Handwritten-Digit-Classification-using-KNN

This repository introduces to my project "Handwritten-Digit-Classification" using MNIST Data-set . This project was implemented and executed by applying KNN algorithm with recognition accuracy of around 91-93 % . The desired results have been obtained by training the machine first using the mnist_train data-set and later testing the obtained results using mnist_test data-set , to recognise the handwritten digit.

Introduction

The aim of this project is to implement a classification algorithm to recognize handwritten digits (0- 9). It has been shown in pattern recognition that no single classifier performs the best for all pattern classification problems consistently. Hence, the scope of the project also included the elementary study the different classifiers and combination methods, and evaluate the caveats around their performance in this particular problem of handwritten digit recognition. This report presents our implementation of the Principal Component Analysis (PCA) combined with 1-Nearest Neighbor to recognize the numeral digits, and discusses the other different classification patterns. I was able to achieve an accuracy rate of 92.8%.

Overview of the Project

Hand writing recognition of characters has been around since the 1980s.The task of handwritten digit recognition, using a classifier, has great importance and use such as – online handwriting recognition on computer tablets, recognize zip codes on mail for postal mail sorting, processing bank check amounts, numeric entries in forms filled up by hand (for example - tax forms) and so on. There are different challenges faced while attempting to solve this problem. The handwritten digits are not always of the same size, thickness, or orientation and position relative to the margins. My goal was to implement a pattern classification method to recognize the handwritten digits provided in the MINIST data set of images of hand written digits (0-9). The data set used for our application is composed of 300 training images and 300 testing images, and is a subset of the MNIST data set [1] (originally composed of 60,000 training images and 10,000 testing images). Each image is a 28 x 28 grayscale (0-255) labeled representation of an individual digit. The general problem we predicted we would face in this digit classification problem was the similarity between the digits like 1 and 7, 5 and 6, 3 and 8, 9 and 8 etc. Also people write the same digit in many different ways - the digit '1' is written as '1', '1', '1' or '1'. Similarly 7 may be written as 7, 7, or 7. Finally the uniqueness and variety in the handwriting of different individuals also influences the formation and appearance of the digits.

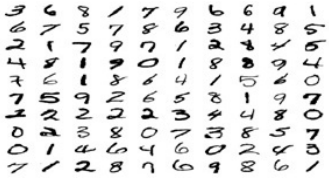


Fig. 1 Sample digits used for training the classifier

Literature Survey

Hand Written Character Recognition using Star-Layered Histogram Features Stephen Karungaru, Kenji Terada and Minoru Fukumi In this method, a character recognition method using features extracted from a star layered histogram is presented and trained using neural networks. After several image preprocessing steps, the character region is extracted. Its contour is then used to determine the center of gravity (COG). This CoG point is used as the origin to create a histogram using equally spaced lines extending from the CoG to the contour. The first point the line touches the character represents the first layer of the histogram. If the line extension has not reached the region boundary, the next hit represents the second layer of the histogram. This process is repeated until the line touches the boundary of the character's region. After normalization, these features are used to train a neural network.

Problem Statement

Given a set of greyscale isolated numerical images taken from MNIST database.

The objectives are:-

 To recognize handwritten digits correctly.

 To improve the accuracy of detection.

 To develop a method which is independent of digit size and writer style/ink independent.

Scope of Study

• Postal mail sorting • Courtesy amounts on cheques • Formation of data entry etc.

System Analysis

The first job of the group of students is to divide the different tasks between all the members of the group and organize the communication to be able to integrate all the parts before the evaluation of the project. System Design

• KNN algorithm K-Nearest Neighbors (or KNN) is a simple classification algorithm that is surprisingly effective. However, to work well, it requires a training dataset: a set of data points where each point is labelled (i.e., where it has already been correctly classified). If we set K to 1 (i.e., if we use a 1-NN algorithm), then we can classify a new data point by looking at all the points in the training data set, and choosing the label of the point that is nearest to the new point. If we use higher values of K, then we look at the K nearest points, and choose the most frequent label amongst those points.

However, in order to apply the k-Nearest Neighbor classifier, we first need to select a distance metric or a similarity function. We briefly discussed the Euclidean distance (often called the L2-distance) in our project:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

•MNIST Dataset The MNIST handwritten digit classification problem is a standard dataset used in computer vision and deep learning. Although, the dataset is effectively solved, it can be used as the basis for learning and practicing how to develop, evaluate, and use convolutional deep learning neural networks for image classification from scratch. This includes how to develop a robust test harness for estimating the performance of the model, how to explore improvements to the model, and how to save the model and later load it to make predictions on new data.

System Overview

Our approach to solve this problem of handwritten numeral recognition can be broadly divided into three blocks: i) Pre-Processing/Digitization ii) Feature Extraction using PCA iii) Classification using 1-Nearest Neighbor algorithm; The block diagram for the system is shown below (Fig. 2):

Fig.2. System Diagram of the implemented pattern classifier

Testing

• The overall classification design of the MNIST digit database is shown in following algorithm. Algorithm: Classification of Digits Input: Isolated Numeral images from MNIST Database Output: Recognition of the Numerals Method: Structural features and KNN classifier.

• Step 1: Convert the gray level image into Binary image

• Step 2: Preprocessing the Binary Image

• Step 3: Convert the Binary Image into a single Dimensional Array of [1,n]

• Step 4: Keep the label of each Array along with it.

• Step 5: Feed the classifier with the train_data set.

• Step 6: Repeat the steps from 1 to 5 for all images in the Sample and Test Database.

• Step 7: Estimate the minimum distance between feature vector and vector stored in the library by using Euclidian distances.

• Step 8: Feed the classifier with test_data set.

• Step 9: Classify the input images into appropriate class label using minimum distance K-nearest neighbor classifier.

• Step10: End.

How to Run ?

The project was executed using Anaconda software, and this source code file was tested on Jupyter notebook , so do ensure that you must have Anaconda software installed on your systems. After installation of Anaconda software , follow the steps mentioned below:-

Step-1 : Upload the source code file in the local directory of Jupyter notebook, by just clicking on upload button in the Home directory of Jupyter notebook.

Step-2 : Similarly, upload the train_data and test_data dataset files by unzipping the files on your systems.

Step-3 : Simply, execute the source code in the Jupyter notebook cells.

Implementation

In short, the problem of Handwritten digits Classification is solve by k-nearest-neighbors algorithm using MNIST data set. The MNIST dataset is an acronym that stands for the Modified National Institute of Standards and Technology dataset. It is a dataset of 60,000 small square 28×28 pixel grayscale images of handwritten single digits between 0 and 9. The task is to classify a given image of a handwritten digit into one of 10 classes representing integer values from 0 to 9, inclusively. It is a widely used and deeply understood dataset and, for the most part, is "solved." Top-performing models are deep learning convolutional neural networks that achieve a classification accuracy of above 99%, with an error rate between 0.4 %and 0.2% on the hold out test dataset. We obtain our data from the MNIST Data Set, and with a few minor modifications, A single line of the data file represents a handwritten digit and its label. The digit is a 256-element vector obtained by flattening a 16×16 binary-valued image in row-major order; the label is an integer representing the number in the picture. The data file contains 1593 instances with about 160 instances per digit. After reading in the data appropriately, we randomly split the data set into two pieces, train on one piece, and test on the other. The following function does this, returning the success rate of the classification algorithm on the testing piece. A run with gives a surprisingly good 89% success rate. Varying , we see this is about as good as it gets without any modifications to the KNN algorithm Of course, there are many improvements we could make to this naive algorithm. But considering that it utilizes no domain knowledge and doesn't manipulate the input data in any way, it's not too shabby. As a side note, it would be fun to get some tablet software and have it use this method to recognize numbers as one writes it. Alas, we have little time for these sorts of applications.

Screenshots

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [16]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
```

In [17]:
```
dataset = pd.read_csv("train_data.csv")
```

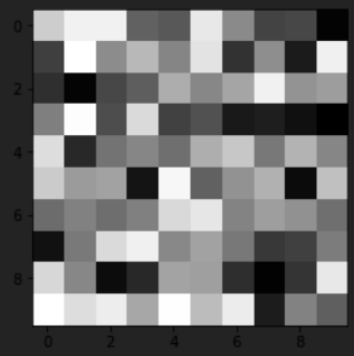
In [18]:
```
dataset.head()
```

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0

5 rows × 785 columns

In [4]:
```
# Plotting images
a = np.random.random((10,10))
plt.figure()
plt.imshow(a, cmap='gray')
plt.show()
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3



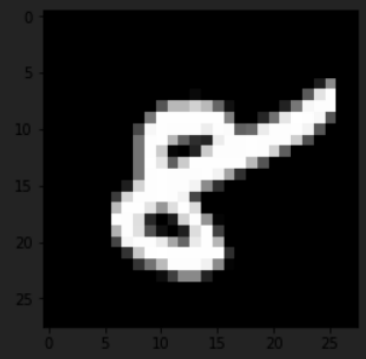
In [19]:
```
data = dataset.values[:5000]
data.shape
```

```
(5000, 785)
```

In [20]:
```
X, y = data[:,1:], data[:, 0]
```

In [21]:
```
# Plotting digit
im = X[4997].reshape((28,28))
plt.figure()
plt.imshow(im, cmap='gray')
plt.show()
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3



In [22]:
```
split = int(X.shape[0] * 0.80)
X_train, X_test, y_train, y_test = X[:split], X[split:], y[:split], y[split:]
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
print(y_train)
print(y_test)
```

```
(4000, 784) (4000,)
(1000, 784) (1000,)
[1 0 1 ... 6 6 4]
[8 8 0 5 0 0 3 8 2 1 2 5 6 3 0 6 6 1 3 9 6 0 2 4 6 3 2 7 8 3 2 9 9 4 7 2 7
 8 6 1 5 0 9 5 5 2 5 7 4 0 2 9 8 3 2 1 2 4 6 2 3 3 2 1 6 8 5 5 6 1 8 4 4 4
 7 8 1 2 0 4 5 8 2 3 4 2 7 7 9 4 7 5 9 5 7 0 4 0 6 8 5 8 6 0 4 3 6 2 0 9 1
 4 9 7 9 5 6 2 2 0 8 4 2 9 9 5 9 6 9 0 4 0 9 0 6 6 8 4 8 7 3 1 4 9 0 1 7 0
 7 7 3 3 2 4 0 9 6 1 9 2 8 6 5 0 2 1 7 8 0 0 8 6 1 5 4 0 6 6 9 7 7 1 0 8 6
 3 8 5 6 0 8 7 3 2 3 4 5 9 2 5 7 7 1 8 1 7 5 6 4 4 2 6 9 6 7 9 1 2 1 9 6 8
 1 4 2 0 9 4 0 1 1 9 9 1 7 6 2 9 0 3 4 3 2 5 2 0 5 1 8 5 6 0 6 3 7 3 4 4 6
 7 1 3 6 1 5 1 0 3 5 6 0 4 6 3 3 9 2 3 9 3 9 3 4 2 6 3 4 6 4 2 7 9 7 5 1 9
 6 4 0 6 0 3 0 9 9 5 1 1 7 2 1 6 2 3 0 9 1 6 5 9 5 9 2 1 2 2 2 3 1 2 2 6 8
 9 4 9 9 4 4 7 2 3 3 6 8 1 2 3 6 5 6 7 8 8 4 8 2 6 8 0 1 3 6 7 5 1 6 1 6 1
 3 0 9 0 9 1 3 0 2 1 2 4 3 6 2 5 5 0 2 4 0 2 3 3 1 1 0 4 7 4 0 4 1 1 8 3 8
 5 7 7 4 4 3 6 6 5 4 9 2 5 8 6 5 1 5 7 3 9 5 7 8 8 0 7 0 1 6 8 9 1 1 7 9 8
 7 9 3 7 3 9 3 2 1 1 2 7 3 9 1 0 8 2 3 4 0 6 7 9 4 7 0 9 2 5 3 0 4 3 6 3 8
 6 5 8 6 1 6 3 3 2 8 2 0 0 0 2 4 3 2 4 5 3 9 8 7 4 5 7 0 1 1 3 3 9 6 5 6 4
 4 4 6 3 0 1 5 8 2 0 0 1 8 1 7 4 7 4 3 2 1 2 5 2 8 5 7 5 0 5 4 2 4 0 9 1 5
 3 1 7 7 2 0 7 5 9 0 1 9 9 7 2 3 6 9 0 1 2 7 1 9 8 5 7 7 5 3 8 2 7 8 1 7 7
 7 6 7 6 5 2 9 4 5 2 8 7 2 9 6 5 0 4 2 6 8 2 1 9 2 0 2 3 6 0 2 4 0 5 8 2 4
 2 1 5 8 7 9 2 9 2 3 5 7 2 1 3 6 8 1 5 7 8 0 0 4 3 0 6 9 8 8 9 4 9 3 8 1 7
 3 9 9 5 2 8 9 6 4 1 4 2 9 1 9 8 5 2 7 1 8 1 3 5 8 8 6 3 6 5 5 1 8 9 4 9 9
 0 2 2 3 6 8 1 2 4 2 9 3 5 1 0 6 6 8 1 0 2 6 0 6 3 2 0 5 5 9 0 8 9 7 7 3 4
 9 1 2 8 9 4 3 9 3 6 1 1 4 2 8 6 2 7 2 2 6 0 1 9 7 6 4 2 6 3 3 7 2 9 1 5 5
 3 7 6 5 3 6 0 9 0 1 2 7 1 0 8 5 1 1 6 6 1 3 8 8 1 7 1 7 9 7 1 3 6 5 5 0 3
 6 8 0 1 3 8 7 5 5 9 3 8 3 6 8 7 5 8 0 2 5 1 6 2 7 4 4 3 9 7 3 9 1 6 0 4 4
 8 6 5 2 3 1 3 2 8 4 7 2 8 9 2 8 2 4 8 4 9 4 1 0 5 9 8 1 7 1 5 9 1 6 3 7 6
 0 9 5 3 6 5 6 5 1 3 2 1 3 2 3 8 3 5 7 2 0 4 3 1 2 9 6 3 7 0 8 1 3 6 4 8 4
 3 5 7 7 0 1 1 8 1 5 6 5 2 3 2 5 1 3 3 4 6 4 1 1 8 1 6 8 1 2 4 8 0 3 4 9 4
 2 0 3 0 3 7 7 8 7 7 2 8 3 8 3 6 7 7 3 6 7 0 9 1 3 0 2 8 7 1 4 0 7 2 5 8 7
 9]
```

In [23]:

```python
def knn(X_train, y_train, test_point, k=5):

    distances = [] # Contains list of tuples (distance, label

    for data_point, label in zip(X_train, y_train):
        distances.append((euclidean(test_point, data_point), label))
    #     for i in range(X_train.shape[0]):
    #         data_point = X_train[i]
    #         label = y_train[i]

    sorted_distances = sorted(distances, key=lambda x: x[0])
    k_nearest_neighbors = np.array(sorted_distances[:k])
    freq = np.unique(k_nearest_neighbors[:,1], return_counts=True)
    labels, counts = freq
    majority_vote = labels[counts.argmax()]
    return majority_vote
```

In [24]:

```python
def euclidean(p1, p2):
    return np.sqrt(np.sum((p1-p2)**2))

euclidean(np.array([1,2,3]), np.array([4,5,6]))
```

```
5.196152422706632
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [25]:
```
def calculate_accuracy(X_test, y_test, X_train, y_train, k=5):

    predictions = []

    for test_point in X_test:
        pred_label = knn(X_train, y_train, test_point, k)
        predictions.append(pred_label)

    predictions = np.array(predictions)

    accuracy = (predictions == y_test).sum() / y_test.shape[0]
    return accuracy
```

In [26]:
```
calculate_accuracy(X_test, y_test, X_train, y_train, k=5)
```

```
0.928
```

In [27]:
```
test_df = pd.read_csv("test_data.csv")
```

In [28]:
```
test_df.head()
```

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775	pixel776	pixel777	pixel778
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0

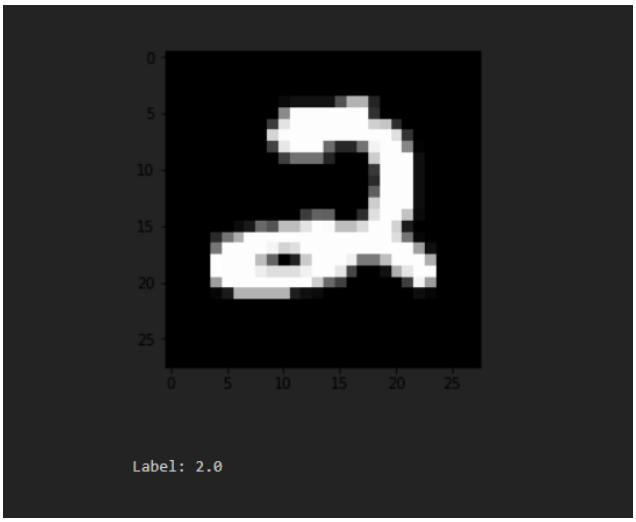
5 rows x 784 columns

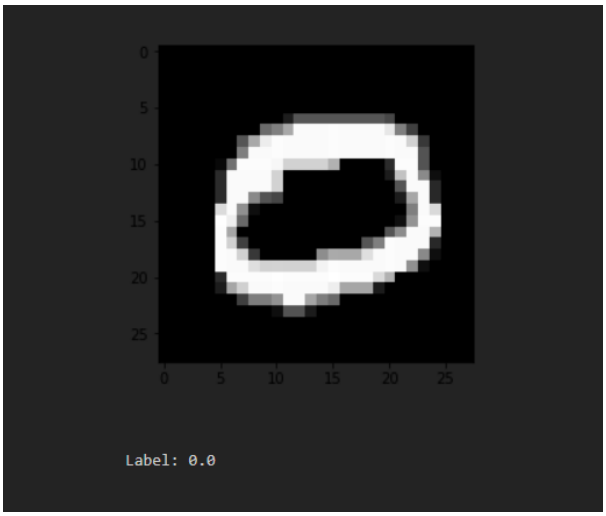
In [29]:
```
test_data = test_df.values
test_images = test_data[:10]
test_images.shape
```

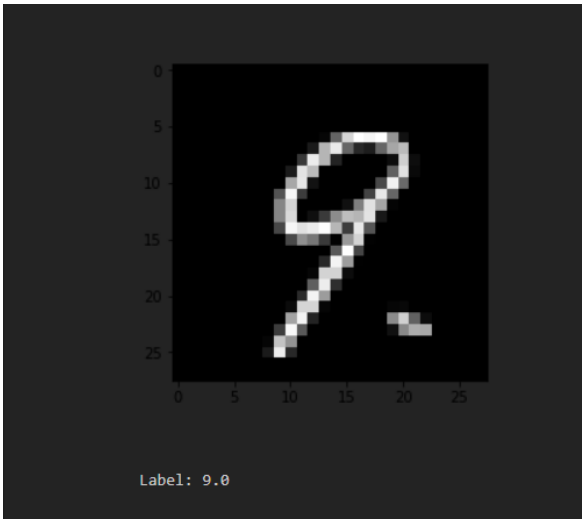
```
(10, 784)
```

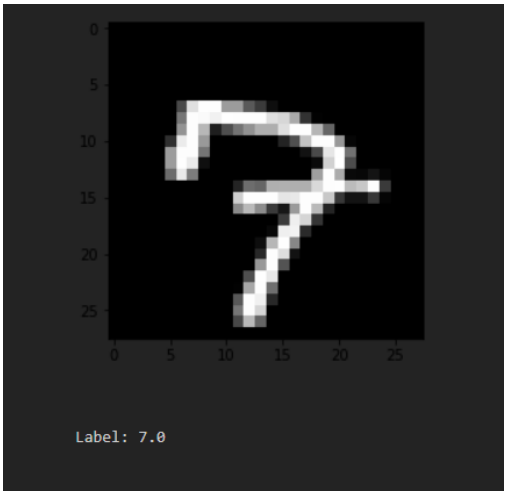
In [30]:
```
for test in test_images:
    im = test.reshape((28,28))
    plt.figure()
    plt.imshow(im, cmap='gray')
    plt.show()
    print("Label:", knn(X_train, y_train, test))
```

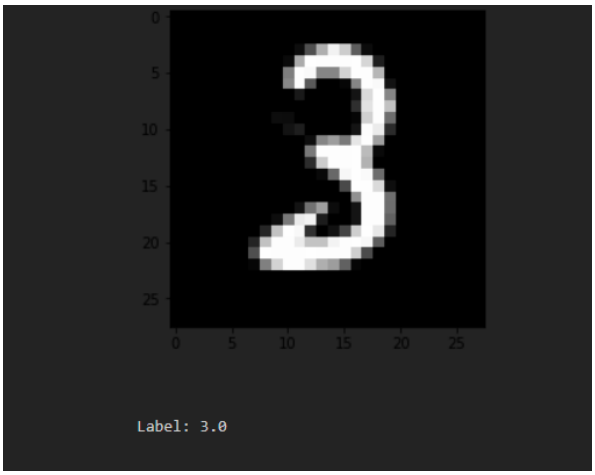
Output


Label: 2.0


Label: 0.0


Label: 9.0


Label: 7.0



Summary

In this project, we learnt about the most simple machine learning classifier — the k-Nearest Neighbor classifier, or simply k-NN for short. The k-NN algorithm classifies unknown data points by comparing the unknown data point to each data point in the training set. This comparison is done using a distance function or similarity metric. Then, from the k most similar examples in the training set, we accumulate the number of "votes" for each label. The category with the highest number of votes "wins" and is chosen as the overall classification.

While simple and intuitive, and though it can even obtain very good accuracy in certain situations, the k-NN algorithm has a number of drawbacks. The first is that it doesn't actually "learn" anything — if the algorithm makes a mistake, it has no way to "correct" and "improve" itself for later classifications. Secondly, without specialized data structures, the k-NN algorithm scales linearly with the number of data points, making it a questionable choice for large datasets.

Conclusion

The proposed project shows the whole process of supervised classification from the data acquisition to the design of a classification system and its evaluation. The project is usually extremely successful with the students. Most of them are very proud of the performance of their system and most of the time, they implement side games such as Sudoku or letter games (such as scrabble) to show the performance of their system online. The project, besides giving practical applications to some machine learning techniques seen in class, gives them the opportunity to work as a group. As explained in the text, some changes can be made on the representations of the images and on the distances used to compare two elements in the database.