Gurleen Kaur

0668818

Letter Recognition

Dataset Description:

Our task is to identify each of a large number of black and white rectangular pixel displays as one of the 26 capital letters in English alphabet. The images of the alphabets were based on 20 different fonts and were randomly distorted to produce a file of 20,000 unique stimuli. Then each of this stimulus was converted to 16 numerical attributes representing statistical moments and edge counts. These 16 numerical attributes were scaled to fit into a range of integers values 0 to 15. The 16 numerical attributes are given below:

1. **x-box**: The horizontal position, counting pixels from the left edge of the image, of the center of the smallest rectangular box that can be drawn with all "on" pixels inside the box.
2. **y-box**: The vertical position, counting pixels from the bottom, of the above box.
3. **width**: The width, in pixels, of the box.
4. **height**: The height, in pixels, of the box.
5. **onpix**: The total number of "on" pixels in the character image.
6. **x-bar**: The mean horizontal position of all "on" pixels relative to the center of the box and divided by the width of the box.
7. **y-bar**: The mean vertical position of all "on" pixels relative to the center of the box and divided by the height of the box.
8. **x2bar**: The mean squared value of the horizontal pixel distances as measured in 6 above. This attribute will have a higher value for images whose pixels are more widely separated in the horizontal direction as would be the case for the letters W or M.
9. **y2bar**: The mean squared value of the vertical pixel distances as measured in 7 above.
10. **xybar:** The mean product of the horizontal and vertical distances for each "on" pixel as measured in 6 and 7 above. This attribute has a positive value for diagonal lines that run from bottom left to top right and a negative value for diagonal lines from top left to bottom right.
11. **x2ybar**: The mean value of the squared horizontal distance times the vertical distance for each "on" pixel. This measures the correlation of the horizontal variance with the vertical position.
12. **xy2bar:** The mean value of the squared vertical distance times the horizontal distance for each "on" pixel. This measures the correlation of the vertical variance with the horizontal position.
13. **x-edge**: The mean number of edges (an "on" pixel immediately to the right of either an "off" pixel or the image boundary) encountered when making systematic scans from left

to right at all vertical positions within the box. This measure distinguishes between letters like "W" or "M" and letters like 'T' or "L."

14. **xedgey**: The sum of the vertical positions of edges encountered as measured in 13 above. This feature will give a higher value if there are more edges at the top of the box, as in the letter "Y."
15. **y-edge**: The mean number of edges (an "on" pixel immediately above either an "off" pixel or the image boundary) encountered when making systematic scans of the image from bottom to top over all horizontal positions within the box.
16. **yedgex:** The sum of horizontal positions of edges encountered as measured in 15 above.

The python code starts from here:

Code 1:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import scale
#Reading the data from a csv file
letter_df=pd.read_csv("C:/Users/gurleen/Desktop/DataMining/Final project/letter-recognition.csv")
#Displaying Number of Rows and Columns
print(letter_df.shape)
```

Output 1:

```
---------------------------------------
(20000, 17)
<class 'pandas.core.frame.DataFrame'>
```

Code 2:

```
#Displaying Columns Information
print(letter_df.info())
```

Output 2:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000 entries, 0 to 19999
Data columns (total 17 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   letter  20000 non-null  object
 1   x-box   20000 non-null  int64
 2   y-box   20000 non-null  int64
 3   width   20000 non-null  int64
 4   height  20000 non-null  int64
 5   onpix   20000 non-null  int64
 6   x-bar   20000 non-null  int64
 7   y-bar   20000 non-null  int64
 8   x2bar   20000 non-null  int64
 9   y2bar   20000 non-null  int64
 10  xybar   20000 non-null  int64
 11  x2ybar  20000 non-null  int64
 12  xy2bar  20000 non-null  int64
 13  x-edge  20000 non-null  int64
 14  xedgey  20000 non-null  int64
 15  y-edge  20000 non-null  int64
 16  yedgex  20000 non-null  int64
dtypes: int64(16), object(1)
```

Code 3:

```
#Displaying first 5 rows of the dataset
pd.set_option("display.max.columns", None)
pd.set_option('display.width', 1000)
print(letter_df.head(5))
```

Output 3:

	letter	x-box	y-box	width	height	onpix	x-bar	y-bar	x2bar	y2bar	xybar	x2ybar	xy2bar	x-edge	xedgey	y-edge	yedgex
0	T	2	8	3	5	1	8	13	0	6	6	10	8	0	8	0	8
1	I	5	12	3	7	2	10	5	5	4	13	3	9	2	8	4	10
2	D	4	11	6	8	6	10	6	2	6	10	3	7	3	7	3	9
3	N	7	11	6	6	3	5	9	4	6	4	4	10	6	10	2	8
4	G	2	1	3	1	1	8	6	6	6	6	5	9	1	7	5	10

Code 4:

```
#Displaying mean of all columns of the dataset
print(letter_df.mean(axis = 0))
```

Output 4:

```
x-box      4.02355
y-box      7.03550
width      5.12185
height     5.37245
onpix      3.50585
x-bar      6.89760
y-bar      7.50045
x2bar      4.62860
y2bar      5.17865
xybar      8.28205
x2ybar     6.45400
xy2bar     7.92900
x-edge     3.04610
xedgey     8.33885
y-edge     3.69175
yedgex     7.80120
```

Code 5:

```
#Displaying variance of all columns of the dataset
print(letter_df.var(axis=0))
```

Output 5:

```
x-box       3.660378
y-box      10.920086
width       4.058506
height      5.113887
onpix       4.798106
x-bar       4.104819
y-bar       5.407270
x2bar       7.289827
y2bar       5.668318
xybar       6.192507
x2ybar      6.922530
xy2bar      4.328975
x-edge      5.440747
xedgey      2.392350
y-edge      6.589861
yedgex      2.616209
dtype: float64
```

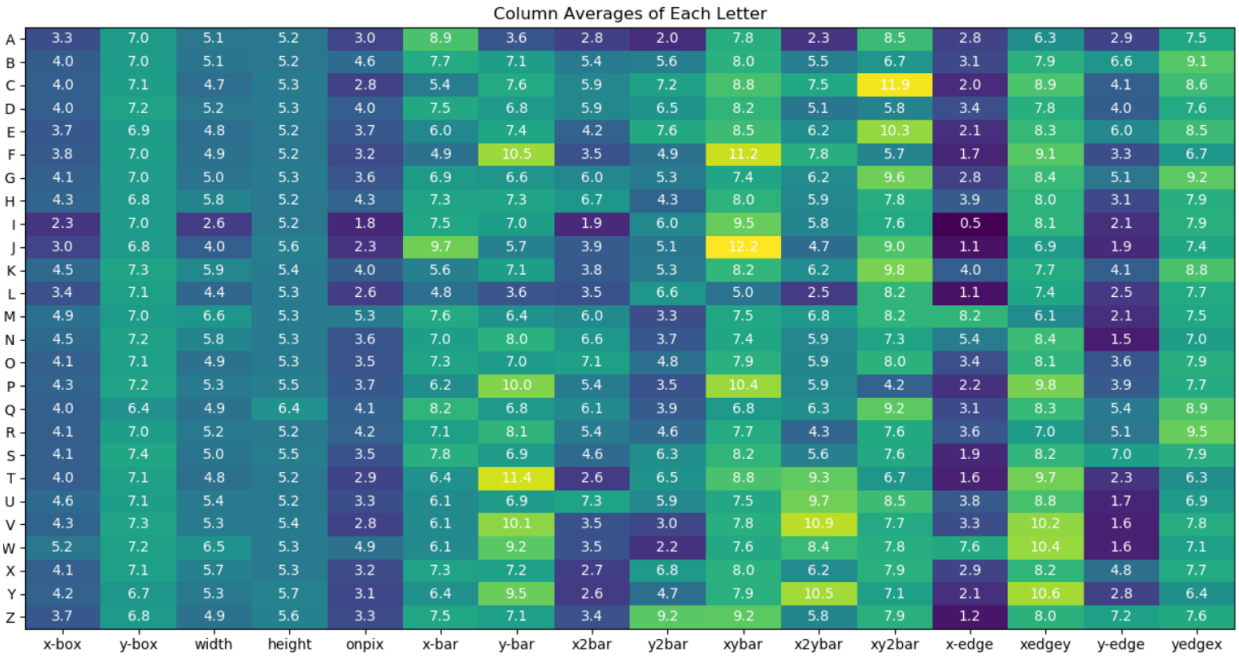
Visualization

Code 6:

```
#Visulaizing all the columns averages for each letter
#Creating data set grouped by alphabets and the mean of every column (dataset will also have the column letter)
avg_data=letter_df.groupby('letter').mean().reset_index()
#Creating list of letters, column names and numpy array of the values for each letter in the respective columns
averages=avg_data.iloc[:,1:].values
letters=list(avg_data['letter'])
col_names=list(avg_data.columns)
col_names=col_names[1:]
fig, ax = plt.subplots()
#Using imshow to plot the map
mp = ax.imshow(averages,aspect='auto')
ax.set_xticks(np.arange(len(col_names)))
ax.set_yticks(np.arange(len(letters)))
ax.set_xticklabels(col_names)
ax.set_yticklabels(letters)
#Adding the values into the map as text
for i in range(len(letters)):
    for j in range(len(col_names)):
        text = ax.text(j, i, round(averages[i, j],1),
                       ha="center", va="center", color="w")

ax.set_title("Column Averages of Each Letter")
plt.show()
```

Output 6:

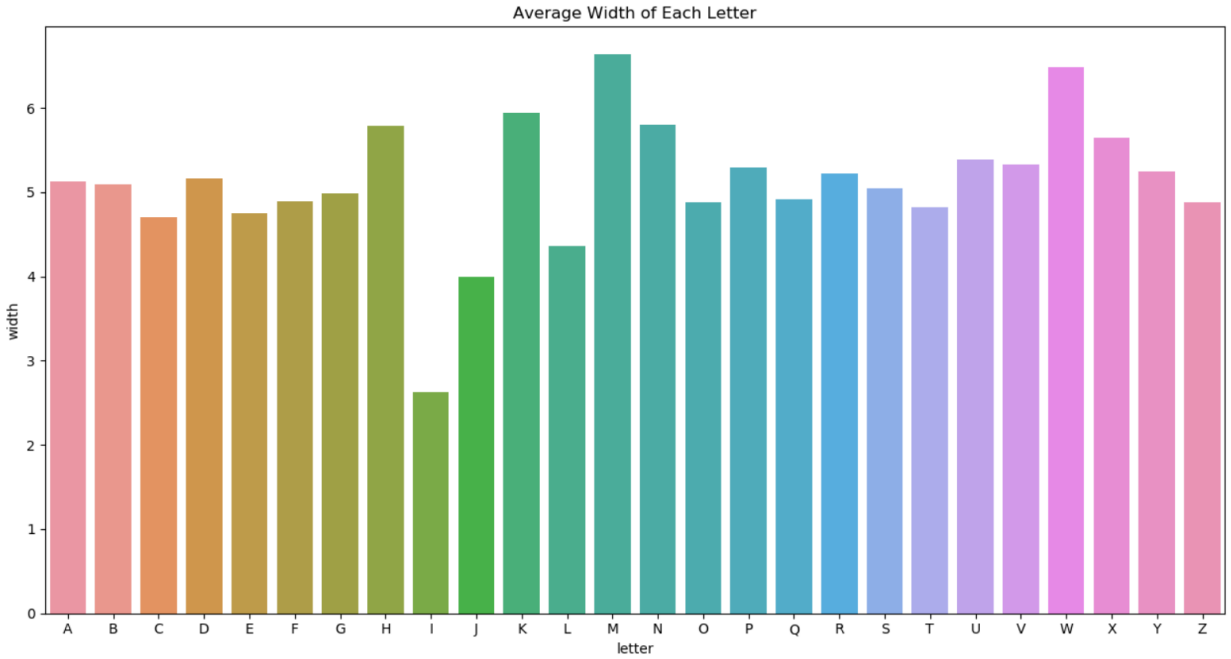


This is basically the average table created shown in the code. It shows all the average values of each column for each letter. Higher the average goes, lighter the color becomes.

Code 7:

```
#Visualizing average width of all the letters
sns.barplot(x='letter',y='width',data=avg_data)
plt.title("Average Width of Each Letter")
plt.show()
```

Output 7:

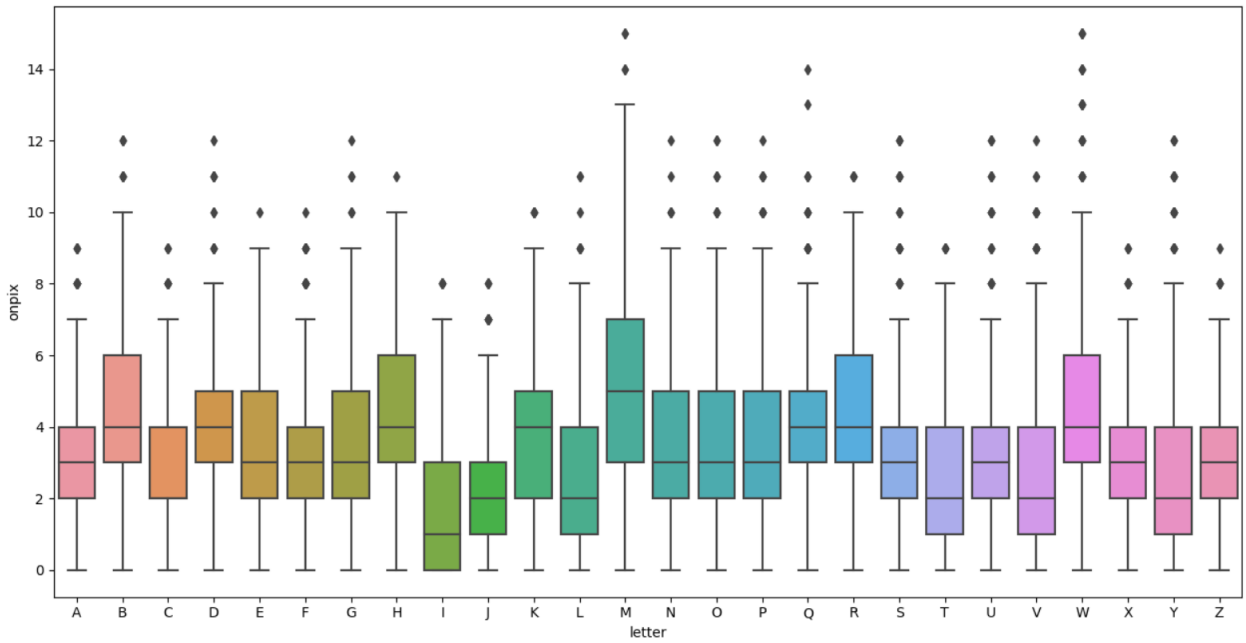


The bar graph shows the average width for each letter. ‘M’ and ‘W’ shows the higher average width than others and ‘I’ the shows the smallest average width.

Code 8:

```
#Visualizing 'on' pixels of all the letters
sns.boxplot(x='letter',y='onpix',data=letter_df,order=letters)
plt.show()
```

Output 8:

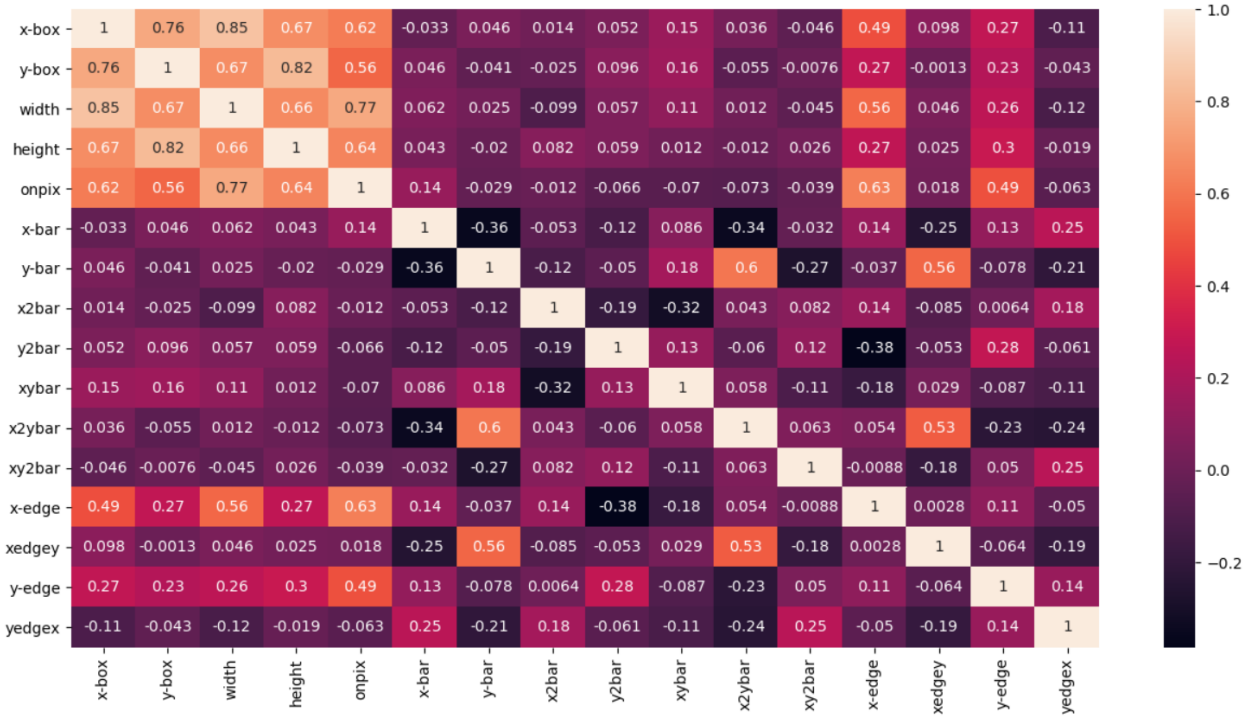


The box plot shows the min, max, quantiles and mean ‘ON’ pixels for each letter. I see a similar trend as followed by the average width of each letter. Let’s see the correlation matrix among the columns.

Code 9:

```
#Visualizing Correlation matrix
cor_df=letter_df[1:]
cor_mat=cor_df.corr()
sns.heatmap(cor_mat, annot=True)
plt.show()
```

Output 9:



As thought so, I can see a strong relationship between ‘onpix’ and ‘width’ (0.77) among other strong relationships such as ‘x-box’ and ‘width’ (0.85) and ‘y-box’ and ‘height’ (0.82).

Preprocessing

The dataset does not need much preprocessing as the values for each column were already fitted to the range 0 to 15 and the dataset had no missing or duplicate values. So, I just divided the data into two sets: training and test dataset. To standardize the dataset, I rescaled it to have mean = 0 and standard deviation = 1.

Code 10:

```
#Preprocessing data
#Splitting it into X and Y
X = letter_df.drop("letter", axis = 1)
y = letter_df['letter']
#Standardizing the dataset (mean=0,std=1)
X_scaled = scale(X)
```

I have split the training and testing data to 75,25 ratio for best results.

```
#Splitting the data into training and test data (75,25 split)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size = 0.25, random_state = 101)
```

Data Mining Algorithms and their Comparison

As the object is to classify the stimulus as one of the 26 English alphabets, classification algorithms will be used for creating the machine learning models.

Classification Algorithm 1: Naïve Bayes

Code 11:

```
#Applying Naive Bayes algorithm
gaussianNB_model = GaussianNB()
gaussianNB_model.fit(X_train,y_train)
#Predicting our test data using the model created above
pred_NB=gaussianNB_model.predict(X_test)
#Computing the accuracy of the Decision Tree model
print("Accuracy of Naive Bayes Model:", metrics.accuracy_score(y_true=y_test, y_pred=pred_NB), "\n")
#Computing the confusion metrics of the Naive Bayes model
print(metrics.confusion_matrix(y_true=y_test, y_pred=pred_NB))
```

Output 11:

```
Accuracy of Naive Bayes Model: 0.6504
```

Confusion Matrix:

```
[[148   0   0   0   0   0   0   1   0   1   0   0   3   2   0   0   1   2
    8   0   0   0   1   1   3   0]
 [  0 130   0   4   0   0   2   1  17   0   2   0   6   0   1   0   1  12
    1   0   0   0   4   2   0   0]
 [  0   0 150   0   6   1   9   0   0   0  10   0   1   0   7   1   4   0
    2   3   3   0   0   0   0   0]
 [  3  18   0 149   0   0   0   0  10   2   5   0   3   0   4   1   1   7
    2   0   0   0   0   1   0   0]
 [  0   3   3   0  64   0  36   1  20   0   3   0   0   0   0   0  10   0
    9   4   2   0   1  30   2   3]
 [  0   9   0   6   0 127   4   0   0   0   0   0   0   3   0  15   1   0
    3   4   0   0   3   5   2   0]
 [  5   4  22   1   0   0 114   0   6   0   7   0   6   1   1   0   9   3
    6   0   0   0   4   0   0   0]
 [  3   8   0  12   0   2   3  55   2   0   5   0  11   3  26   0   1  10
    1   1   3   2   5  35   1   0]
 [  0   5   0   9   1   3   0   0 141   6   1   3   0   0   0   1   1   0
    9   0   0   0   0   0   0   1]
 [  0   7   0  11   0   5   0   0   6 134   0   0   0   1   2   2   1   4
   10   0   0   0   0   1   0   2]
 [  0   7   1   5  22   0   8   2   4   0  96   0  11   3   0   0   0  19
    0   1   4   0   1  28   0   0]
 [  1   6   1   0   1   0   3   0   0  12   8 155   0   0   0   0   9   2
    1   0   0   0   1   0   1   0]
 [  4   3   0   0   0   0   0   3   0   0   4   0 173   0   1   0   0   0
    0   0   0   0   6   0   0   0]
 [  0   1   0   6   0   2   0  16   1   0   3   0   2 141   5   0   0   2
    0   0   5   6   9   0   0   0]
 [  1   3   2   3   0   0   5   1   6   0   1   0   7   1 128   1   3   8
    0   0   0   0   5   0   0   0]
 [  0   3   0   7   0  20   5   2   0   0   0   0   1   1   1 161   1   0
    0   2   0   0  12   0   2   0]
 [  6   1   0   1   0   0   3   1   1   0   0   0   6   0  46   0 109   3
   10   0   0   0   1   2   1   1]
 [  1  12   0   9   0   0   2   7   3   6   7   0  15   0   1   0   3 123
    0   0   0   0   5   1   0   0]
 [ 12  40   0   2   2   2   0   2  25   1   0   0   0   0   0   0   9   1
   48   6   3   0   0  26   0   7]
 [  0   0   0   2   4   9   1   0   0   0   4   0   0   0   0   0   0   0
    6 153   0   4   0   8  11   1]
 [  0   0   4   0   0   0   0   3   0   0  11   1  13   2   7   0   0   0
    0   0 146   0   1   1   0   0]
 [  0   2   0   0   0   2   3   1   0   0   0   0   5   1   0   5   0   0
    1   1   0 142  17   0   4   0]
 [  0   0   0   0   0   0   2   1   0   0   0   0  12   0   2   0   0   0
    0   0   0  12 160   0   0   0]
 [  0   9   0   5   4   0   0   0  24   1   6   2   0   0  21   0   0   0
    6   2   9   0   0 107   1   5]
 [  0   0   0   0   0  12   0   0   0   0   1   0   2   0   1   0   3   0
   11  26   1  46  11   0  85   0]
 [  1   0   0   0  11   0   0   0  25   1   0   0   0   0   0   0   1   2
   24   3   0   0   0   3   3 113]]
```

The accuracy for this model is not very good (65%). We can also see from the confusion matrix, the true positives are not quite high. The Naïve Bayes classifier assumes that there is no dependency among the attributes, which in our case is not quite correct as we saw from the correlation matrix. As Naive Bayes only works if the decision boundary is linear, elliptic, or parabolic, may be the data is not linear. The above two reasons can result in low accuracy. Let's apply other algorithms and see if this justifies.

Classification Algorithm 2: SVM (linear)

Code 12:

```
#Applying SVM algorithm as 'linear' kernel function
linear_model = SVC(kernel='linear')
linear_model.fit(X_train, y_train)
#Predicting our test data using the model created above
pred_linear_model = linear_model.predict(X_test)
#Computing the accuracy of the linear model
print("Accuracy of SVM Linear Model:", metrics.accuracy_score(y_true=y_test, y_pred=pred_linear_model), "\n")
#Computing the confusion metrics of the linear model
print(metrics.confusion_matrix(y_true=y_test, y_pred=pred_linear_model))
```

Output 12:

```
Accuracy of SVM Linear Model: 0.8576

[[162   0   0   1   0   0   1   0   0   0   1   1   0   0   0   0   0   1
    0   0   1   0   0   0   3   0]
 [  0 161   0   0   0   1   3   3   0   0   1   0   0   1   0   0   1   7
    3   0   0   1   0   1   0   0]
 [  1   0 172   0   7   0   8   1   0   0   4   0   0   0   3   0   0   0
    0   0   1   0   0   0   0   0]
 [  1   8   0 181   0   0   2   1   1   0   0   0   0   5   3   0   0   4
    0   0   0   0   0   0   0   0]
 [  0   0   3   0 165   2   5   0   0   0   1   5   0   0   0   0   2   2
    0   2   0   0   0   1   0   3]
 [  0   0   0   2   1 161   1   3   1   0   0   0   0   1   0   3   0   0
    1   6   0   0   0   0   2   0]
 [  0   0   9   5   2   2 145   0   0   0   4   2   2   0   0   1   6   1
    6   0   0   2   2   0   0   0]
 [  0   6   3   9   0   3   3 119   0   1   4   2   2   0  10   0   5  11
    0   0   4   2   0   4   1   0]
 [  0   0   1   3   0   4   0   0 158   8   0   0   0   0   1   0   0   0
    2   0   0   0   0   3   0   1]
 [  1   0   0   2   0   1   0   1   8 162   0   0   0   1   1   0   0   1
    5   0   1   0   0   0   0   2]
 [  0   1   4   2   0   0   0   3   0   0 173   4   1   0   0   0   0  13
    0   0   0   0   0  11   0   0]
 [  1   1   3   2   1   0   7   1   0   0   1 178   0   0   0   0   3   0
    2   1   0   0   0   0   0   0]
 [  0   2   0   0   0   0   0   3   0   0   0   0 184   1   0   0   0   2
    0   0   0   0   2   0   0   0]
 [  0   0   0   1   0   0   0   5   0   0   0   0   2 186   2   1   0   0
    0   0   0   1   1   0   0   0]
 [  1   0   3   4   0   0   0  19   0   0   0   0   1   0 127   2   2   3
    0   0   3   0  10   0   0   0]
 [  0   1   0   2   0  14   3   0   0   1   2   0   0   0   0 187   0   0
    0   0   0   1   0   7   0   0]
 [  3   0   0   0   3   0   7   0   0   2   0   1   0   0   5   0 162   0
    6   0   0   0   1   0   0   2]
 [  9   5   0   2   0   1   5   3   0   0   9   0   1   2   4   0   1 152
    0   0   0   0   0   1   0   0]
 [  0  11   0   0   8   4   4   0   4   0   0   2   0   0   0   1   9   1
  126   5   0   0   0   2   0   9]
 [  0   0   0   1   1   3   1   0   1   0   0   0   0   0   0   1   0   1
    3 185   0   0   0   1   1   4]
 [  2   0   1   1   0   0   0   1   0   0   0   0   2   1   3   0   0   0
    0   0 178   0   0   0   0   0]
 [  2   2   0   0   0   0   1   3   0   0   0   0   0   1   0   1   0   2
    0   0   0 165   6   0   1   0]
 [  0   0   0   0   0   0   1   0   0   0   0   0   5   0   2   0   0   0
    0   0   0   0 181   0   0   0]
 [  0   1   0   4   4   0   1   0   2   1   3   3   0   0   0   0   0   0
    2   2   1   0   0 176   1   1]
 [  2   0   0   0   0   2   0   1   0   0   0   0   0   0   0   0   0   0
    0   3   2   9   0   0 180   0]
 [  1   0   0   0   3   1   0   0   0   2   0   0   0   0   0   0   3   0
   14   1   0   0   0   0   0 162]]
```

The SVM (linear) model show a decent accuracy (85%). Much higher than the Naïve Bayes. So, there are very high chances of Naïve Bayes failure because of its class conditional independence assumption. Let's check for a non-linear SVM algorithm and see if the accuracy increases.

Classification Algorithm 3: SVM (radial basis function)

Code 13:

```
#Applying SVM algorithm as 'rbf'(Radial Basis Function) kernel function
#Parameters of non linear classifications:C and gamma are set to default
nonLinear_model = SVC(kernel='rbf')
nonLinear_model.fit(X_train, y_train)
#Predicting our test data using the model created above
pred_nonLinear_model = nonLinear_model.predict(X_test)
#Computing the accuracy of the linear model
print("Accuracy of SVM Non Linear Model:", metrics.accuracy_score(y_true=y_test, y_pred=pred_nonLinear_model), "\n")
#Computing the confusion metrics of the non linear model
print(metrics.confusion_matrix(y_true=y_test, y_pred=pred_nonLinear_model))
```

Output 13:

```
Accuracy of SVM Non Linear Model: 0.9442

[[169   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   2   0]
 [  0 172   0   1   1   0   1   0   0   0   0   0   0   0   0   0   0   6
    1   0   0   0   0   1   0   0]
 [  0   0 182   0   4   0   7   0   0   0   0   0   0   0   4   0   0   0
    0   0   0   0   0   0   0   0]
 [  0   1   0 199   0   0   1   1   0   0   0   0   0   3   1   0   0   0
    0   0   0   0   0   0   0   0]
 [  0   0   0   0 181   1   6   0   0   0   0   0   0   0   0   0   0   2
    0   0   0   0   0   0   0   1]
 [  0   0   0   0   0 177   1   0   1   0   0   0   0   1   0   0   0   0
    0   2   0   0   0   0   0   0]
 [  0   0   0   4   1   1 177   0   0   0   0   0   1   0   1   0   0   1
    0   0   0   1   2   0   0   0]
 [  0   6   0   4   0   1   4 154   0   0   2   1   1   0   2   0   3  10
    0   0   1   0   0   0   0   0]
 [  0   0   1   0   0   0   0   0 169   9   0   0   0   0   0   1   0   0
    0   0   0   0   0   1   0   0]
 [  1   0   0   0   0   0   0   0   6 174   0   0   0   1   2   0   0   0
    2   0   0   0   0   0   0   0]
 [  0   1   0   2   0   0   0   4   0   0 190   0   1   0   0   0   0  10
    0   0   0   0   0   4   0   0]
 [  0   0   1   0   1   0   4   0   0   0   1 192   0   0   0   0   0   2
    0   0   0   0   0   0   0   0]
 [  0   4   0   0   0   0   0   1   0   0   0   0 186   0   0   0   0   0
    0   0   0   3   0   0   0   0]
 [  0   2   0   0   0   0   0   1   0   0   0   0   0 192   2   0   0   2
    0   0   0   0   0   0   0   0]
 [  0   0   0   1   0   0   0   0   0   0   0   0   0   0 164   0   2   1
    0   0   1   0   6   0   0   0]
 [  0   1   0   3   2  11   0   1   0   0   0   0   0   0   1 195   1   0
    0   0   0   0   0   3   0   0]
 [  0   0   0   0   2   0   1   0   0   0   0   0   0   0   4   0 182   0
    1   0   0   0   2   0   0   0]
 [  0   8   0   2   0   0   0   0   0   0   1   0   1   4   0   0   0 179
    0   0   0   0   0   0   0   0]
 [  0   2   0   0   2   4   0   0   0   0   0   0   0   0   0   0   0   0
  178   0   0   0   0   0   0   0]
 [  0   0   0   1   1   1   0   0   0   0   0   0   0   0   0   1   0   1
    0 195   0   0   0   2   1   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0
    0   0 188   0   0   0   0   0]
 [  0   5   0   0   0   0   0   1   0   0   0   0   1   4   0   1   0   0
    0   0   0 172   0   0   0   0]
 [  0   1   0   0   0   0   1   0   0   0   0   0   1   0   0   0   0   0
    0   0   0   0 186   0   0   0]
 [  0   1   0   4   1   0   0   0   0   0   0   1   0   0   0   0   0   0
    0   0   0   0   0 195   0   0]
 [  2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   1   2   2   1   0 191   0]
 [  0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   3   0   1
    0   0   0   0   0   0   0 182]]
```

Here we go, the accuracy increased by approximately 10%. So, the radial basis function is basically a squared Euclidean distance between two features. As it is a non-linear function, there is higher chances that the data is non-linear as the accuracy of this model is 94% which pretty good. Let's focus on more non-linear models for the data.

Classification Algorithm 4: Neural Networks (Multilayer Perceptron)

Code 14:

```
#Applying MLP (multi-layer perceptron) algorithm
mlp_model = MLPClassifier(hidden_layer_sizes=(250, 300), max_iter=1000000, activation='logistic')
mlp_model.fit(X_train,y_train)
#Predicting our test data using the model created above
pred_mlp=mlp_model.predict(X_test)
#Computing the accuracy of the MLP model
print("Accuracy of MLP Model:", metrics.accuracy_score(y_true=y_test, y_pred=pred_mlp), "\n")
#Computing the confusion metrics of the MLP model
print(metrics.confusion_matrix(y_true=y_test, y_pred=pred_mlp))
```

Output 14:

```
Accuracy of MLP Model: 0.9702
[[167   0   0   0   0   0   0   0   0   0   1   1   0   0   0   0   0   0
    0   0   0   0   0   0   2   0]
 [  0 177   0   0   0   0   1   0   1   0   0   0   0   0   0   0   0   2
    0   0   0   2   0   0   0   0]
 [  0   0 195   0   0   0   1   0   0   0   0   0   0   0   1   0   0   0
    0   0   0   0   0   0   0   0]
 [  0   0   0 203   0   0   0   0   0   0   0   0   0   0   1   0   0   1
    1   0   0   0   0   0   0   0]
 [  0   0   1   0 183   0   5   0   0   0   0   1   0   0   0   0   0   0
    1   0   0   0   0   0   0   0]
 [  0   0   0   0   0 178   0   0   0   0   0   0   0   0   0   3   0   0
    0   0   0   1   0   0   0   0]
 [  0   0   0   0   1   1 182   1   0   0   0   0   1   0   0   0   1   0
    1   0   0   1   0   0   0   0]
 [  1   0   1   0   0   1   0 174   0   1   5   0   0   0   2   0   3   0
    0   0   0   0   0   1   0   0]
 [  0   0   0   1   0   0   0   0 174   5   0   0   0   0   0   0   0   0
    1   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   8 173   0   0   0   1   2   0   0   0
    0   0   1   0   0   1   0   0]
 [  0   0   0   0   0   0   0   4   0   0 202   0   0   0   0   0   0   2
    0   0   0   0   1   3   0   0]
 [  0   0   0   0   0   0   1   0   0   0   0 198   0   0   0   0   0   2
    0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   1   0   0   0   0   0 193   0   0   0   0   0
    0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   1   0   0   0   0   1 195   0   0   0   1
    0   0   1   0   0   0   0   0]
 [  0   0   1   1   0   0   0   0   0   0   0   0   0   0 169   0   2   1
    0   0   1   0   0   0   0   0]
 [  0   0   0   0   0   6   0   0   0   0   0   0   0   0   0 207   2   0
    0   0   0   0   1   0   2   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   3   0 185   0
    0   0   0   1   0   1   0   2]
 [  0   6   0   0   0   0   0   1   0   0   1   0   0   2   0   0   1 183
    0   0   0   0   1   0   0   0]
 [  0   1   0   0   1   0   0   0   0   1   0   1   0   0   0   0   0   0
  181   0   0   0   0   1   0   0]
 [  0   0   0   1   0   2   0   0   0   0   0   0   0   0   0   0   0   0
    0 197   0   0   0   1   2   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0
    0   0 188   0   0   0   0   0]
 [  0   2   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0
    0   0   0 179   2   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   1   0   0
    0   0   0   0 187   0   0   0]
 [  0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

```
    0    0    0    0    0 201    0    0]
 [  0    0    0    0    0    0    0    1    1    0    0    0    0    0    0    0    0    0
    0    0    1    0    1    0 195    0]
 [  0    0    0    0    1    0    0    0    0    0    0    0    0    0    0    0    1    0
    0    0    0    0    0    0    0 185]]
```

The accuracy for this model is very good 97%. The MLPClassifier implements a multi-layer perceptron neural network algorithm which trains using backpropagation. I've used 2 hidden layers as for image processing it is enough to use 2 layers. Due to its deep learning and ability to learn non-linear data, the accuracy of this model is quite good.

Classification Algorithm 5: KNN

Code 15:

```
#Applying KNN algorithm
knn_model=KNeighborsClassifier(n_neighbors=5, metric='manhattan', algorithm='auto', weights='distance', n_jobs=-1)
knn_model.fit(X_train,y_train)
#Predicting our test data using the model created above
pred_knn=knn_model.predict(X_test)
#Computing the accuracy of the KNN model
print("Accuracy of KNN Model:", metrics.accuracy_score(y_true=y_test, y_pred=pred_knn), "\n")
#Computing the confusion metrics of the KNN model
print(metrics.confusion_matrix(y_true=y_test, y_pred=pred_knn))
```

Output 15:

```
Accuracy of KNN Model: 0.954

[[171    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0]
 [  0 175    0    2    0    0    0    1    0    0    0    0    0    1    0    1    0    1
    0    0    0    2    0    0    0    0]
 [  0    0 186    0    2    0    4    0    0    0    0    0    0    2    0    2    0
    0    0    0    0    0    1    0    0]
 [  0    1    0 202    0    0    0    0    0    0    0    0    0    1    1    0    0    1
    0    0    0    0    0    0    0    0]
 [  0    0    4    0 176    1    4    0    0    0    1    1    1    0    0    0    0    0
    2    0    0    0    0    0    0    1]
 [  0    0    0    1    0 168    0    0    0    0    0    0    0    2    0    5    0    0
    1    3    0    1    1    0    0    0]
 [  0    0    0    4    2    0 179    1    0    0    0    0    0    0    1    0    0    0
    1    0    0    1    0    0    0    0]
 [  0    0    0    6    2    0    1 162    0    0    4    1    1    0    2    1    1    4
    0    0    2    0    0    2    0    0]
 [  0    0    0    0    0    0    0    0 173    7    1    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0]
 [  0    0    0    0    0    0    0    1    8 175    0    0    0    0    0    0    0    0
    0    0    2    0    0    0    0    0]
 [  0    1    0    0    2    0    0    8    0    0 194    0    0    0    0    0    0    4
    0    0    1    0    0    2    0    0]
 [  0    0    0    0    1    0    0    0    0    0    1 198    0    0    0    0    0    1
    0    0    0    0    0    0    0    0]
 [  0    3    0    0    0    0    1    0    0    0    0    0 187    0    0    0    0    0
    0    0    0    0    3    0    0    0]
 [  0    0    0    0    2    0    0    0    2    0    0    0    1    1 189    1    0    0    2
    0    0    0    1    0    0    0    0]
 [  0    0    0    1    0    0    0    0    0    0    0    0    0    0    0 171    0    1    0
    0    0    2    0    0    0    0    0]
 [  0    0    0    2    0    9    0    0    0    0    0    0    0    0    0    0 202    1    0
    0    1    0    0    0    3    0]
 [  0    0    0    0    1    0    0    0    0    0    0    0    0    0    5    0 184    0
    0    0    0    0    1    0    1    0]
 [  0    8    0    1    0    0    0    2    0    0    1    0    0    2    0    0    0 180
    1    0    0    0    0    0    0    0]
 [  0    0    0    0    1    0    0    0    0    0    0    0    0    0    0    0    0    1
  184    0    0    0    0    0    0    0]
 [  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0  200    0    0    0    0    3    0]
 [  1    0    0    0    0    0    0    2    0    0    0    0    0    0    0    0    0    0
    0    0  186    0    0    0    0    0]
 [  0    4    0    0    0    0    0    0    0    0    0    0    0    1    0    1    0    0
    0    0    0  177    1    0    0    0]
 [  0    0    0    0    0    0    0    0    0    0    0    0    1    0    1    0    0    1
    0    0    1    0  185    0    0    0]
 [  0    0    0    1    1    0    0    0    0    1    4    1    0    0    0    0    0    0
```

```
    0   0   1   0   0 193   0   0]
 [  2   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   1   1   1   1   0 192   0]
 [  0   0   0   0   1   0   0   0   0   0   0   1   0   0   0   0   3   0
    0   1   0   0   0   0   0 181]]
```

As KNN works good for non-linear data, the accuracy of this model has turned out to be quite good (95%). The similarity measure used here is manhattan distance as the dimensionality of the data is high.

Classification Algorithm 6: Decision Trees

Code 16:

```
#Applying Decision Tree algorithm
dec_model = DecisionTreeClassifier()
dec_model.fit(X_train,y_train)
#Predicting our test data using the model created above
pred_dec=dec_model.predict(X_test)
#Computing the accuracy of the Decision Tree model
print("Accuracy of Decision Tree Model:", metrics.accuracy_score(y_true=y_test, y_pred=pred_dec), "\n")
#Computing the confusion metrics of the Decision Tree model
print(metrics.confusion_matrix(y_true=y_test, y_pred=pred_dec))
```

Output 16:

```
Accuracy of Decision Tree Model: 0.8678
```

```
[[164   0   0   0   0   0   1   0   0   1   0   2   1   1   0   0   0   0
    0   1   0   0   0   0   0   0]
 [  0 154   0   2   0   0   2   1   1   1   1   2   0   0   1   1   0   2
    9   1   0   2   1   1   1   0]
 [  0   0 176   0   5   1   4   1   0   0   0   4   0   0   1   0   2   0
    1   1   1   0   0   0   0   0]
 [  0   4   0 177   0   1   2   3   0   1   0   0   0   3   4   1   3   3
    3   1   0   0   0   0   0   0]
 [  1   2   4   0 156   2   5   1   0   1   4   1   0   0   0   0   4   1
    1   3   1   0   1   2   0   1]
 [  0   2   0   0   0 157   2   1   1   2   0   0   0   0   0   9   0   0
    2   3   0   0   3   0   0   0]
 [  0   3   9   2   3   0 152   0   0   1   2   1   0   0   3   1   4   1
    2   4   1   0   0   0   0   0]
 [  0   2   0   2   0   1   3 144   0   0   4   2   4   1   2   1   0   7
    2   5   2   0   0   2   4   1]
 [  0   5   0   2   0   1   0   0 161   5   0   0   0   0   0   2   2   0
    0   0   0   0   0   2   0   1]
 [  0   1   0   0   0   1   0   2   9 162   0   1   2   0   1   1   0   0
    1   0   1   0   0   1   2   1]
 [  3   0   0   1   0   0   3   7   0   1 170   1   2   1   1   0   0   3
    0   0   0   0   1  18   0   0]
 [  0   0   0   0   1   1   4   0   0   0   0 189   0   0   0   1   0   1
    3   1   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   2   0   0   0   0 176   2   0   2   0   2
    2   0   3   0   1   2   2   0]
 [  0   0   0   0   0   0   0   1   1   1   0   0   3 178   2   0   1   6
    0   0   2   2   2   0   0   0]
 [  0   1   0   5   1   0   5   4   0   0   1   0   0   1 144   1   4   0
    2   0   0   3   2   0   0   1]
 [  1   2   0   2   0  11   0   0   2   0   1   0   0   0   1 191   2   0
    1   0   0   1   1   1   1   0]
 [  0   0   3   1   3   0   0   3   0   0   1   2   2   0   5   2 164   0
    3   1   0   1   0   1   0   0]
 [  0   6   0   7   2   0   1   1   0   0   5   5   0   4   2   2   1 151
    3   0   0   0   2   3   0   0]
 [  0   5   1   0   6   3   2   3   1   0   1   0   0   0   0   0   1   0
  158   0   1   0   0   0   2   2]
 [  0   0   2   0   1   3   1   0   1   1   2   0   1   0   0   0   1   0
    2 178   0   1   3   1   5   0]
 [  1   0   0   0   1   0   0   1   0   0   0   0   2   0   2   1   0   0
    1   0 178   0   0   1   1   0]
 [  0   5   0   0   1   1   1   2   0   0   0   0   2   4   3   0   1   0
    0   5   2 153   3   0   1   0]
 [  1   0   0   0   0   0   0   0   0   0   1   0   3   0   1   0   1   2
    0   0   1   1 176   0   2   0]
 [  1   2   0   0   4   0   2   1   0   0   4   2   0   0   0   0   0   2
    1   0   0   0   0 183   0   0]
 [  0   1   1   1   0   1   1   0   0   0   0   0   1   3   1   3   0   0
    0   3   0   3   0   0 179   1]
 [  0   0   0   1   2   0   0   0   1   2   0   0   0   0   0   0   4   0
    8   1   0   0   0   0   0 168]]
```

The accuracy of decision tree is not quite good as other non-linear algorithms (86%). The reason of the decrease in accuracy might be due to the high number of classes (26). Decision trees works best if the number of classes are less.

Below are ROC scores for all the algorithms

Code 17:

```
y_score_NB = gaussianNB_model.predict_proba(X_test)
print('roc_auc_score for NaiveBayes: ', roc_auc_score(y_test, y_score_NB, multi_class='ovo'))
y_score_linear = linear_model.predict_proba(X_test)
print('roc_auc_score for SVMLinear: ', roc_auc_score(y_test, y_score_linear, multi_class='ovo'))
y_score_svm = nonLinear_model.predict_proba(X_test)
print('roc_auc_score for SVMnonLinear: ', roc_auc_score(y_test, y_score_svm, multi_class='ovo'))
y_score_mlp = mlp_model.predict_proba(X_test)
print('roc_auc_score for MLP: ', roc_auc_score(y_test, y_score_mlp, multi_class='ovo'))
y_score_knn = knn_model.predict_proba(X_test)
print('roc_auc_score for KNN: ', roc_auc_score(y_test, y_score_knn, multi_class='ovo'))
y_score_dec = dec_model.predict_proba(X_test)
print('roc_auc_score for DecisionTree: ', roc_auc_score(y_test, y_score_dec, multi_class='ovo'))
```

Output 17:

```
roc_auc_score for NaiveBayes:  0.9587551817154645
roc_auc_score for SVMLinear:  0.9939681849615405
roc_auc_score for SVMnonLinear:  0.9990258526847193
roc_auc_score for MLP:  0.9996976906522625
roc_auc_score for KNN:  0.9945427790073929
roc_auc_score for DecisionTree:  0.9326192021158256
```

So, these scores tell us the classifier output quality. We can see that the best quality classifiers are SVM(rbf) and MLP.