Sabastian Mugazambi
HMW 1 : Data Mining Spring 2017

---

1. **Pre-processing**

   In the data we have the images of the handwritten digits have been pre-processed by using the center of mass of the grayscale values to center the images and scale them to the same size. The pre-processing is necessary to standardize and normalize the data so that our results are not affected by inconsistencies the input data. Preprocessing will also increase the accuracy of the models we use.

   In this hand written digit recognition another pre-processing that could have been done would have been rotation removal. Some handwritten digits are slightly rotated by a certain degree and one could remove this rotation before running the prediction models. The centering of the images could have also been done by looking at the first non-zero pixel coming from all directions of the digit. This crops the image such that we have a minimal number of zeros in the grayscale value vectors.

2. **knn.py**

   knn.py has been implemented and is included within the same directory. It is run as follows;

   ```
   $ python knn.py {k} {distance}
   ```

   e.g   python knn.py 3 euclidean
   e.g   python knn.py 5 manhattan

3. **Results of knn.py**

   ➜ **HMW1 python knn.py 3 euclidean**

```
k :  3
Distance :  euclidean
Confusion Matrix:
              1     2     7
        1 100.0   0.0   0.0
        2   7.0  87.0   6.0
        7   6.0   1.0  93.0
Accuracy: 0.933333333333
```

More clearly the accuracy is **0.93333333** and the confusion matrix is as below;

| | | **PREDICTED** | | |
|---|---|---|---|---|
| | | **1** | **2** | **7** |
| **ACTUAL** | **1** | 100.0 | 0.0 | 0.0 |
| | **2** | 7.0 | 87.0 | 6.0 |
| | **7** | 6.0 | 1.0 | 93.0 |

➜ **HMW1 python knn.py 1 manhattan**

```
k :  1
Distance :  manhattan
Confusion Matrix:
              1     2     7
        1 100.0   0.0   0.0
        2  10.0  85.0   5.0
        7   5.0   0.0  95.0
Accuracy: 0.9333333333333
```

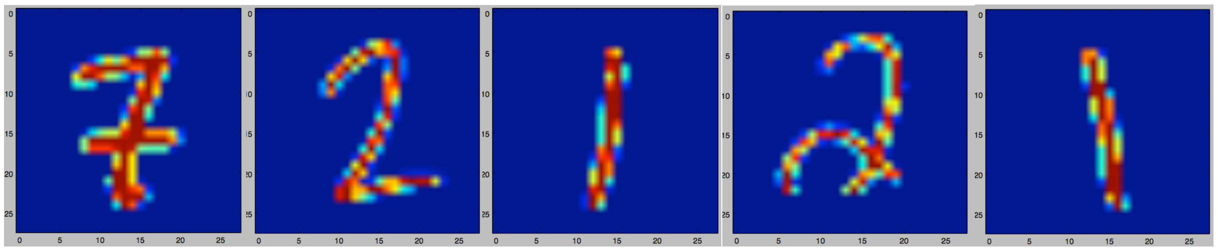
More clearly the accuracy is **0.93333333** and the confusion matrix is as below;

| | | **PREDICTED** | | |
|---|---|---|---|---|
| | | **1** | **2** | **7** |
| **ACTUAL** | **1** | 100.0 | 0.0 | 0.0 |
| | **2** | 10.0 | 85.0 | 5.0 |
| | **7** | 5.0 | 0.0 | 95.0 |

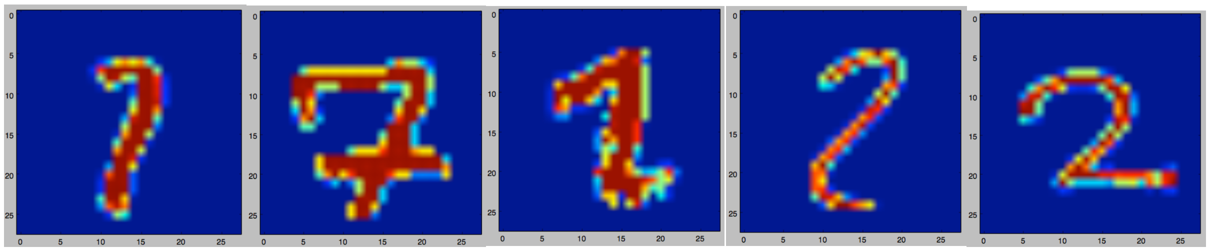
## 4. Images

**$ python knn.py 3 euclidean**

I randomly selected the images to plot out from my pool of **correctly** predicted images and these where the 5 randomly picked images;



The following images where from the incorrectly predicted digits;



**Patterns**

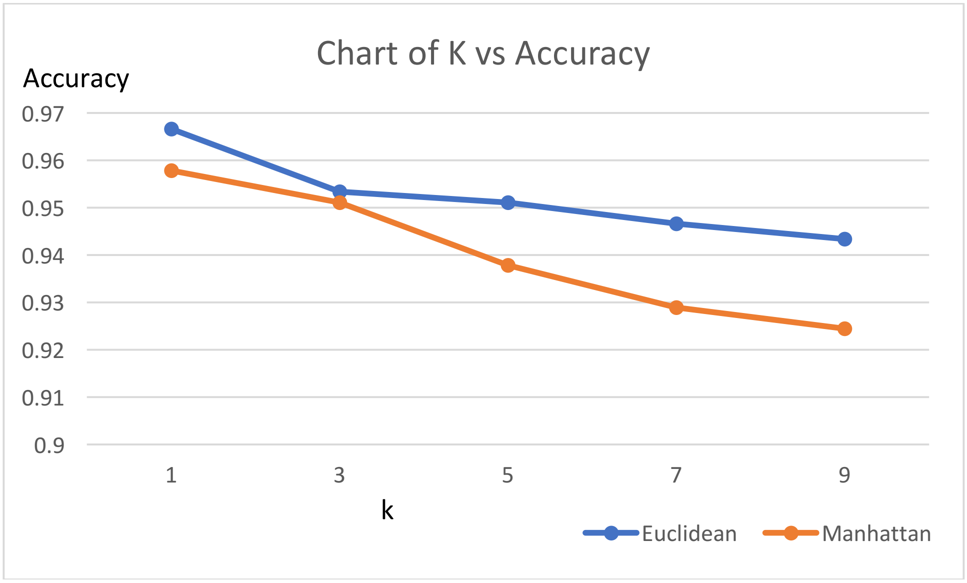
It is easy to see that the incorrectly recognized digits do not follow the conventional shape of the actual digits they represent. Some of the digits have loops on them which makes it hard for the model to correctly predict what the digits are. It also seems as if rotated digits seem to throw the model off that is why I suggested rotation removal as part of pre-processing. The trend in correctly recognized digits seems to be that of non-rotation and consistency with conventional shapes of the digits. Un-finished or digits that have minimized some of the features like the 7 in the second group seem to be incorrectly predicted.

## 5. Cross Validation

After running the cross validation for the set of k values {1, 3, 5, 7, 9 } on a 5-fold cross validation I got the following results for accuracy;

| **k** | **Euclidean** | **Manhattan** |
|---|---|---|
| **1** | 0.96667 | 0.95778 |
| **3** | 0.95333 | 0.95111 |
| **5** | 0.95111 | 0.93778 |
| **7** | 0.94667 | 0.92889 |
| **9** | 0.94333 | 0.92444 |

A plot of the results shows that **a k of 1 using the Euclidean distance** measure has the highest accuracy of **0.96667** however it is 0.01 higher than the accuracy of the same classifier when run on the test data which is **0.956667**.



## 6. Scaling the data

The computation cost of classifying test images would increase by a factor of about 5800*3*900 since for each test image we are comparing it to 900 other images in our training data set and taking the nearest neighbors.

The computation cost of classifying test images would increase by a factor of about ((18000/5)*4*(18000/5))*5 since we are deviding the training data set into 5 groups and for a total of (18000/5) test images we are training them on (18000/5)*4 images and we are doing this 5 times.