_train.shape)
print('Shape of X_test = ', X_test.shape)
print('Shape of y_test = ', y_test.shape)
```

```
Shape of X_train =  (381, 3)
Shape of y_train =  (381,)
Shape of X_test =  (128, 3)
Shape of y_test =  (128,)
```

6 Q4) Write the code to train SVM model with kernels (Linear, polynomial and RBF) and compare the performance. Also, calculate the accuracy scores for each kernel? (4 M)

[458]:
```
# Importing library of SVC
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,␣
 ↪classification_report,confusion_matrix
```

[459]:
```
# Creating instance of SVM classifier with Linear Kernal
svm_Linear = SVC(kernel='linear')
```

[460]:
```
# Training the SVM model using training data set
svm_Linear.fit(X_train, y_train)
```

[460]:
```
SVC(kernel='linear')
```

[461]:
```
# Prediction on test data for SVM classifier with Linear Kernal
y_prediction_svm_Linear=svm_Linear.predict(X_test)
```

[462]:
```
# Calculationg the accuracy of SVM classifier with Linear Kernal
accuracy_svm_Linear = accuracy_score(y_test, y_prediction_svm_Linear)
print("SVM model accuracy with Linear Kernal: {}".
 ↪format(round(accuracy_svm_Linear,2)))
```

```
SVM model accuracy with Linear Kernal: 0.73
```

[463]:
```
# Evaluating classfication report for SVC classifier with Linear Kernal
report_svm_Linear=classification_report(y_test,y_prediction_svm_Linear)
print(report_svm_Linear)
```

```
              precision    recall  f1-score   support

           0       0.73      1.00      0.84        93
           1       0.00      0.00      0.00        35

    accuracy                           0.73       128
   macro avg       0.36      0.50      0.42       128
weighted avg       0.53      0.73      0.61       128
```

[464]:
```
# Evaluating confusion matrix for SVM classifier with Linear Kernal
conf_matrix_svm_Linear=confusion_matrix(y_test,y_prediction_svm_Linear)
print(conf_matrix_svm_Linear)
```

```
[[93  0]
 [35  0]]
```

[465]:
```
falsePositive_svm_Linear = conf_matrix_svm_Linear.sum(axis=0) - np.
 ↪diag(conf_matrix_svm_Linear)
falseNegative_svm_Linear = conf_matrix_svm_Linear.sum(axis=1) - np.
 ↪diag(conf_matrix_svm_Linear)
truePositive_svm_Linear = np.diag(conf_matrix_svm_Linear)
trueNegative_svm_Linear = conf_matrix_svm_Linear.sum() -␣
 ↪(falsePositive_svm_Linear + falseNegative_svm_Linear +␣
 ↪truePositive_svm_Linear)
print('************* SVM model with Linear Kernal*************')
for i in range(len(truePositive_svm_Linear)):
    print(f"Class {i}:")
    print(f"truePositive: {truePositive_svm_Linear[i]}, falsePositive:␣
 ↪{falsePositive_svm_Linear[i]}, falseNegative: {falseNegative_svm_Linear[i]},␣
 ↪trueNegative: {trueNegative_svm_Linear[i]}")
    print()
```

```
************* SVM model with Linear Kernal*************
Class 0:
truePositive: 93, falsePositive: 35, falseNegative: 0, trueNegative: 0

Class 1:
truePositive: 0, falsePositive: 0, falseNegative: 35, trueNegative: 93
```

[466]:
```
# Creating instance of SVM classifier with Polynomial Kernal
svm_Polynomial = SVC(kernel='poly', degree=3)  # Polynomial kernel with degree 3
```

[467]:
```
# Training the SVM model using training data set
svm_Polynomial.fit(X_train, y_train)
```

[467]:
```
SVC(kernel='poly')
```

[468]:
```
# Prediction on test data for SVM classifier with Polynomial Kernal
y_prediction_svm_Polynomial=svm_Polynomial.predict(X_test)
```

[469]:
```
# Calculationg the accuracy of SVM classifier with Polynomial Kernal
accuracy_svm_Polynomial = accuracy_score(y_test, y_prediction_svm_Polynomial)
print("SVM model accuracy with Polynomial Kernal: {}".
 ↪format(round(accuracy_svm_Polynomial,2)))
```

```
SVM model accuracy with Polynomial Kernal: 0.86
```

[470]:
```
# Evaluating classfication report for SVM classifier with Polynomial Kernal
report_svm_Polynomial=classification_report(y_test,y_prediction_svm_Polynomial)
print(report_svm_Polynomial)
```

```
              precision    recall  f1-score   support

           0       0.84      1.00      0.91        93
           1       1.00      0.49      0.65        35

    accuracy                           0.86       128
   macro avg       0.92      0.74      0.78       128
weighted avg       0.88      0.86      0.84       128
```

[471]:
```
# Evaluating confusion matrix for SVM classifier with Polynomial Kernal
conf_matrix_svm_Polynomial=confusion_matrix(y_test,y_prediction_svm_Polynomial)
print(conf_matrix_svm_Polynomial)
```

```
[[93  0]
 [18 17]]
```

[472]:
```
falsePositive_svm_Polynomial = conf_matrix_svm_Polynomial.sum(axis=0) - np.
 ↪diag(conf_matrix_svm_Polynomial)
falseNegative_svm_Polynomial = conf_matrix_svm_Polynomial.sum(axis=1) - np.
 ↪diag(conf_matrix_svm_Polynomial)
truePositive_svm_Polynomial = np.diag(conf_matrix_svm_Polynomial)
trueNegative_svm_Polynomial = conf_matrix_svm_Polynomial.sum() -␣
 ↪(falsePositive_svm_Polynomial + falseNegative_svm_Polynomial +␣
 ↪truePositive_svm_Polynomial)
print('************* SVM model with Polynomial Kernal*************')
for i in range(len(truePositive_svm_Polynomial)):
    print(f"Class {i}:")
```

```
    print(f"truePositive: {truePositive_svm_Polynomial[i]}, falsePositive:␣
 ↪{falsePositive_svm_Polynomial[i]}, falseNegative:␣
 ↪{falseNegative_svm_Polynomial[i]}, trueNegative:␣
 ↪{trueNegative_svm_Polynomial[i]}")
    print()
```

```
************* SVM model with Polynomial Kernal*************
Class 0:
truePositive: 93, falsePositive: 18, falseNegative: 0, trueNegative: 17

Class 1:
truePositive: 17, falsePositive: 0, falseNegative: 18, trueNegative: 93
```

[473]:
```
# Creating instance of SVM classifier with Radial Basis Function kernel
svm_RBF = SVC(kernel='rbf')
```

[474]:
```
# Training the SVM model using training data set
svm_RBF.fit(X_train, y_train)
```

[474]:
```
SVC()
```

[475]:
```
# Prediction on test data for SVM classifier with Radial Basis Function kernel
y_prediction_svm_RBF=svm_RBF.predict(X_test)
```

[476]:
```
# Calculationg the accuracy of SVM classifier with Radial Basis Function kernel
accuracy_svm_RBF = accuracy_score(y_test, y_prediction_svm_RBF)
print("SVM model accuracy with Radial Basis Function kernel: {}".
 ↪format(round(accuracy_svm_RBF,2)))
```

```
SVM model accuracy with Radial Basis Function kernel: 0.78
```

[477]:
```
# Evaluating classfication report for SVM classifier with Radial Basis Function␣
 ↪kernel
report_svm_RBF=classification_report(y_test,y_prediction_svm_RBF)
print(report_svm_RBF)
```

```
              precision    recall  f1-score   support

           0       0.77      1.00      0.87        93
           1       1.00      0.20      0.33        35

    accuracy                           0.78       128
   macro avg       0.88      0.60      0.60       128
weighted avg       0.83      0.78      0.72       128
```

[478]:
```
# Evaluating confusion matrix for SVM classifier with Radial Basis Function␣
 ↪kernel
conf_matrix_svm_RBF=confusion_matrix(y_test,y_prediction_svm_RBF)
print(conf_matrix_svm_RBF)
```

```
[[93  0]
 [28  7]]
```

[479]:
```
falsePositive_svm_RBS = conf_matrix_svm_RBF.sum(axis=0) - np.
 ↪diag(conf_matrix_svm_RBF)
falseNegative_svm_RBS = conf_matrix_svm_RBF.sum(axis=1) - np.
 ↪diag(conf_matrix_svm_RBF)
truePositive_svm_RBS = np.diag(conf_matrix_svm_RBF)
trueNegative_svm_RBS = conf_matrix_svm_RBF.sum() - (falsePositive_svm_RBS +␣
 ↪falseNegative_svm_RBS + truePositive_svm_RBS)
print('************* SVM model with Radial Basis Function kernel*************')
for i in range(len(truePositive_svm_RBS)):
    print(f"Class {i}:")
    print(f"truePositive: {truePositive_svm_RBS[i]}, falsePositive:␣
 ↪{falsePositive_svm_RBS[i]}, falseNegative: {falseNegative_svm_RBS[i]},␣
 ↪trueNegative: {trueNegative_svm_RBS[i]}")
    print()
```

```
************* SVM model with Radial Basis Function kernel*************
Class 0:
truePositive: 93, falsePositive: 28, falseNegative: 0, trueNegative: 7

Class 1:
truePositive: 7, falsePositive: 0, falseNegative: 28, trueNegative: 93
```

7 Q5) Write a summary of which kernel performed best on test data and why. State conclusion properly. (1.5M)

7.0.1 Summary and Conclusion

Performance Overview:

Linear Kernel: Achieved an accuracy of 73% on the test data. This kernel assumes a linear decision boundary and is suited for linearly separable data. However, the lower accuracy suggests that the linear assumption may not adequately capture the complexities in the data.

Polynomial Kernel: Achieved an accuracy of 86% on the test data. This kernel maps the data into a higher-dimensional space using polynomial functions, allowing the model to capture non-linear relationships and interactions in the data. The highest accuracy indicates that this kernel is particularly effective for the given dataset, suggesting that the data has complex patterns that a polynomial kernel can better model.

Radial Basis Function (RBF) Kernel: Achieved an accuracy of 78% on the test data. The RBF

kernel also maps data into a higher-dimensional space but uses a Gaussian function to measure similarity between data points. While the RBF kernel improved performance over the linear kernel, it did not perform as well as the polynomial kernel.

Conclusion:

The polynomial kernel performed the best on the test data with an accuracy of 86%. This superior performance indicates that the polynomial kernel is particularly well-suited for the dataset, effectively capturing the non-linear patterns and interactions that the linear and RBF kernels could not fully model. The higher accuracy achieved with the polynomial kernel suggests that the decision boundaries required to separate the classes are complex and best represented by polynomial functions.

The RBF kernel, while improving upon the linear kernel, did not match the polynomial kernel's performance, potentially due to suboptimal parameter settings or the nature of the data. Therefore, based on the test data performance, the polynomial kernel is the most effective choice for this specific problem.