Project Update 1 **Nikita Bhatt**

Progress so far: So far, I have prepared my data analysis and algorithm part for the KNN Algorithm. I utilized a pre-curated section of the MNIST Dataset with 5620 data sets from the MNIST Dataset which originally has 50,000 data points. I did this to make processing faster, as parsing through 50,000 data points can take a lot of time. Although, I am unsure if using a pre-processed data set like this one would be allowed for the course project, I will confirm this with the TA before the next project deadline. All the information regarding how the parameters were picked for the best k values in the KNN Algorithm are mentioned in the pdf attached below, and the pdf has also been uploaded on my GitHub project page. I think I am entirely done with the report and data analysis part of the KNN Algorithm, which is what I expected to finish by this point.

Future Plan: So far, I am somewhat on track with regards to my timeline. Although I have been making regular progress with the project, I was not aware that we had to periodically update our Git to show this. Consequently, I have been working on my jupyter notebook and not been updating git. I will update git more regularly from now on however, since professor informed us of the grades associated with update regularity. According to my timeline, I should have started working on my second algorithm by now, which was using Linear Regression techniques for predictive modeling such as lasso and/or least squares. Although I have gained significant knowledge in these domains through the course, I have not made significant progress with regards to my project in these domains. I have until the 20th to finish working on this, and I might take a few days more. So for now, I will keep my Project Deadline intact.

Deadline	***Tasks to Accomplish***
11/17	**First Update Due**
11/17-11/20	Complete work on Algo 2 and add to report
11/20-11/30	Start working on Algo 3
11/30-12/1	Work on Second Update Document
12/1	**Second Update Due**
12/1 – 12/5	Finish working on Algo3 and wrap up
12/5-12/10	Last Minute Wrap ups and wiggle room
12/10 – 12/12	**Final Report Writing**
12/12	Final Report Due
12/13 - 12/17	Peer Review of Projects
12/17	Peer Review Due

Project_Update 1_Work

November 18, 2020

0.1 Applying KNN Classifier to the MNIST Dataset

Preliminary Information: MNIST Data set has 40k training samples and 10k test samples with each image having 28x28 pixels. In this step we will utilize the k-NN algorithm. This algorithm primarily assumes that feature vectors thatare closer in an n-dimensional space will display similar visual features. Given that the MNIST is an already well -curated data set, KNN despite being a simple algorithm can achieve high classification accuracy as will be seen in the following code. This is because the data set is heavily pre-processed with perfect cropping and thresholding, making it well curated. Real life problems are not as well put together.

Parameter Tuning

k-NN has 2 main types of parameters that can be changed to alter model accuracy. 1) the value of k: if k is too small, we can be more efficient, but at the cost of being more susceptible to noise and outliers. If k selected is too large, then we can increase bias.

2) The method used to calculate distance between nearest neighbors: In my code, I have used a pre-developed knn algorihm which utilises the Euclidean distance to measure so we will not be altering this parameter.

Procedure

To carry out my analysis, the following procedure is followed:

1) We train the data using a specific set with different values of k and possibly different metrics for measuring distance

2) We test the data on the validation set to observe the metrics that give us the hughest accuracy

3) Finally use these parameters to create a model which will be tested on the testing set

[121]:
```
import numpy as np
import pandas as pd
import sklearn
from sklearn import datasets, metrics
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from termcolor import colored

%matplotlib inline
```

```
#Using a preprocessed data set from the MNIST data set which has 8x8 pixels and␣
 ↪a little more than 5k samples
numbs = datasets.load_digits()
#Provides a brief description of the small sample of pictures selected from the␣
 ↪50k data set
print(numbs.DESCR)

#Setting matrix of features and labels
A = numbs.data
b = numbs.target
print("\n Printing some dimensions")
print(A.shape)
print(b.shape)

#Attempting to view an image from the data set being used:
plt.imshow(numbs.images[0])
plt.axis('off')
print("\nPrinting a digit 0 image")
```

```
.. _digits_dataset:

Optical recognition of handwritten digits dataset
--------------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 5620
    :Number of Attributes: 64
    :Attribute Information: 8x8 image of integer pixels in the range 0..16.
    :Missing Attribute Values: None
    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
    :Date: July; 1998

This is a copy of the test set of the UCI ML hand-written digits datasets
https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digit
s

The data set contains images of hand-written digits: 10 classes where
each class refers to a digit.

Preprocessing programs made available by NIST were used to extract
normalized bitmaps of handwritten digits from a preprinted form. From a
total of 43 people, 30 contributed to the training set and different 13
to the test set. 32x32 bitmaps are divided into nonoverlapping blocks of
4x4 and the number of on pixels are counted in each block. This generates
an input matrix of 8x8 where each element is an integer in the range
0..16. This reduces dimensionality and gives invariance to small
```

distortions.

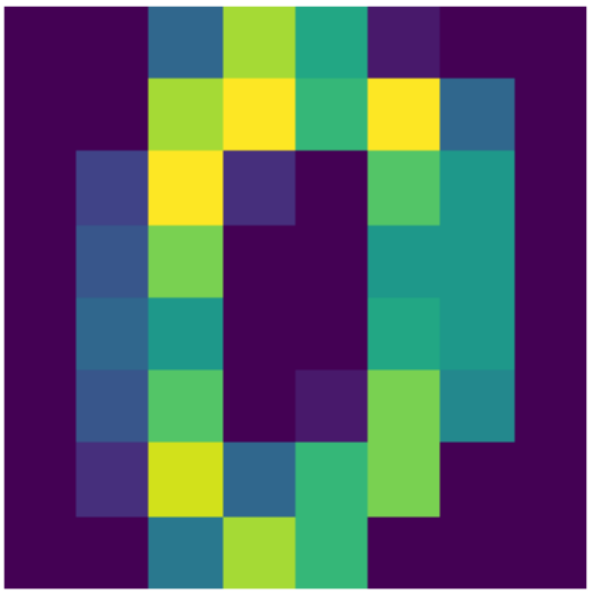
For info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G. T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C. L. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469, 1994.

.. topic:: References

  - C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their Applications to Handwritten Digit Recognition, MSc Thesis, Institute of Graduate Studies in Science and Engineering, Bogazici University.
  - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.
  - Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin. Linear dimensionalityreduction using relevance weighted LDA. School of Electrical and Electronic Engineering Nanyang Technological University. 2005.
  - Claudio Gentile. A New Approximate Maximal Margin Classification Algorithm. NIPS. 2000.

 Printing some dimensions
(1797, 64)
(1797,)

Printing a digit 0 image



[150]:
```
#Take the MNIST data and construct the training and testing data sets, use some␣
 ↪of the data for training and some for testing
(X_train, X_test, b_train, b_test) = train_test_split(np.array(numbs.data),␣
 ↪numbs.target, test_size=0.25)
print("All X matrix training data points: {}".format(len(b_train)))
# Taking 15% of the training data and setting aside for validation
(X_train, X_validation, b_train, b_validation) = train_test_split(X_train,␣
 ↪b_train, test_size=0.10)
print("Points from previously assigned training data points to be used for␣
 ↪VALIDATION:", len(b_validation))
print("All X matrix data points for TESTING: ", len(b_test))
print("All X matrix data points for TRAINING:", len(b_train))
print("\n")

#For loop to test accuracy for different k values ranging from 1 to 40
for k in range(1, 41, 1):
    accuracy_model = sklearn.neighbors.KNeighborsClassifier(n_neighbors=k)
    #Training KNN Algo
    accuracy_model.fit(X_train, b_train)
    #Calculating Model Accuracy
    print("k=%d, Classification Accuracy =%.2f%%" % (k, accuracy_model.
 ↪score(X_validation, b_validation) * 100))
print(colored("k = 1 gives us the maximum accuracy, and so k =2 is the best␣
 ↪parameter that we will choose to test the KNN model on the test data set",␣
 ↪'green', attrs=['bold']))

#Training KNN using k = 1
knn_model_new = sklearn.neighbors.KNeighborsClassifier(n_neighbors=1)
#Training KNN Algo
knn_model_new.fit(X_train, b_train)
#Estimating model accuracy on Test data set
b_predicted = knn_model_new.predict(X_test)
print(colored("\n\n\nClassification Report for KNN Classification", 'red',␣
 ↪attrs=['bold']))
#Generates a classification report analysing the accuracy for each digit's␣
 ↪classification
classif_report = sklearn.metrics.classification_report(b_test, b_predicted)
print(classif_report)
```

```
All X matrix training data points: 1347
Points from previously assigned training data points to be used for VALIDATION:
135
All X matrix data points for TESTING:  450
All X matrix data points for TRAINING: 1212


k=1, Classification Accuracy =98.52%
```

```
k=2, Classification Accuracy =97.78%
k=3, Classification Accuracy =97.78%
k=4, Classification Accuracy =97.78%
k=5, Classification Accuracy =98.52%
k=6, Classification Accuracy =97.04%
k=7, Classification Accuracy =97.04%
k=8, Classification Accuracy =97.04%
k=9, Classification Accuracy =97.04%
k=10, Classification Accuracy =97.04%
k=11, Classification Accuracy =97.78%
k=12, Classification Accuracy =97.78%
k=13, Classification Accuracy =97.78%
k=14, Classification Accuracy =97.78%
k=15, Classification Accuracy =97.78%
k=16, Classification Accuracy =97.78%
k=17, Classification Accuracy =97.78%
k=18, Classification Accuracy =97.78%
k=19, Classification Accuracy =97.78%
k=20, Classification Accuracy =97.04%
k=21, Classification Accuracy =97.04%
k=22, Classification Accuracy =97.04%
k=23, Classification Accuracy =96.30%
k=24, Classification Accuracy =96.30%
k=25, Classification Accuracy =96.30%
k=26, Classification Accuracy =96.30%
k=27, Classification Accuracy =96.30%
k=28, Classification Accuracy =96.30%
k=29, Classification Accuracy =96.30%
k=30, Classification Accuracy =96.30%
k=31, Classification Accuracy =96.30%
k=32, Classification Accuracy =96.30%
k=33, Classification Accuracy =96.30%
k=34, Classification Accuracy =96.30%
k=35, Classification Accuracy =96.30%
k=36, Classification Accuracy =96.30%
k=37, Classification Accuracy =96.30%
k=38, Classification Accuracy =96.30%
k=39, Classification Accuracy =96.30%
k=40, Classification Accuracy =95.56%
k = 1 gives us the maximum accuracy, and so k =2 is the best parameter

that we will choose to test the KNN model on the test data set

Classification Report for KNN Classification
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        46
```

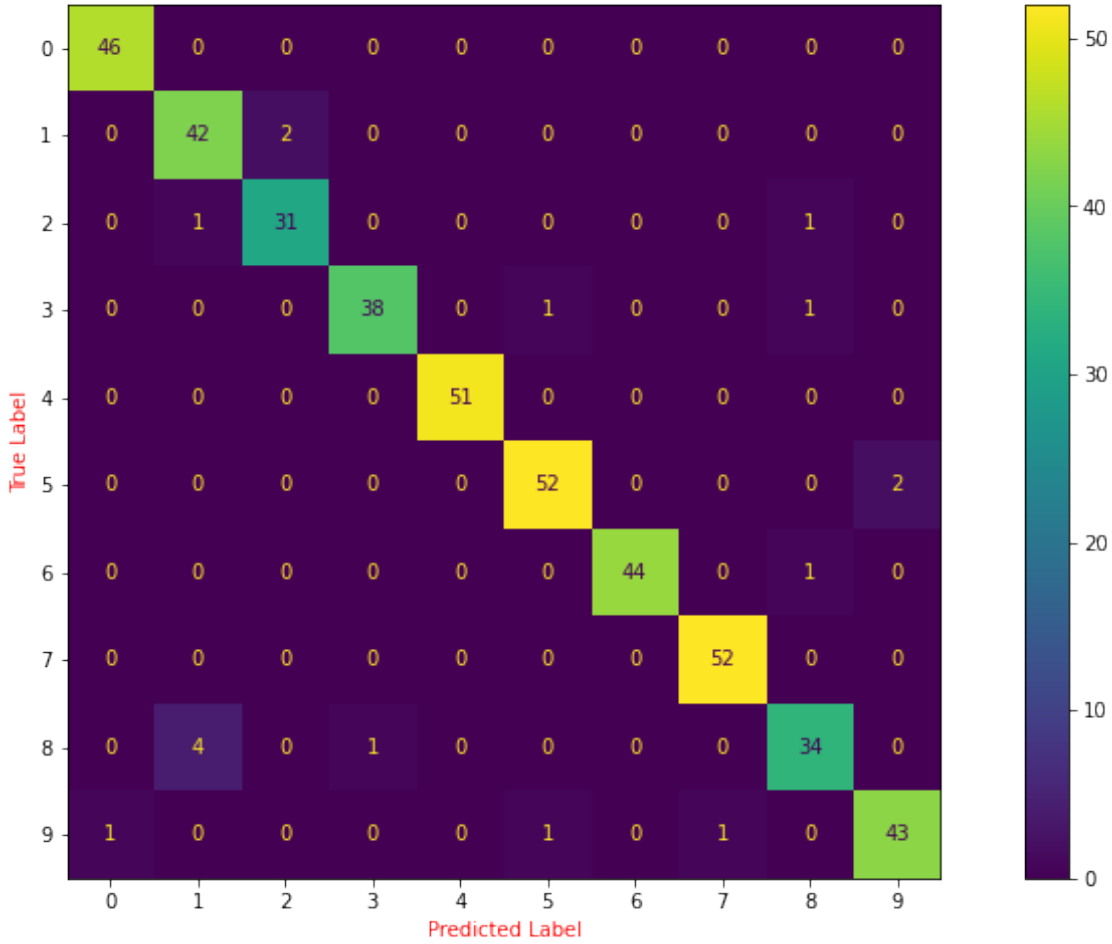
```
              1       1.00      1.00      1.00        44
              2       1.00      1.00      1.00        33
              3       0.93      1.00      0.96        40
              4       1.00      1.00      1.00        51
              5       1.00      0.98      0.99        54
              6       1.00      1.00      1.00        45
              7       1.00      0.98      0.99        52
              8       0.97      1.00      0.99        39
              9       0.95      0.91      0.93        46

    accuracy                            0.99       450
   macro avg       0.99      0.99      0.99       450
weighted avg       0.99      0.99      0.99       450
```

1 Generating a confusion matrix to qualitatively visualise the quality of the output of the KNN classifier

The confusion matrix below allows us to visualise the areas where the classifier might be "confused". The plot below gives us a heatmap type visual that will help us determine which samples have been well/poorly predicted.

[151]:
```
confusion_MAT = sklearn.metrics.plot_confusion_matrix(knn_model, X_test, b_test)
confusion_MAT.ax_.set_title('Confusion Matrix for KNN', color = 'white')
plt.xlabel("Predicted Label", color = 'red')
plt.ylabel("True Label", color = 'red')
plt.gcf().set_size_inches(15,8)
print(colored("CONFUSION MATRIX", 'red', attrs=['bold']))
```

```
CONFUSION MATRIX
```


True Label
Predicted Label
0 1 2 3 4 5 6 7 8 9
46 0 0 0 0 0 0 0 0 0
0 42 2 0 0 0 0 0 0 0
0 1 31 0 0 0 0 0 1 0
0 0 0 38 0 1 0 0 1 0
0 0 0 0 51 0 0 0 0 0
0 0 0 0 0 52 0 0 0 2
0 0 0 0 0 0 44 0 1 0
0 0 0 0 0 0 0 52 0 0
0 4 0 1 0 0 0 0 34 0
1 0 0 0 0 1 0 1 0 43
50 40 30 20 10 0