Text Classification - Project

Introduction

The letters dataset will be the focus of this study. The primary purpose of this project is to create a K Nearest Neighbor and Neutral Network model that can correctly predict numbers. With this model, we hope to assist the school in determining whether a writing exam may be used to determine which students may need to work on their motor skills at an early age. We are given a dataset including data on numbers written by students, and we must create a model to correctly forecast the numbers.

Analysis

We will first assess the dataset to see whether there are any null values, and then we will prepare the data for modeling.

```
In [39]: df
Out[39]:
```

	label	pixel43	pixel44	pixel92	pixel124	pixel125	pixel126	pixel127	pixel128	pixel129	...	pixel329	pixel351	pixel410	pixel411	pixel412	pixel413
0	1	0	0	0	0	0	0	0	0	0	...	0	254	0	0	0	0
1	0	0	0	0	137	137	192	86	72	1	...	254	0	0	75	254	254
2	1	0	0	0	3	141	139	3	0	0	...	0	184	0	0	0	0
3	4	0	0	0	0	0	0	0	0	0	...	0	0	94	255	69	0
4	0	0	0	0	155	254	254	254	157	30	...	253	0	0	0	223	253
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
41995	2	0	0	1	248	253	176	43	0	0	...	0	0	0	0	0	0
41996	0	0	0	0	0	0	0	0	0	128	...	0	0	0	0	255	255
41997	2	0	0	0	255	255	191	0	0	0	...	0	0	0	0	0	0
41998	2	0	0	0	255	128	0	0	0	0	...	0	255	0	0	0	0
41999	2	0	0	227	253	229	133	19	0	0	...	0	0	253	160	1	0

42000 rows × 46 columns

As we can see, our dataset has **42000** rows and **46** columns. We have one label column, and the rest are image pixel columns. We try to get some more insights from these columns.

```
In [40]: df.describe()
```

Out[40]:

	label	pixel43	pixel44	pixel92	pixel124	pixel125	pixel126	pixel127	pixel128	pixel129	...	42(
count	42000.000000	42000.000000	42000.000000	42000.000000	42000.000000	42000.000000	42000.000000	42000.000000	42000.000000	42000.000000	...	
mean	4.456643	0.171357	0.164476	1.192833	28.043952	36.084976	42.713952	46.092310	44.542452	38.948524	...	
std	2.887730	5.726352	5.515774	14.692403	70.505431	78.631145	84.390533	87.287033	85.740313	81.223946	...	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	
25%	2.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	
50%	4.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	
75%	7.000000	0.000000	0.000000	0.000000	0.000000	0.000000	10.000000	29.000000	21.000000	0.000000	...	
max	9.000000	255.000000	255.000000	255.000000	255.000000	255.000000	255.000000	255.000000	255.000000	255.000000	...	2

According to the above analysis, the maximum value in the label column is **9** and the minimum value is **0**. This is due to the fact that we only collect data for numerical purposes. When we look at the pixel's columns, we can see that the maximum pixel value is **255**, which is logical, and the minimum pixel value is **0**. Images are stored in a computer as a matrix of numbers, which are referred to as pixel values. The intensity of each pixel is represented by these pixel values. The numbers **0** and **255** represent black and white, respectively.

```
In [41]: df.isnull().sum()
```

Out[41]:

```
label       0
pixel43     0
pixel44     0
pixel92     0
pixel124    0
pixel125    0
pixel126    0
pixel127    0
pixel128    0
pixel129    0
pixel130    0
pixel131    0
pixel132    0
pixel133    0
pixel134    0
pixel135    0
pixel136    0
pixel137    0
pixel138    0
pixel146    0
pixel147    0
pixel148    0
pixel149    0
pixel150    0
pixel151    0
pixel152    0
```

There are no null values in our dataset and our dataset is clean and ready for further modeling purposes.

```
[6]  y.unique()

     array([1, 0, 4, 7, 3, 5, 8, 9, 2, 6])
```

As we see above, we have **0-9** unique values in our target variable.

Further split the data into training and testing

Scaled the independent variables in training and testing data.

```
[10] X_train=X_train/255
     X_test=X_test/255
```

Further

```
[12] model=Sequential([ Dense(44, input_shape=(45,),activation='relu'),
         Dense(22, input_shape=(45,),activation='relu'),
         Dense(10, activation='sigmoid')])
     model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
     model.fit(X_train,y_train, epochs=10)

     Epoch 1/10
     985/985 [==============================] - 5s 4ms/step - loss: 1.4013 - accuracy: 0.5190
     Epoch 2/10
     985/985 [==============================] - 4s 4ms/step - loss: 1.0635 - accuracy: 0.6314
     Epoch 3/10
     985/985 [==============================] - 4s 4ms/step - loss: 0.9943 - accuracy: 0.6487
     Epoch 4/10
     985/985 [==============================] - 4s 4ms/step - loss: 0.9609 - accuracy: 0.6590
     Epoch 5/10
     985/985 [==============================] - 3s 3ms/step - loss: 0.9402 - accuracy: 0.6658
     Epoch 6/10
     985/985 [==============================] - 3s 3ms/step - loss: 0.9247 - accuracy: 0.6686
     Epoch 7/10
     985/985 [==============================] - 4s 4ms/step - loss: 0.9156 - accuracy: 0.6716
     Epoch 8/10
     985/985 [==============================] - 3s 3ms/step - loss: 0.9061 - accuracy: 0.6754
     Epoch 9/10
     985/985 [==============================] - 3s 3ms/step - loss: 0.8991 - accuracy: 0.6766
     Epoch 10/10
     985/985 [==============================] - 3s 3ms/step - loss: 0.8918 - accuracy: 0.6782
     <keras.callbacks.History at 0x7f5ea723ae10>
```

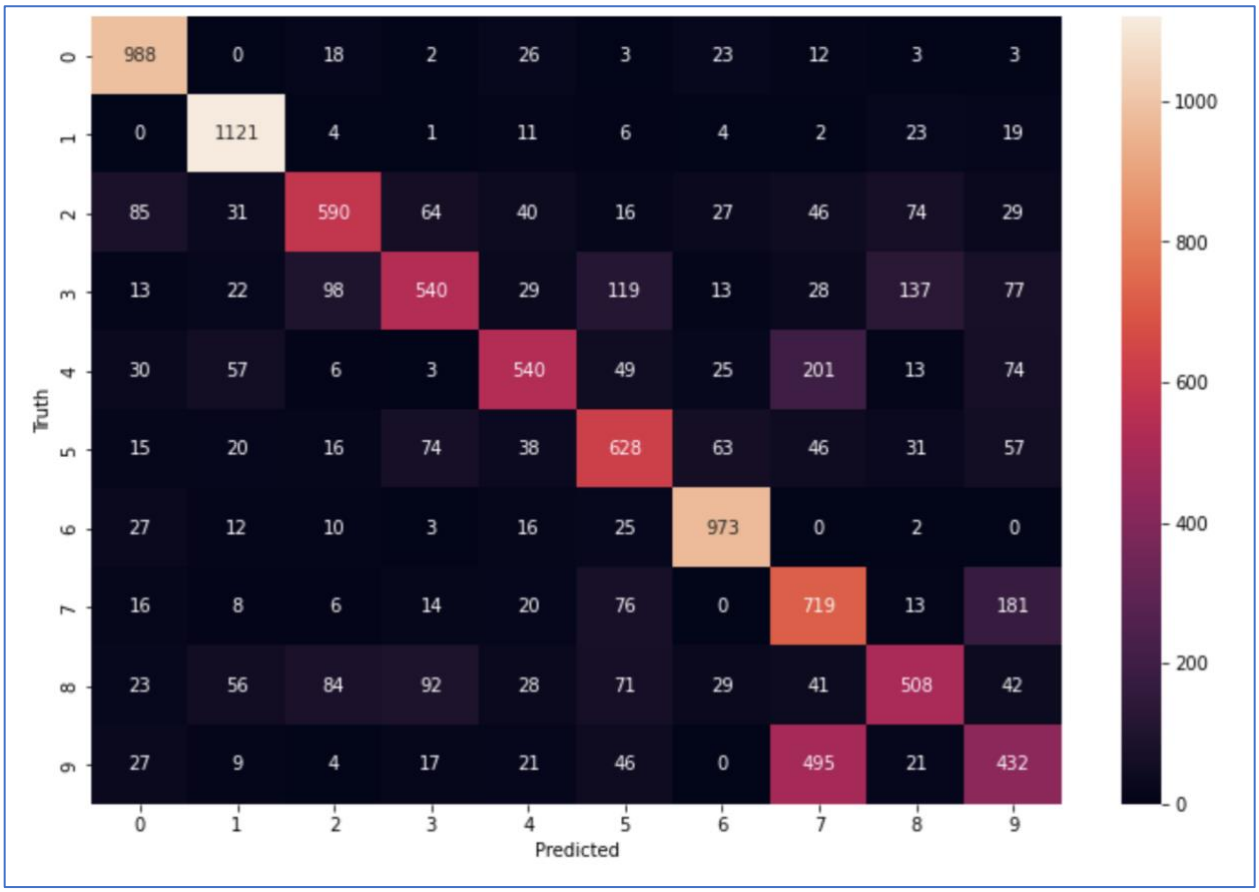
One loop across the entire training dataset is referred as an epoch. To improve accuracy, 10 Epoch were used. Rectolinear function was used to interpret the sequential model **(ReLU)**. The activation function in the output layer is **'Sigmoid'**, and the optimizer is **'adam'**.

```
[13] model.evaluate(X_test,y_test)

     329/329 [==============================] - 1s 1ms/step - loss: 0.9275 - accuracy: 0.6704
     [0.9275478720664978, 0.6703809499740601]
```

```
[25] print(classification_report(y_test,predictions_label))

                   precision    recall  f1-score   support

                0       0.88      0.89      0.88      1078
                1       0.82      0.95      0.88      1191
                2       0.71      0.58      0.64      1002
                3       0.63      0.55      0.59      1076
                4       0.72      0.53      0.61       998
                5       0.59      0.66      0.63       988
                6       0.83      0.92      0.87      1068
                7       0.45      0.71      0.55      1053
                8       0.65      0.49      0.56       974
                9       0.47      0.37      0.42      1072

         accuracy                           0.67     10500
        macro avg       0.68      0.67      0.66     10500
     weighted avg       0.68      0.67      0.67     10500
```

Achieved an accuracy of **67%** on the testing dataset

Later plot the confusion matrix for predicted vs truth values, it is given below



Here, we see that the diagonal has good bright colors which shows us that in most of the cases the scenarios where the actual and predicted values being matched has high percentage. Especially, the case of actual and predicted value being 1 has the highest amount of success rate where we see a total of **1121** cases being successful. Similarly, most of the values are predicted correct successfully expect for the values 8 and 9 where the prediction is not up to the mark. Particularly, for value **9** we predicted it to be **7** as many as **495** times but this has been the only case and that too as per my understanding and research has caused due to the possibility of incomplete dataset.

For modeling, we used the K nearest algorithm.

```
[20] knn = KNeighborsClassifier()
     knn.fit(X_train,y_train)
     y_pred_knn = knn.predict(X_test)

     'KNN Accuracy:'
     f'{round(accuracy_score(y_test, y_pred_knn) * 100, 2)} %'

     '64.66 %'
```

The accuracy for the K Nearest Algorithm was '**64.66**'.

```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_pred_knn))
print(classification_report(y_test, y_pred_knn))
```

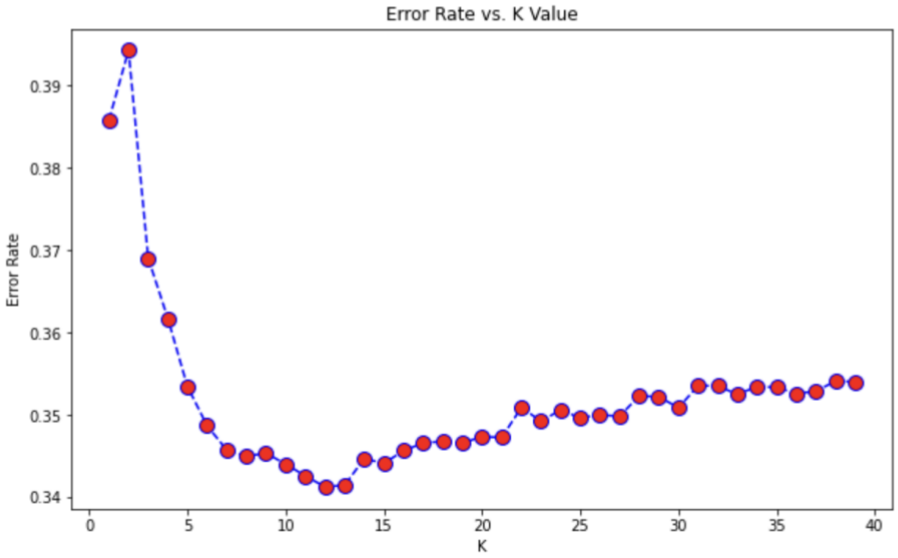
```
[[ 964    0   36    2   18    4   28   11    8    7]
 [   0 1134    7    3    7    8    4   13    9    6]
 [  71   38  618   85   25   13   19   56   59   18]
 [  19   28  132  581   30   75    9   41  117   44]
 [  25   80   11   17  569   29   22  157   13   75]
 [  12   41   35  154   38  522   55   62   31   38]
 [  37   26   14   18   14   18  932    1    8    0]
 [  12   12   15   35   90   51    0  627   15  196]
 [  20   80  113  133   20   59   26   38  453   32]
 [  30   22   10   41   72   40    0  446   22  389]]
              precision    recall  f1-score   support

           0       0.81      0.89      0.85      1078
           1       0.78      0.95      0.86      1191
           2       0.62      0.62      0.62      1002
           3       0.54      0.54      0.54      1076
           4       0.64      0.57      0.60       998
           5       0.64      0.53      0.58       988
           6       0.85      0.87      0.86      1068
           7       0.43      0.60      0.50      1053
           8       0.62      0.47      0.53       974
           9       0.48      0.36      0.41      1072

    accuracy                           0.65     10500
   macro avg       0.64      0.64      0.64     10500
weighted avg       0.64      0.65      0.64     10500
```

As seen with above modelling technique, i.e., that our predictions have been pretty accurate with most of the values expect with the value **9** which has repeated here as well and probably again because of the incompleteness of the dataset.

So, after computing both models, we can see that our **Neural Network** model outperforms the **K Nearest Neighbor** model in terms of accuracy.

```
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

```
Text(0, 0.5, 'Error Rate')
```



Iterated the model with several k values in the range of **1** to **40** and selected the k value after which the error rate was stagnant. Further, terminated the iteration process and concluded the model.