Pen-Based Recognition of Handwritten Digits

<u>**Team Members:**</u>

Divyansh Singh (18ucs127)

Lakshay Bhagtani (18ucs132)

Raghav Chugh (18ucs195)

Harshit Malhotra (18ucs207)

Table of Contents:

Introduction:

Citation:

E. Alpaydin, Fevzi. Alimoglu

Department of Computer Engineering

Bogazici University, 80815 Istanbul Turkey

alpaydin@boun.edu.tr, July 1998

Description:

A digit database was created by collecting 250 samples from 44 writers. The samples written by 30 writers are used for training, cross-validation and writer dependent testing, and the digits written by the other 14 are used for writer independent testing. This database is also available in the UNIPEN format.

A WACOM PL-100V pressure sensitive tablet was used with an integrated LCD display and a cordless stylus. The input and display areas are located in the same place. Attached to the serial port of an Intel 486 based PC allowed to collect handwriting samples. The tablet sends x and y tablet coordinates and pressure level values of the pen at fixed time intervals (sampling rate) of 100 milliseconds.

So, the input vector size is 2*T, two times the number of points resampled. Spatial resampling to T=8,12,16 points was considered in the experiments and it was found that T=8 gave the best trade-off between accuracy and complexity.

Features:

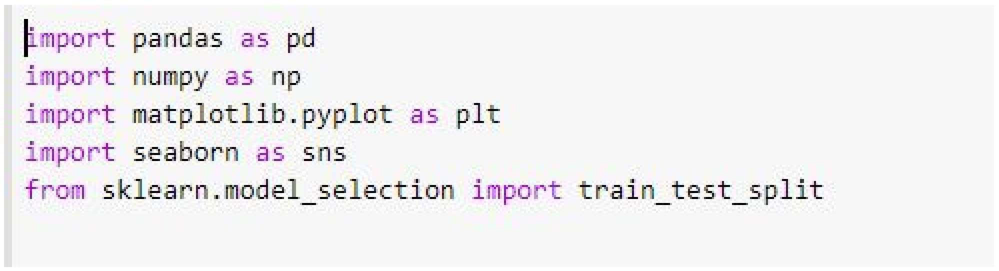
16 continuous variables each between the range 0-100, two times the number of points resampled which are 8 that measure the x and y coordinates and pressure values which are measured by the device which then reports these values to the computer.

Target Variable:

The Target Variable is a categorical variable which are digits between 0-9 that represent 10 classes/categories. Hence it is a classification Task and not a regression one.

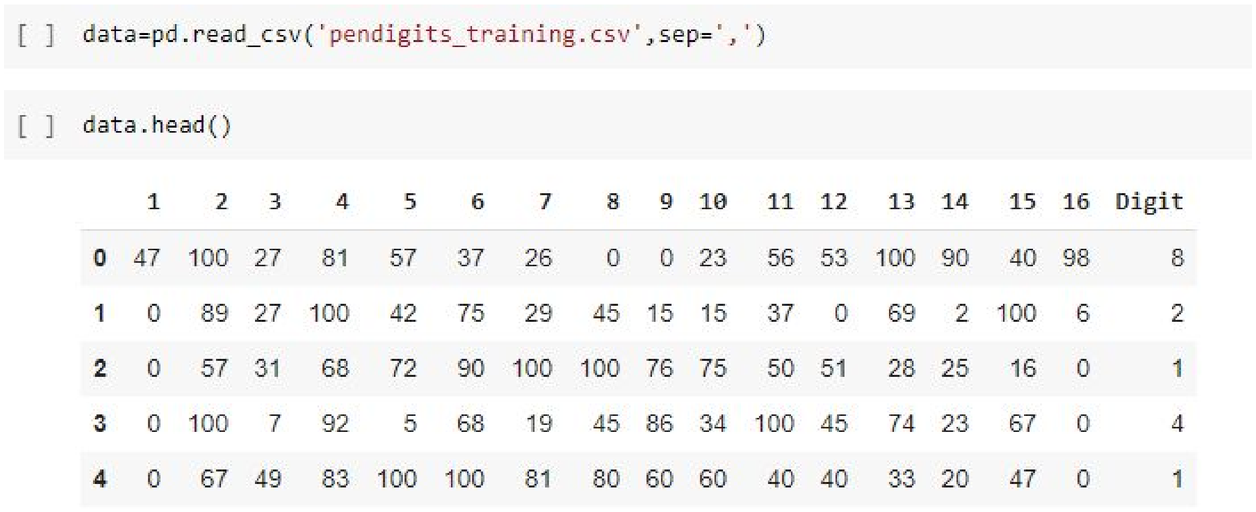
Importing and Exploring the Data:

In order to perform descriptive data analytics, we first need to import the dataset from our system and a few libraries that were necessary for the implementation.



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
```

Importing the dataset as csv file and printing the dataset to have a look at how our dataset looks.data.head() function by default prints top 5 instances of the dataset using pandas library.

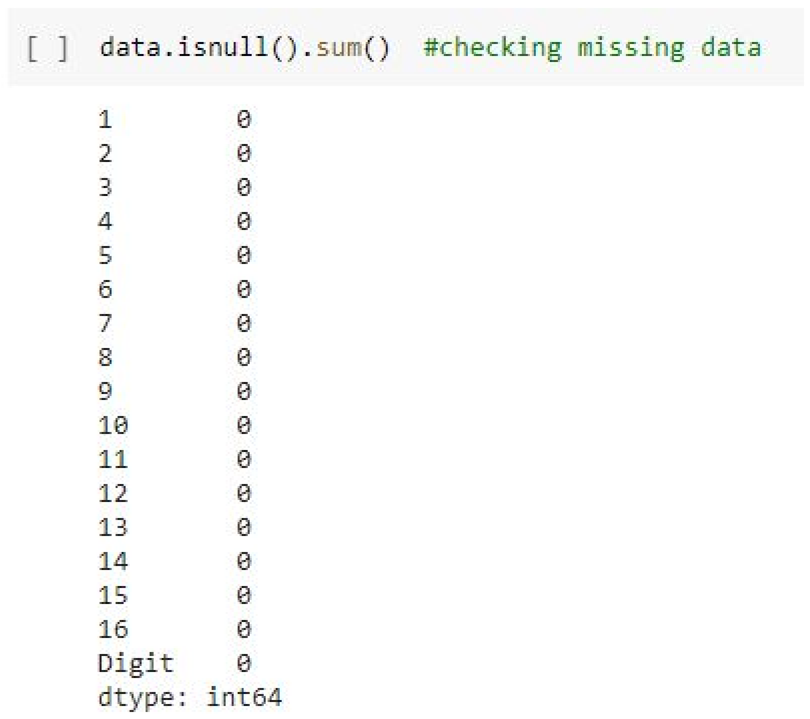


```
[ ]  data=pd.read_csv('pendigits_training.csv',sep=',')
```

```
[ ]  data.head()
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	Digit
0	47	100	27	81	57	37	26	0	0	23	56	53	100	90	40	98	8
1	0	89	27	100	42	75	29	45	15	15	37	0	69	2	100	6	2
2	0	57	31	68	72	90	100	100	76	75	50	51	28	25	16	0	1
3	0	100	7	92	5	68	19	45	86	34	100	45	74	23	67	0	4
4	0	67	49	83	100	100	81	80	60	60	40	40	33	20	47	0	1

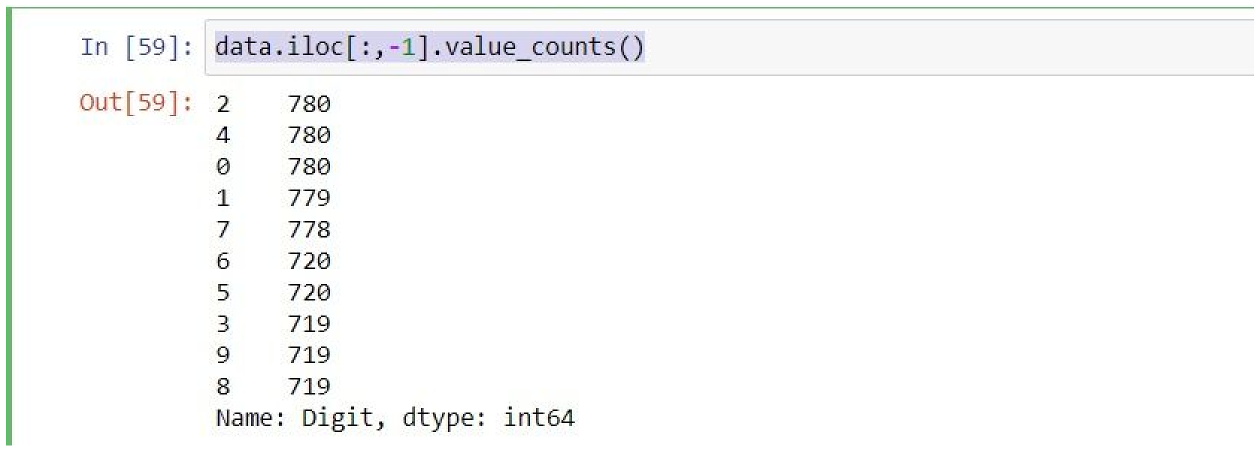
Checking for null or missing values in the dataset.



```
[ ]  data.isnull().sum()  #checking missing data

1        0
2        0
3        0
4        0
5        0
6        0
7        0
8        0
9        0
10       0
11       0
12       0
13       0
14       0
15       0
16       0
Digit    0
dtype: int64
```

Looks like there are no missing or null values in our dataset, so we proceed further.

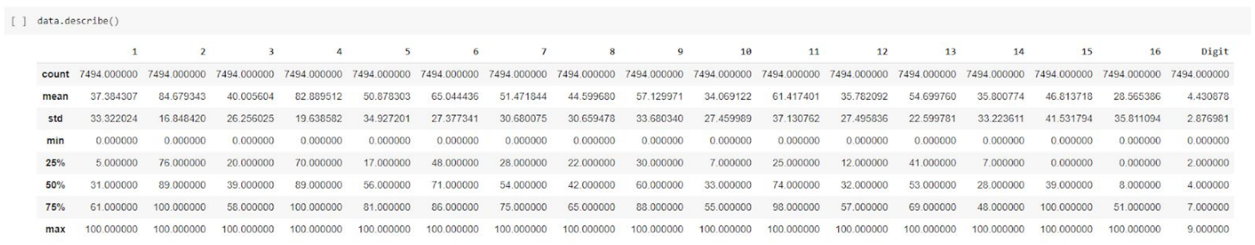
Checking the class distribution of the target variable



```
In [59]: data.iloc[:,-1].value_counts()

Out[59]: 2    780
         4    780
         0    780
         1    779
         7    778
         6    720
         5    720
         3    719
         9    719
         8    719
         Name: Digit, dtype: int64
```

From this, we infer that the data set is balanced as there are almost equal samples from each class.

Checking the numerical statistics of the features. (Zoom in for a clearer image)



```
[ ]  data.describe()
```

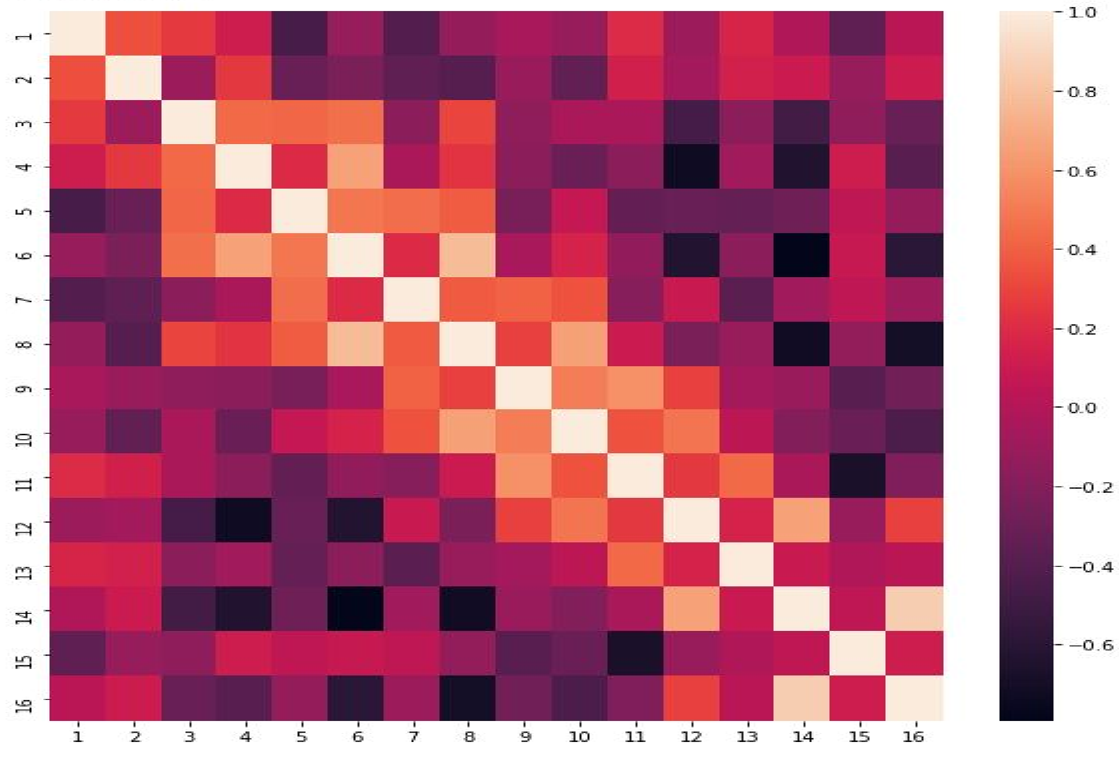
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	Digit
count	7494.000000	7494.000000	7494.000000	7494.000000	7494.000000	7494.000000	7494.000000	7494.000000	7494.000000	7494.000000	7494.000000	7494.000000	7494.000000	7494.000000	7494.000000	7494.000000	7494.000000
mean	37.384307	84.679343	40.005604	82.889512	50.878303	65.044436	51.471844	44.599680	57.129971	34.069122	61.417401	35.782092	54.699760	35.800774	46.813718	28.565386	4.430878
std	33.322024	16.848420	26.256025	19.638582	34.927201	27.377341	30.680075	30.659478	33.680340	27.459989	37.130762	27.495836	22.599781	33.223611	41.531794	35.811094	2.876981
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	5.000000	76.000000	20.000000	70.000000	17.000000	48.000000	28.000000	22.000000	30.000000	7.000000	25.000000	12.000000	41.000000	7.000000	0.000000	0.000000	2.000000
50%	31.000000	89.000000	39.000000	89.000000	56.000000	71.000000	54.000000	42.000000	60.000000	33.000000	74.000000	32.000000	53.000000	28.000000	39.000000	8.000000	4.000000
75%	61.000000	100.000000	58.000000	100.000000	81.000000	86.000000	75.000000	65.000000	88.000000	55.000000	98.000000	57.000000	69.000000	48.000000	100.000000	51.000000	7.000000
max	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	9.000000

Hence there are a total 7494 samples in the dataset.

Printing the correlation matrix :

```
[ ]  #plotting the correlation matrix with each column
     plt.figure(figsize=(10,10))
     sns.heatmap(data_normalized.corr())
```

```
<AxesSubplot:>
```



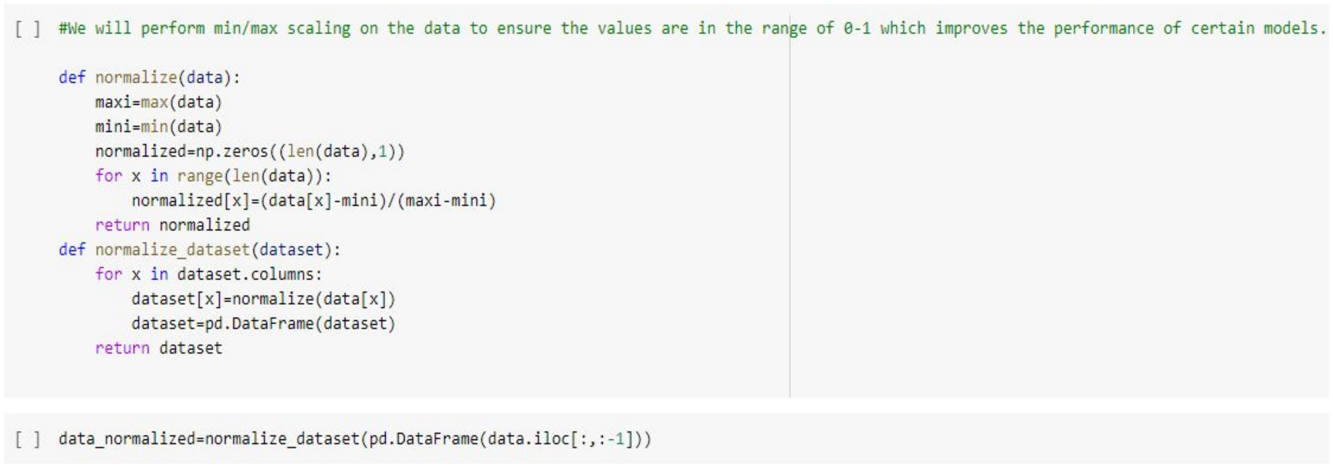
From this, we infer that the feature 14 has the most correlation with the target variable.

Preprocessing and Dividing the Data:

Normalization

Since all the features were continuous variables, we performed min-max normalization on the dataset as machine learning models perform better on normalized data.

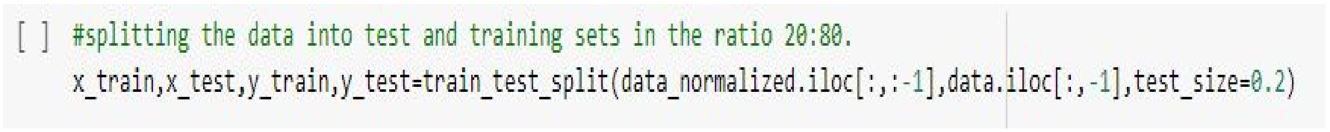
$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$



```
[ ]  #We will perform min/max scaling on the data to ensure the values are in the range of 0-1 which improves the performance of certain models.

     def normalize(data):
         maxi=max(data)
         mini=min(data)
         normalized=np.zeros((len(data),1))
         for x in range(len(data)):
             normalized[x]=(data[x]-mini)/(maxi-mini)
         return normalized
     def normalize_dataset(dataset):
         for x in dataset.columns:
             dataset[x]=normalize(data[x])
             dataset=pd.DataFrame(dataset)
         return dataset
```

```
[ ]  data_normalized=normalize_dataset(pd.DataFrame(data.iloc[:,:-1]))
```

We have divided the data into 80:20 ratio for training and testing the machine learning models.



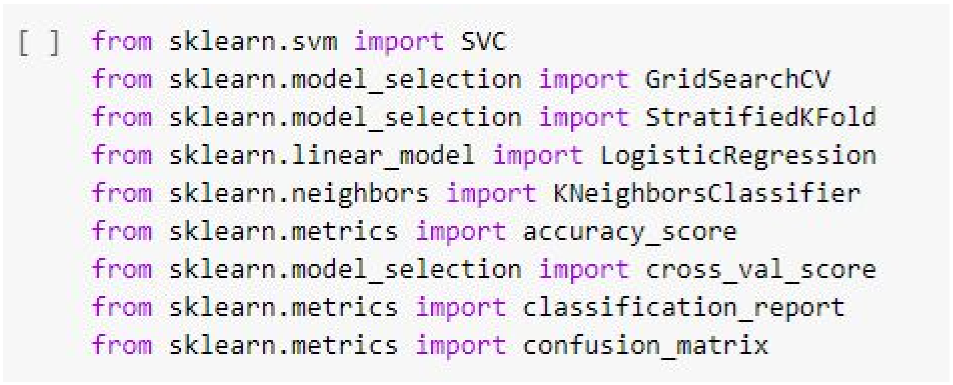
```
[ ]  #splitting the data into test and training sets in the ratio 20:80.
     x_train,x_test,y_train,y_test=train_test_split(data_normalized.iloc[:,:-1],data.iloc[:,-1],test_size=0.2)
```

Modelling and Classification:

We have used the following three machine learning classifiers:

1. Support Vector Machines (SVM)
2. Logistic Regression
3. K-nearest neighbours (KNN)

Importing the necessary libraries and functions

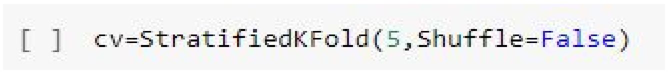


```
[ ]  from sklearn.svm import SVC
     from sklearn.model_selection import GridSearchCV
     from sklearn.model_selection import StratifiedKFold
     from sklearn.linear_model import LogisticRegression
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.metrics import accuracy_score
     from sklearn.model_selection import cross_val_score
     from sklearn.metrics import classification_report
     from sklearn.metrics import confusion_matrix
```

Cross-Validation

We have used a stratified K-fold cross validation algorithm. Stratified K-Folds cross-validation. Provides train/test indices to split data in train/test sets. This cross-validation object is a variation of KFold that returns stratified folds. The folds are made by preserving the percentage of samples for each class.

We have used cross-validation for tuning the hyperparameters in SVM and finding the optimal K value in KNN.



```
[ ]  cv=StratifiedKFold(5,Shuffle=False)
```

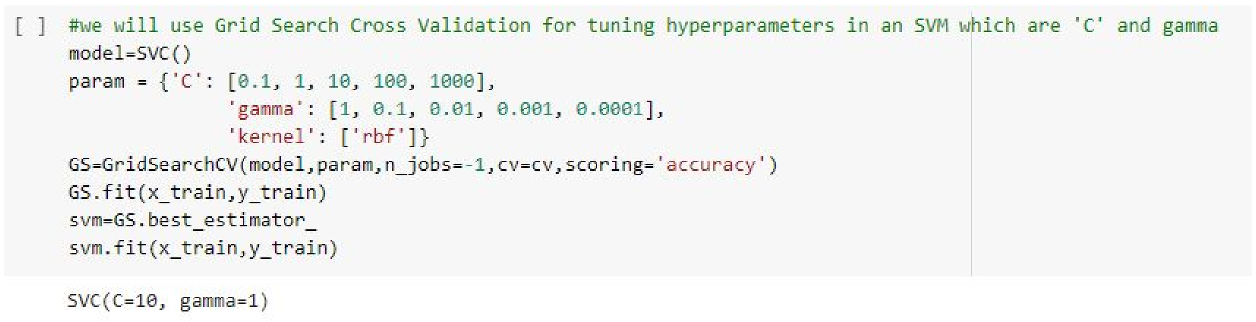
The value of K in stratified K-fold is 5 to ensure accurate results.

Training and Testing

We have trained the model on the training data and tested the model on testing data. The performance metrics used to compare the models are accuracy, classification report, and confusion matrix.

Support Vector Machines

To find the optimal values of hyperparameters gamma and C, we have used grid search CV with stratified K-fold. Grid Search CV performs cross validation by testing out all the combinations of parameters given to it and selects the model with the best metric that we have considered here to be accuracy.



```
[ ]  #we will use Grid Search Cross Validation for tuning hyperparameters in an SVM which are 'C' and gamma
     model=SVC()
     param = {'C': [0.1, 1, 10, 100, 1000],
                   'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                   'kernel': ['rbf']}
     GS=GridSearchCV(model,param,n_jobs=-1,cv=cv,scoring='accuracy')
     GS.fit(x_train,y_train)
     svm=GS.best_estimator_
     svm.fit(x_train,y_train)

     SVC(C=10, gamma=1)
```

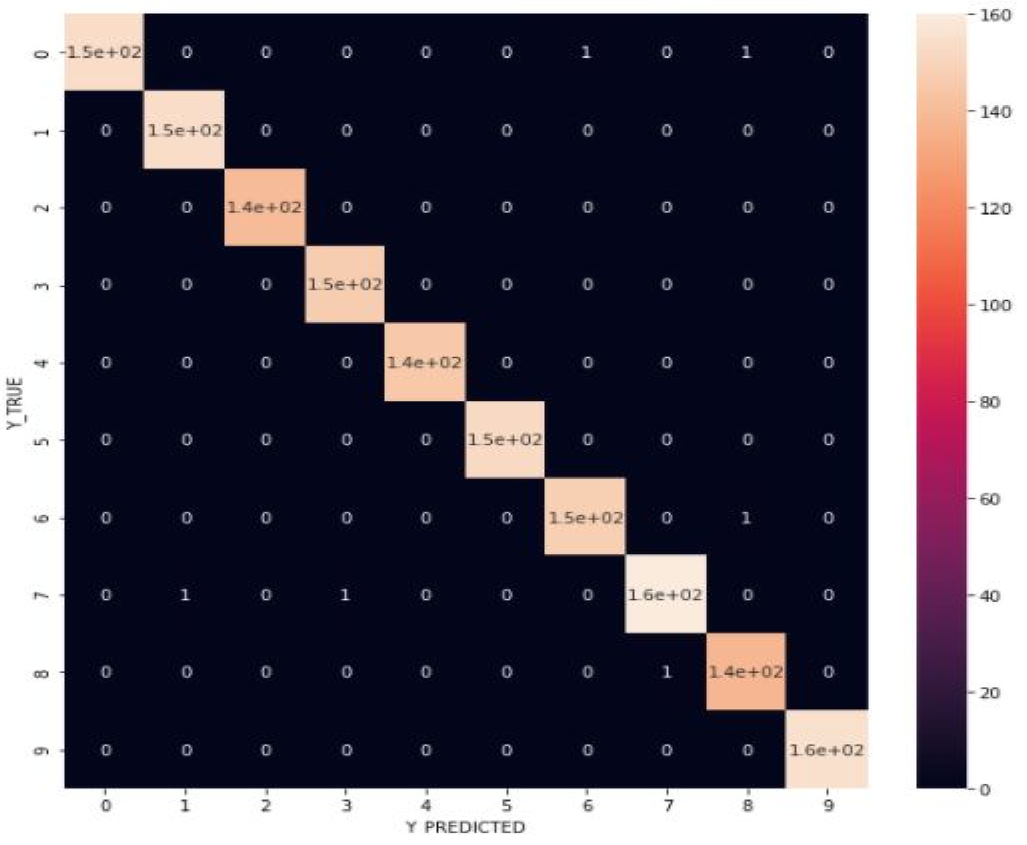
The optimal value determined by the algorithm is C = 10 and gamma = 1.

Performance on Test Data and Confusion Matrix

```
[ ]  print('The accuracy scores are '+ str(accuracy_score(y_test,svm.predict(x_test))))
     print('The Classification Report is: ')
     print(classification_report(y_test,svm.predict(x_test)))
     #Confusion Matrix of SVM
     import seaborn as sns
     plt.figure(figsize=(10,10))
     cf_svm=confusion_matrix(y_test,svm.predict(x_test))
     sns.heatmap(cf_svm, annot=True)
     plt.xlabel("Y_PREDICTED")
     plt.ylabel("Y_TRUE")
```

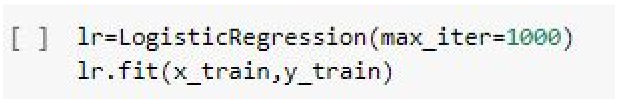
```
The accuracy scores are 0.9959973315543695
The Classification Report is:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       156
           1       0.99      1.00      1.00       154
           2       1.00      1.00      1.00       140
           3       0.99      1.00      1.00       147
           4       1.00      1.00      1.00       145
           5       1.00      1.00      1.00       152
           6       0.99      0.99      0.99       149
           7       0.99      0.99      0.99       162
           8       0.99      0.99      0.99       139
           9       1.00      1.00      1.00       155

    accuracy                           1.00      1499
   macro avg       1.00      1.00      1.00      1499
weighted avg       1.00      1.00      1.00      1499
```

```
Text(69.0, 0.5, 'Y_TRUE')
```



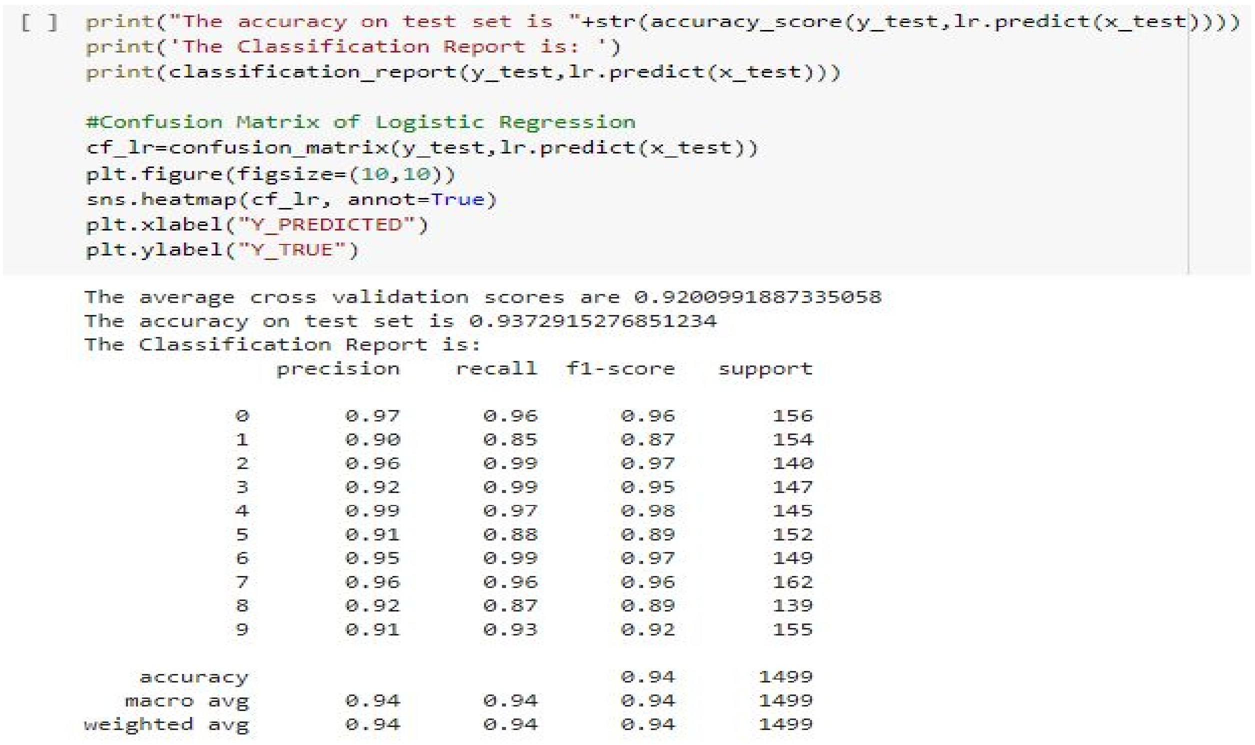
Logistic Regression

We performed logistic regression with L2 regularization on the training data.



```
[ ]  lr=LogisticRegression(max_iter=1000)
     lr.fit(x_train,y_train)
```

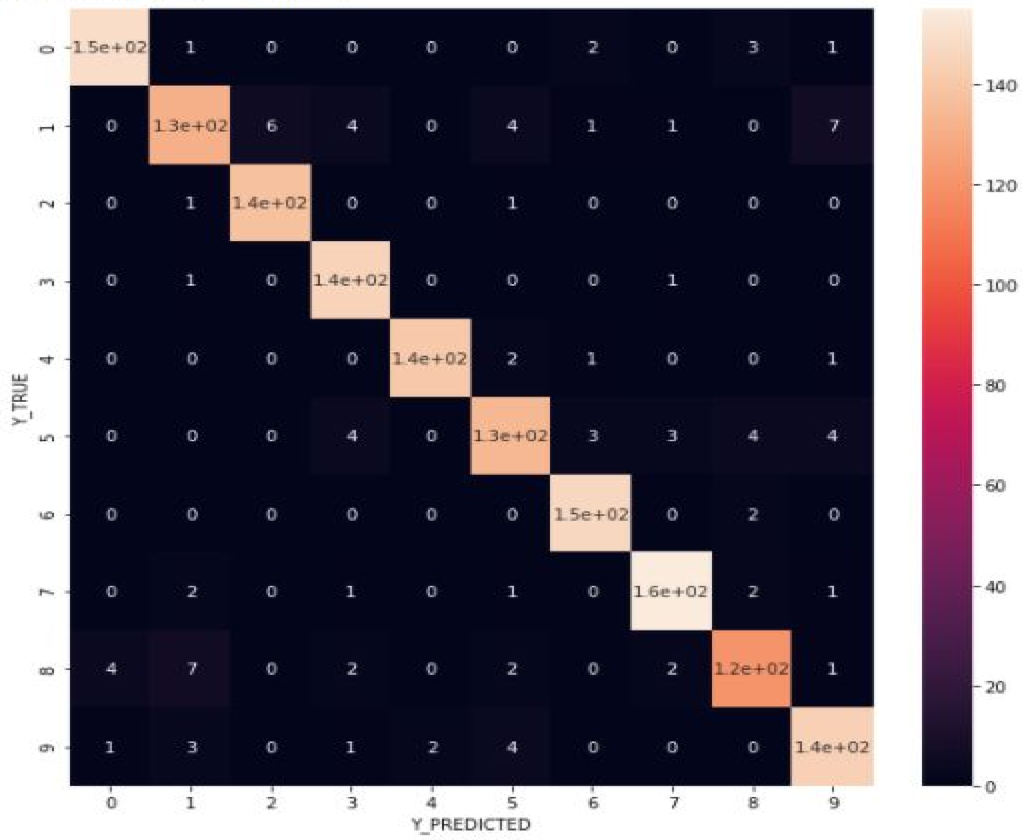
Performance on Test Data and Confusion Matrix



```
[ ]  print("The accuracy on test set is "+str(accuracy_score(y_test,lr.predict(x_test))))
     print('The Classification Report is: ')
     print(classification_report(y_test,lr.predict(x_test)))

     #Confusion Matrix of Logistic Regression
     cf_lr=confusion_matrix(y_test,lr.predict(x_test))
     plt.figure(figsize=(10,10))
     sns.heatmap(cf_lr, annot=True)
     plt.xlabel("Y_PREDICTED")
     plt.ylabel("Y_TRUE")
```

```
The average cross validation scores are 0.9200991887335058
The accuracy on test set is 0.9372915276851234
The Classification Report is:
              precision    recall  f1-score   support

           0       0.97      0.96      0.96       156
           1       0.90      0.85      0.87       154
           2       0.96      0.99      0.97       140
           3       0.92      0.99      0.95       147
           4       0.99      0.97      0.98       145
           5       0.91      0.88      0.89       152
           6       0.95      0.99      0.97       149
           7       0.96      0.96      0.96       162
           8       0.92      0.87      0.89       139
           9       0.91      0.93      0.92       155

    accuracy                           0.94      1499
   macro avg       0.94      0.94      0.94      1499
weighted avg       0.94      0.94      0.94      1499
```

```
Text(69.0, 0.5, 'Y_TRUE')
```

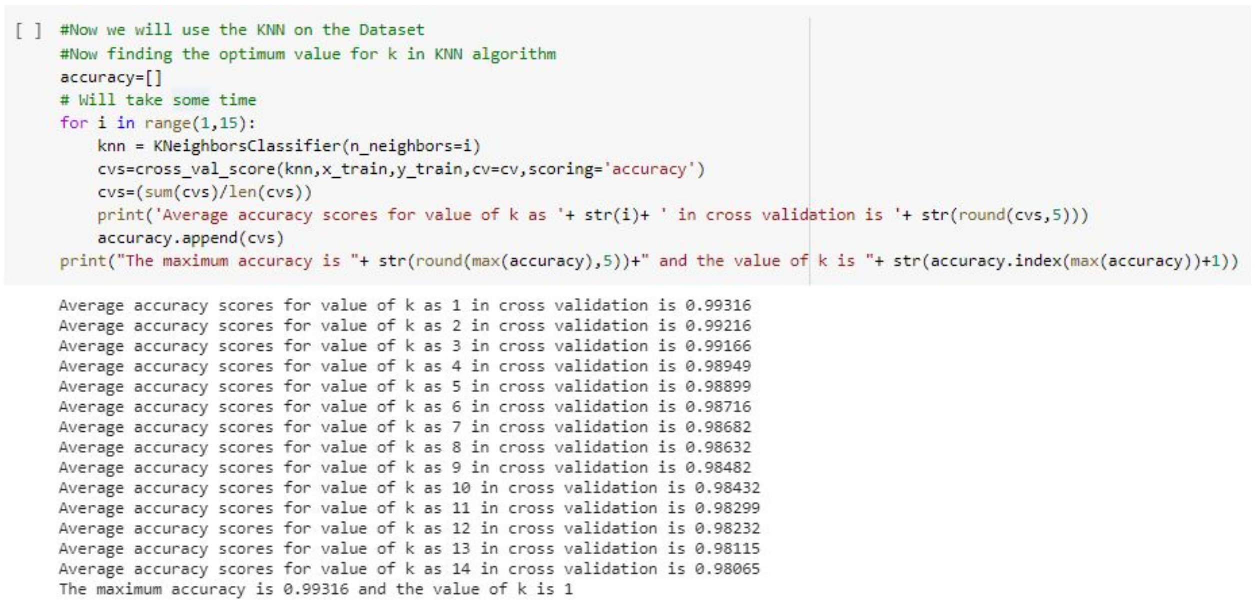


We also performed some feature engineering and introduced 32 more exponential variables i.e. each feature was squared and cubed and used as a new one. This did not improve the performance of Logistic Regression.

**We also tried different values of alpha but no significant cha
nges in performance were noted.**

K Nearest Neighbours

To determine the optimal K value for using KNN algorithm, we used stratified K fold cross validation and selected the value of K with the most average accuracy.

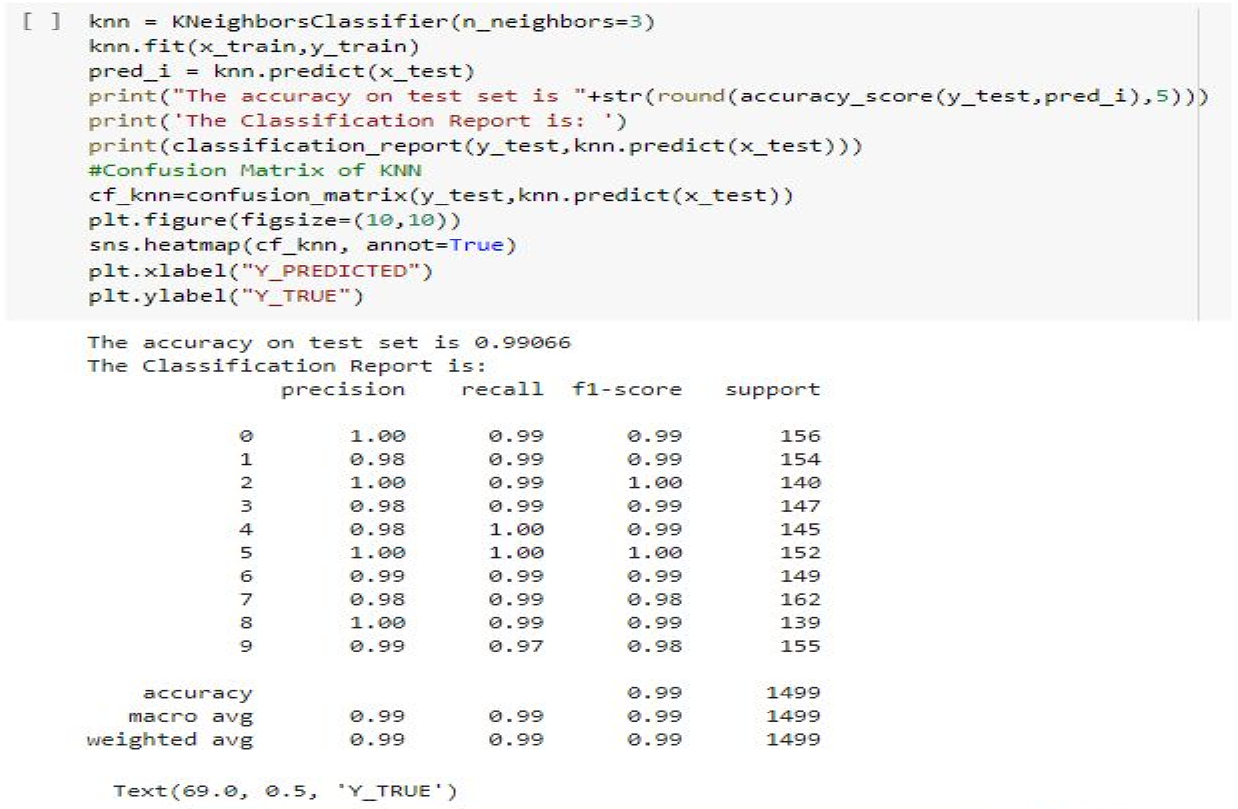


```
[ ] #Now we will use the KNN on the Dataset
    #Now finding the optimum value for k in KNN algorithm
    accuracy=[]
    # Will take some time
    for i in range(1,15):
        knn = KNeighborsClassifier(n_neighbors=i)
        cvs=cross_val_score(knn,x_train,y_train,cv=cv,scoring='accuracy')
        cvs=(sum(cvs)/len(cvs))
        print('Average accuracy scores for value of k as '+ str(i)+ ' in cross validation is '+ str(round(cvs,5)))
        accuracy.append(cvs)
    print("The maximum accuracy is "+ str(round(max(accuracy),5))+" and the value of k is "+ str(accuracy.index(max(accuracy))+1))
```

```
Average accuracy scores for value of k as 1 in cross validation is 0.99316
Average accuracy scores for value of k as 2 in cross validation is 0.99216
Average accuracy scores for value of k as 3 in cross validation is 0.99166
Average accuracy scores for value of k as 4 in cross validation is 0.98949
Average accuracy scores for value of k as 5 in cross validation is 0.98899
Average accuracy scores for value of k as 6 in cross validation is 0.98716
Average accuracy scores for value of k as 7 in cross validation is 0.98682
Average accuracy scores for value of k as 8 in cross validation is 0.98632
Average accuracy scores for value of k as 9 in cross validation is 0.98482
Average accuracy scores for value of k as 10 in cross validation is 0.98432
Average accuracy scores for value of k as 11 in cross validation is 0.98299
Average accuracy scores for value of k as 12 in cross validation is 0.98232
Average accuracy scores for value of k as 13 in cross validation is 0.98115
Average accuracy scores for value of k as 14 in cross validation is 0.98065
The maximum accuracy is 0.99316 and the value of k is 1
```

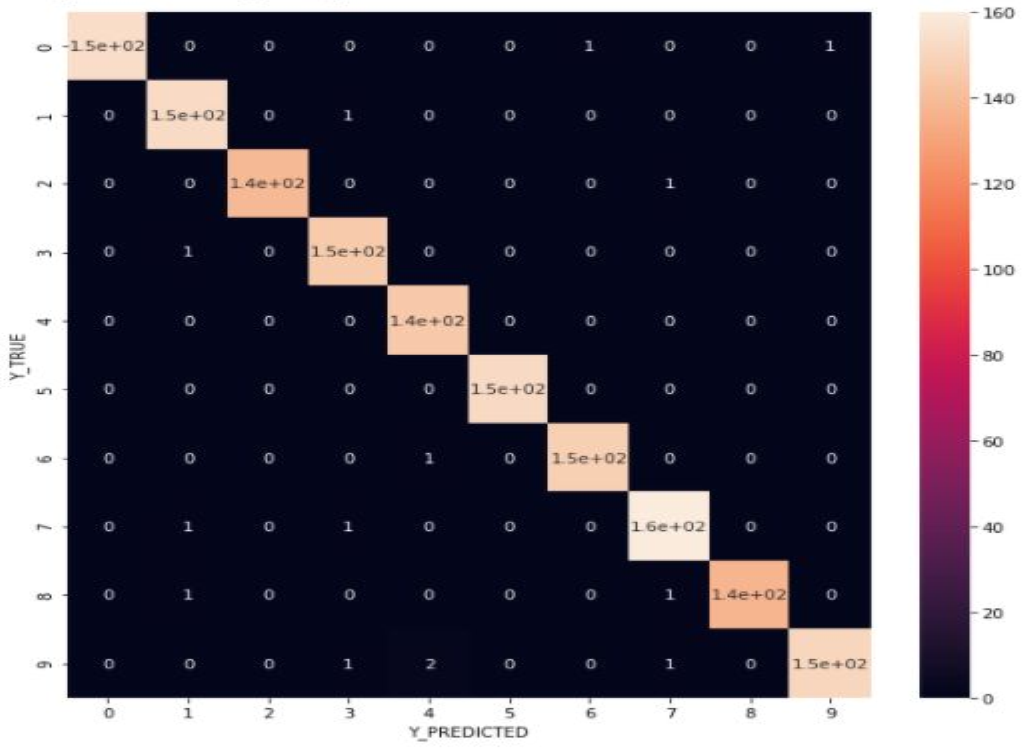
We determined the values K = 1 and K = 3 gave the best performance in cross validation. So we selected K = 1 for fitting our model.

Performance on Test Data and Confusion Matrix



```
[ ]  knn = KNeighborsClassifier(n_neighbors=3)
     knn.fit(x_train,y_train)
     pred_i = knn.predict(x_test)
     print("The accuracy on test set is "+str(round(accuracy_score(y_test,pred_i),5)))
     print('The Classification Report is: ')
     print(classification_report(y_test,knn.predict(x_test)))
     #Confusion Matrix of KNN
     cf_knn=confusion_matrix(y_test,knn.predict(x_test))
     plt.figure(figsize=(10,10))
     sns.heatmap(cf_knn, annot=True)
     plt.xlabel("Y_PREDICTED")
     plt.ylabel("Y_TRUE")
```

```
The accuracy on test set is 0.99066
The Classification Report is:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       156
           1       0.98      0.99      0.99       154
           2       1.00      0.99      1.00       140
           3       0.98      0.99      0.99       147
           4       0.98      1.00      0.99       145
           5       1.00      1.00      1.00       152
           6       0.99      0.99      0.99       149
           7       0.98      0.99      0.98       162
           8       1.00      0.99      0.99       139
           9       0.99      0.97      0.98       155

    accuracy                           0.99      1499
   macro avg       0.99      0.99      0.99      1499
weighted avg       0.99      0.99      0.99      1499

 Text(69.0, 0.5, 'Y_TRUE')
```



Conclusion:

- **From comparing the above machine learning classification models, we infer that SVM with c = 10 and gamma = 1 as the hyperparameters perform the best on the testing data.**
  F1 score ≈ 1.0 and accuracy = 0.996

- **KNN performs slightly worse than SVM with K = 1 and F1 score = 0.99 and accuracy = 0.991.**

- **Logistic Regression was the poorest of all with F1 score = 0.94 and accuracy = 0.937.**