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

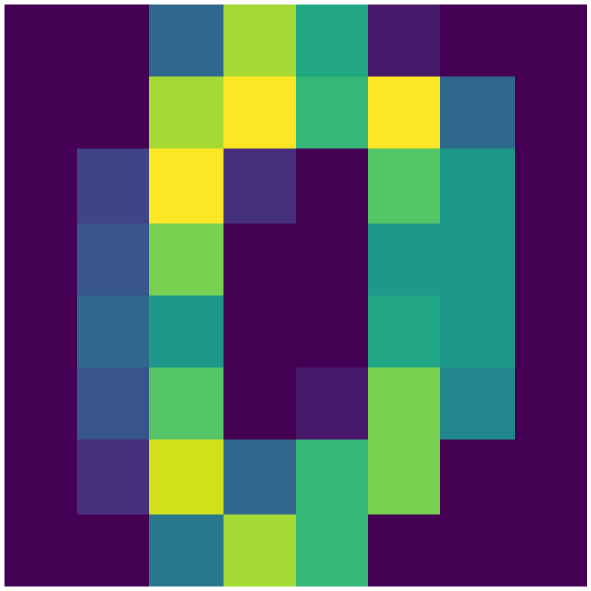
3) Finally use these parameters to create a model which will be tested on the testing set

[76]:
```
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
%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()
```

```python
#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
```



From this selective data set curated in sklearn using the massive MNIST Dataset, 3823 samples are used for training and the rest for testing. TESTING DATA SET: The shape of the X(feature) matrix is 1797 x 64. The shape of the y (labels) vector is 1797 x 1. This is because this selective data set has only used 8x8 pixels from the provided 28x28 pixels. Each sample has 64 features which are represented by the 8x8 image pixels with values from 0-16 representing pixel attributes.

```
[114]: #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=0.20)
       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=0.25)
       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):
           model = sklearn.neighbors.KNeighborsClassifier(n_neighbors=k)
           #Training KNN Algo
           model.fit(X_train, b_train)
           #Calculating Model Accuracy
           print("k=%d, accuracy=%.2f%%" % (k, model.score(X_validation, b_validation)␣
        ↪* 100))
       print("It can be observed that k = 1 gives us the maximum accuracy, and so k =1␣
        ↪is the best parameter that we will choose to test the KNN model on the test␣
        ↪data set")

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

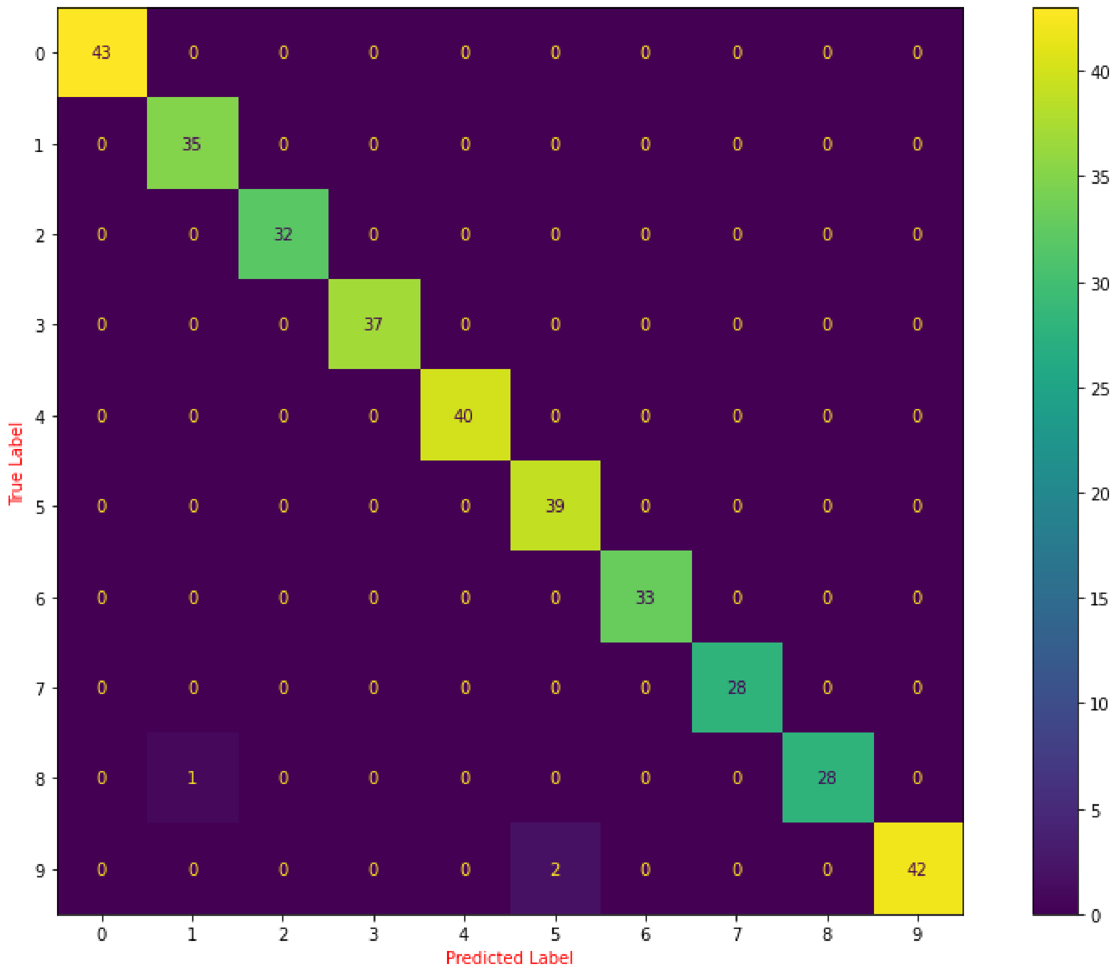
```
All X matrix training data points: 1437
Points from previously assigned training data points to be used for VALIDATION:
360
All X matrix data points for TESTING:  360
All X matrix data points for TRAINING: 1077


k=1, accuracy=98.33%
k=2, accuracy=97.78%
k=3, accuracy=97.50%
k=4, accuracy=98.06%
```

```
k=5, accuracy=97.78%
k=6, accuracy=97.50%
k=7, accuracy=97.50%
k=8, accuracy=97.22%
k=9, accuracy=97.22%
k=10, accuracy=97.22%
k=11, accuracy=97.50%
k=12, accuracy=97.50%
k=13, accuracy=97.22%
k=14, accuracy=97.22%
k=15, accuracy=96.67%
k=16, accuracy=96.67%
k=17, accuracy=96.39%
k=18, accuracy=95.83%
k=19, accuracy=95.56%
k=20, accuracy=95.56%
k=21, accuracy=95.83%
k=22, accuracy=95.56%
k=23, accuracy=95.56%
k=24, accuracy=95.83%
k=25, accuracy=95.28%
k=26, accuracy=95.56%
k=27, accuracy=95.28%
k=28, accuracy=95.56%
k=29, accuracy=95.28%
k=30, accuracy=94.72%
k=31, accuracy=94.17%
k=32, accuracy=94.17%
k=33, accuracy=94.17%
k=34, accuracy=93.89%
k=35, accuracy=93.89%
k=36, accuracy=93.89%
k=37, accuracy=93.61%
k=38, accuracy=93.61%
k=39, accuracy=93.61%
k=40, accuracy=93.61%
It can be observed that k = 1 gives us the maximum accuracy, and so k =1 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        43
           1       0.97      1.00      0.99        35
           2       1.00      1.00      1.00        32
           3       1.00      1.00      1.00        37
           4       1.00      1.00      1.00        40
           5       0.95      1.00      0.97        39
           6       1.00      1.00      1.00        33
           7       1.00      1.00      1.00        28
           8       1.00      0.97      0.98        29
           9       1.00      0.95      0.98        44

    accuracy                           0.99       360
   macro avg       0.99      0.99      0.99       360
weighted avg       0.99      0.99      0.99       360
```

0.2 Generating a confusion matrix to qualitatively visualise the quality of the output of the 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.

[115]:
```
confusion_MAT = sklearn.metrics.plot_confusion_matrix(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(15,10)
```



True Label \ Predicted Label	0	1	2	3	4	5	6	7	8	9
0	43	0	0	0	0	0	0	0	0	0
1	0	35	0	0	0	0	0	0	0	0
2	0	0	32	0	0	0	0	0	0	0
3	0	0	0	37	0	0	0	0	0	0
4	0	0	0	0	40	0	0	0	0	0
5	0	0	0	0	0	39	0	0	0	0
6	0	0	0	0	0	0	33	0	0	0
7	0	0	0	0	0	0	0	28	0	0
8	0	1	0	0	0	0	0	0	28	0
9	0	0	0	0	0	2	0	0	0	42