CS584 – Machine Learning

Digit and Letter: Recognition and Classification

Harsh Parikh
Sanket Nitin Wagh

Abstract

Handwriting recognition is the ability to detect what is being written/input by the user, by the computer. The input is usually on paper. Usually when it is written on paper the content is optically scanned to generate the data. This entails a concept called optical character recognition. We will be focusing on off line recognition based on structural features. We will be scanning a set of digits and see whether or not these digits are correctly being predicted. Classification will be performed using algorithms like K-NN, PCA, Random Forests. We will also be using the MNIST dataset that already contains large amounts of digit data.

Key words: OpenCV, Hand written digits, Structural Recognition, Classification.

I. INTRODUCTION

Offline character recognition is essential in many fields that require the scanned versions of digits and letters. For example a bank will require the character recognition system to extract the digits from a check scanned on a mobile app. We will be validating the accuracy of the classification once the data is being classified using the algorithms that we build from scratch. In this project we use OpenCV for our feature extraction method and then analyzing the structural composition of the digit. We will check if predictions are correct or not. Also, we will be using a massive digits dataset to cross validate our findings.

In the Section III we will discuss the type of data used and how it shall impact our classifiers and also the feature extraction method. Section IV shall give an idea of how the algorithms have been implemented and in Section V we will discuss the results of our classification.

II. RELATED WORK

Since this field is booming since there is a shift in primary use of form filling from direct paper to mobile apps and scanned documents. It has become important to extract and recognize the data efficiently. Apart from an instance that includes banks, there is a wide range of applications such as postal mail sorting and all forms of data entry. There is research that suggests the use of structural features of the digits to be used such as number of holes, water reservoirs in four directions, maximum profile distances in four directions, and fill-hole density for the recognition of digits.

III. DATA

We will be using the MNIST dataset. Recognition of Handwritten Digits Dataset. The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. We also tested the classification techniques on Load-Digits dataset available in the sklearn library. Since form filling involves writing words as well we tried implementing the algorithms on Letter-Recognition dataset.

FEATURE EXTRACTION:

We also created a mechanism where we scan digits written by hand by any user and applied the classification algorithms on that image.
We use Open CV to implement the feature extraction. Since all the images can be of any form we use gray-scale to alter the image for better results. For example an image with a set of digits along with a stray scratch or a dot will alter the result considerably. Using gray scale as mentioned we convert the background into a black background and the digits are converted to a white color. This will get rid of the spots and scratched and will yield better results. The exact implementation consists of setting a threshold. If the color of any material found in the scan crosses a pre-set values it will be converted to white. This proved to be quite effective as the result was as follows:

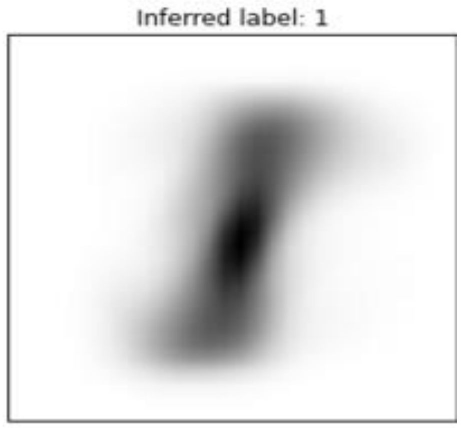


Figure1: Example of a digit 1 in MNIST

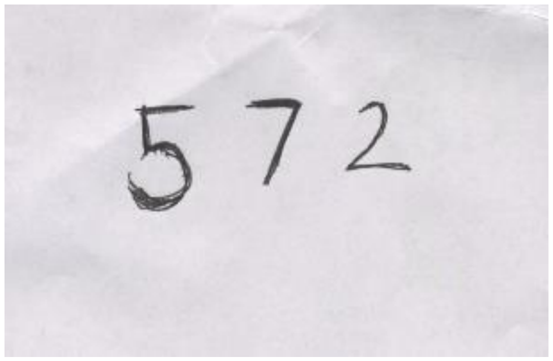


Figure 2: Manual user input

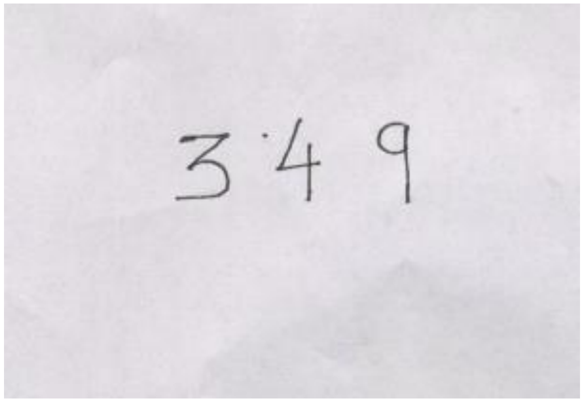


Figure 3: Manual input with a scratch.

Figure 4: Input converted using gray scale.

Since we want data to obtain the best results possible we utilize the Gaussian Blur to make the digits into an MNIST- equivalent digit.

The blur causes the digit to become slightly disoriented and hence we get different versions or samples of the same digit.

IV. Algorithm Design and Implementation.

A. K-NN (K-Nearest Neighbor):

The K-NN algorithm is an instance based lazy learning algorithm. There is no assumption of data of any sort. The Euclidean distance is used obtain the nearest neighbors. These neighbors cast 'votes' in order to predict the class. It will create a localized model just before prediction and feed it to the unseen data. The advantage here is that only relevant data is going to be fed to the unseen sample of data.

The working is as follows: The data is fed and distance between the instances is determined. In the next step the K most similar instances are obtained. These are the neighbors selected. Followed by which the neighbors will collectively predict a response. As we iterate through the set of neighbors, we get a vote form the nearest neighbors as to which class the instance in question is part of. In our case, we are trying to predict the type of digit that is being given as an input. The output is obtained using a number of training samples. We also predict letters.

We have implemented this algorithm to determine the neighbors and hence find the predicted digits and letters.

B. Principal Component Analysis:

This is an intuitive process where emphasis underlying structure of the data. It is used to identify patterns which are used to reduce dimensions. This helps in classification with minimal loss of information. The data is projected as data points on a hyper plane. This is represented in the form of eigenvalues and eigenvectors.

We will be focusing on dimensionality reduction. We apply PCA to reduce data and feed it to the K-NN algorithm. We scale from a small dimension of features to the max dimensions. Accuracy for each will be obtained.

C. Random Forests:

The Random Forests Algorithm is a unique one which can handle large datasets with ease. It can perform regression as well as classification. We will be focusing on the classification aspect of it.

Instead of using one single decision tree, random forests use multiple decision trees in collect multiple weak models to come up with a strong one. The trees are essential as they will generate a classification result. In a way it can be compared to K-NN since the trees are going to 'vote' for the class that is going to be predicted.

The working is as follows: The number of cases in the training data is chosen as N. This is going to generate random subsamples of data. These subsamples or subsets of data are used to create the decision trees for each subset.

If there are M input variables, a number ‘m’ which is less than M is specified such that at each node in each tree. ‘m’ variables are selected at random out of the M. The best split on these ‘m’ is used to split the node. The value of m is maintained as a constant while we grow the forest. This is done such that each tree is grown to the fullest capacity. Each of the tree will provide a prediction. These can be considered as the ‘votes’. An aggregate of these votes will help predict the class.

We set values in our implementation. Each subsample is going to produce a tree that has at most two child nodes. The depth and max number of features to be generated are also preset. These values will be used to train the data. Training will produce a list of decision trees. This is going to be used to predict the class while testing. Accuracy is also obtained. Results are cross validated using inbuilt functions. The advantage of the random forest algorithm is that we do not need an external algorithm to reduce the dimensions of the data.

V. RESULTS AND DISCUSSION

Here we compare and observe the various results obtained. We start off by checking the various forms of input:

1. Analyzing the result for the image with “5 7 2” written by the user:

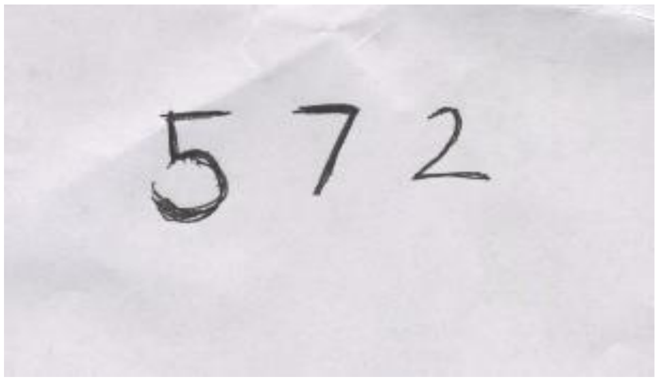


Figure 4: Phase 1; Data extraction

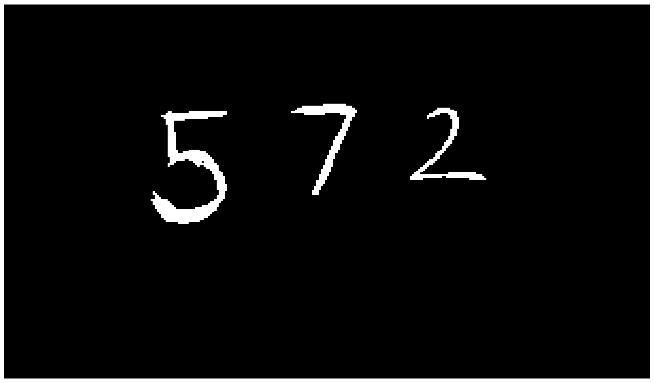


Figure 5: Phase2, Gray scale application

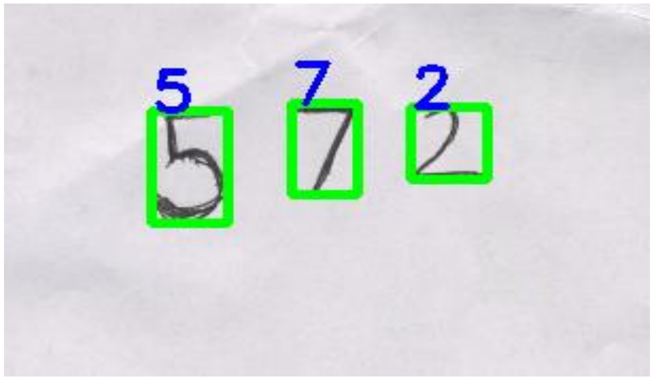


Figure 5: Phase 3; Prediction

Phase 1 is the feature extraction phase. In this phase the user’s input is captured and processed as shown in the above images. The data is accurately parsed and each digit is correctly classified. This was obtained using the K-NN algorithm.

2. Analyzing the result for the image with “3 4 9” written by the user which includes a *stray mark:*

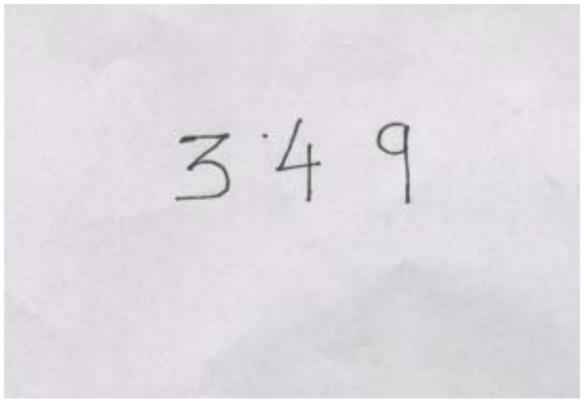


Figure 6: Phase 1; Data extraction

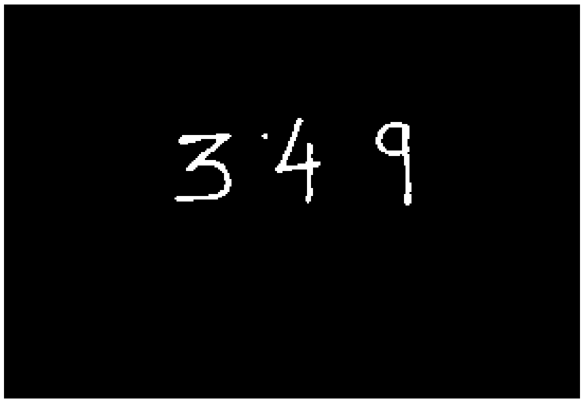


Figure 7: Phase2; Gray scale application

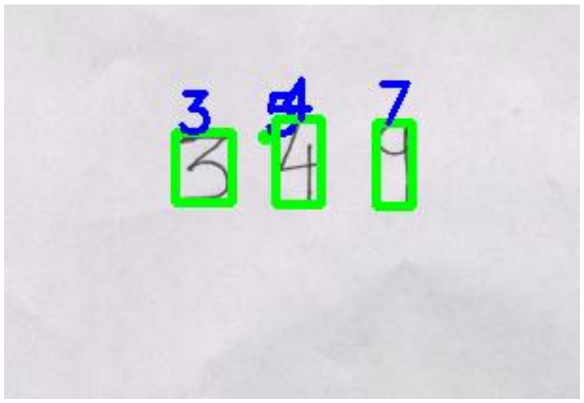


Figure 8: Phase 3; Prediction

This result was also obtained using the K-NN algorithm and was not 100% accurate. This is due to the stray mark present.

3. Analyzing the result for the image with “I AM AWESOME” written by the user which includes a *stray mark:*

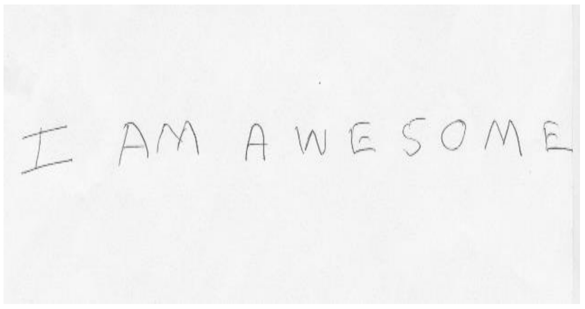


Figure 9: Phase 1; Data extraction

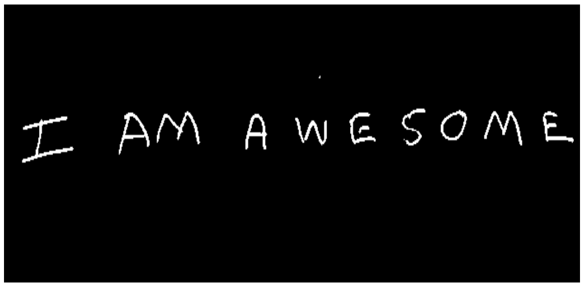


Figure 10: Phase2; Gray scale application

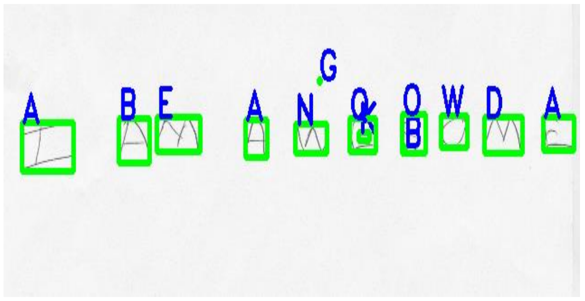
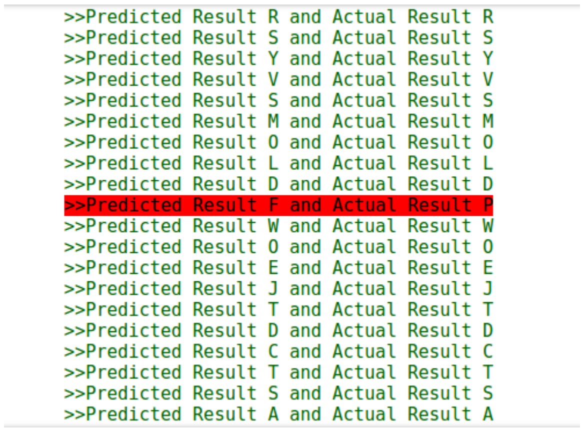


Figure 11: Phase 3; Prediction

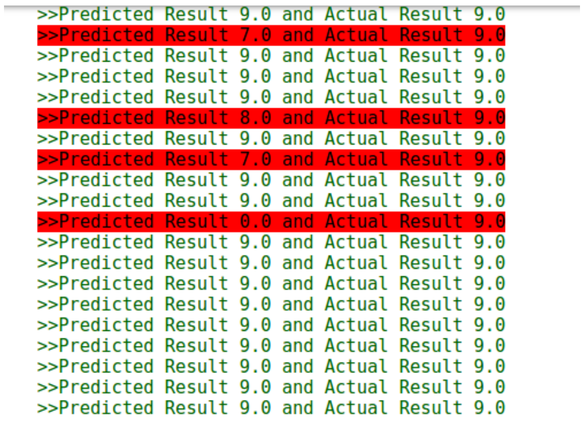
The above was an attempt at letter recognition and was unsuccessful. As it can be observed, the stray mark was not able to get ignored and the letters have been incorrectly classified.



```
>>Predicted Result R and Actual Result R
>>Predicted Result S and Actual Result S
>>Predicted Result Y and Actual Result Y
>>Predicted Result V and Actual Result V
>>Predicted Result S and Actual Result S
>>Predicted Result M and Actual Result M
>>Predicted Result O and Actual Result O
>>Predicted Result L and Actual Result L
>>Predicted Result D and Actual Result D
>>Predicted Result F and Actual Result P
>>Predicted Result W and Actual Result W
>>Predicted Result O and Actual Result O
>>Predicted Result E and Actual Result E
>>Predicted Result J and Actual Result J
>>Predicted Result T and Actual Result T
>>Predicted Result D and Actual Result D
>>Predicted Result C and Actual Result C
>>Predicted Result T and Actual Result T
>>Predicted Result S and Actual Result S
>>Predicted Result A and Actual Result A
```

Figure 12: Prediction of letters.

Although the user input was not successfully classified the letter-recognition.data is a dataset that we used and the algorithm performed effectively on it. An accuracy of 99.88% was obtained. The prediction in read is the incorrectly classified letter.



```
>>Predicted Result 9.0 and Actual Result 9.0
>>Predicted Result 7.0 and Actual Result 9.0
>>Predicted Result 9.0 and Actual Result 9.0
>>Predicted Result 9.0 and Actual Result 9.0
>>Predicted Result 9.0 and Actual Result 9.0
>>Predicted Result 8.0 and Actual Result 9.0
>>Predicted Result 9.0 and Actual Result 9.0
>>Predicted Result 7.0 and Actual Result 9.0
>>Predicted Result 9.0 and Actual Result 9.0
>>Predicted Result 9.0 and Actual Result 9.0
>>Predicted Result 0.0 and Actual Result 9.0
>>Predicted Result 9.0 and Actual Result 9.0
>>Predicted Result 9.0 and Actual Result 9.0
>>Predicted Result 9.0 and Actual Result 9.0
>>Predicted Result 9.0 and Actual Result 9.0
>>Predicted Result 9.0 and Actual Result 9.0
>>Predicted Result 9.0 and Actual Result 9.0
>>Predicted Result 9.0 and Actual Result 9.0
>>Predicted Result 9.0 and Actual Result 9.0
>>Predicted Result 9.0 and Actual Result 9.0
```

Figure 13: Prediction of numbers.

In similar fashion we used the algorithm on MNIST, which is our goal dataset. We obtained 89% accuracy classifying this dataset.

4. Accuracy obtained using PCA:

Number of Features	Accuracy
5	77.3
10	80.2
20	81.2
40	81.9
60	82.7
80	81.2
100	82.8
200	81.7
400	82.9
784	86

Table 1: Reduced dimensions vs Accuracy values – K-NN

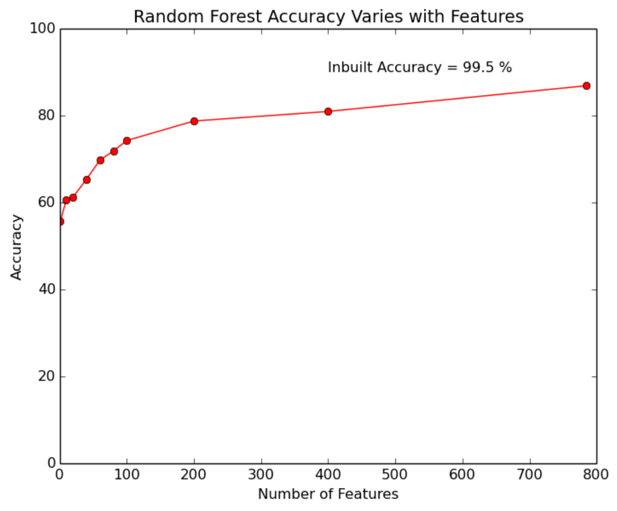


Figure 14: Graphical representation of accuracy values – K-NN.

We can deduce that the best accuracy was obtained when large amounts of data were sampled.

5. Accuracy using Random Forests:

Number of Features	Accuracy
1	55.6
10	60.6
20	61.2
40	65.3
60	69.7
80	71.8
100	74.2
200	78
400	80.9
784	86.8

Table 2: Reduced dimensions vs Accuracy values – Random Forests

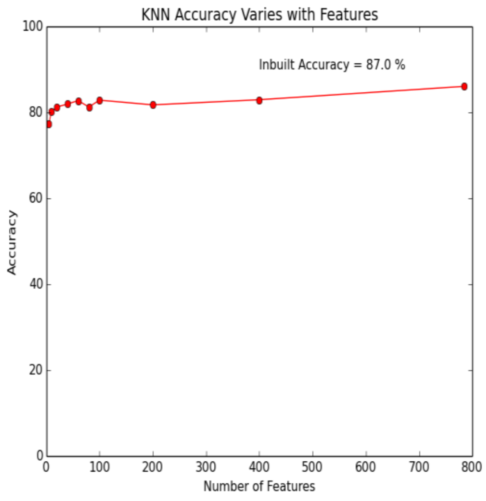
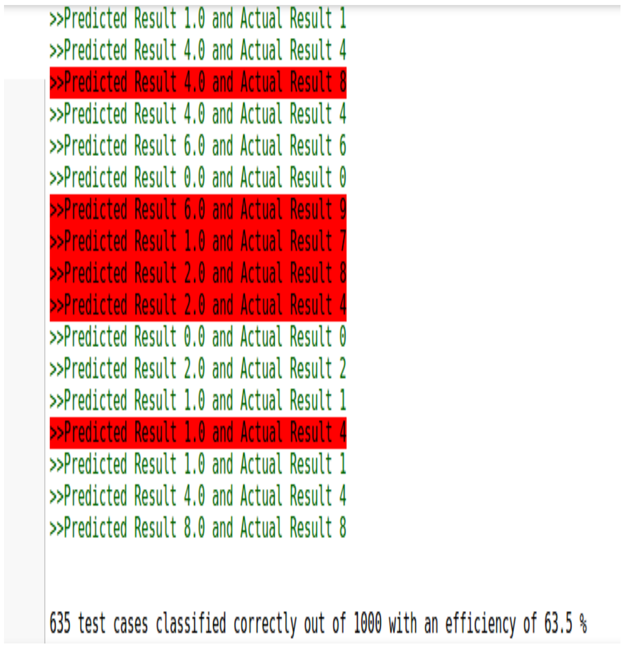


Figure 14: Graphical representation of accuracy values – Random Forests.

Data can be scaled using the random forests algorithm. The best accuracy obtained was again when max samples were used.



```
>>Predicted Result 1.0 and Actual Result 1
>>Predicted Result 4.0 and Actual Result 4
>>Predicted Result 4.0 and Actual Result 8
>>Predicted Result 4.0 and Actual Result 4
>>Predicted Result 6.0 and Actual Result 6
>>Predicted Result 0.0 and Actual Result 0
>>Predicted Result 6.0 and Actual Result 9
>>Predicted Result 1.0 and Actual Result 7
>>Predicted Result 2.0 and Actual Result 8
>>Predicted Result 2.0 and Actual Result 4
>>Predicted Result 0.0 and Actual Result 0
>>Predicted Result 2.0 and Actual Result 2
>>Predicted Result 1.0 and Actual Result 1
>>Predicted Result 1.0 and Actual Result 4
>>Predicted Result 1.0 and Actual Result 1
>>Predicted Result 4.0 and Actual Result 4
>>Predicted Result 8.0 and Actual Result 8


635 test cases classified correctly out of 1000 with an efficiency of 63.5 %
```

Figure 15: Prediction of random forests.

The above is are presentation of the classification algorithm on test cases. 1000 test cases were used. 635 cases were correctly classified. With an accuracy of 63.5 percent.

VI. CONCLUSION

Method	In-Built Library Implementation
KNN for Digits	90.41%
KNN for Letters	99.88%
KNN with PCA	87.00%
Random Forest	99.50%
Method	User Implementation
KNN for Digits	89.00%
KNN for Letters	97.00%
KNN with PCA	86.00%
Random Forest	86.80%

Table 3: User implanted results vs results obtained using inbuilt function

To conclude we can say that the classifiers worked effectively on the datasets. MNIST was classified with excellent accuracy.

When you compare the user defined implementation with the inbuilt libraries, we can say there is a mixed result. While the classifiers do work well.

When MNIST data was classified using our classifier it performed approximately 1% less efficiently than the inbuilt classifier.

When the K-NN algorithm was used on letter-recognition data, our classifier was outperformed by the inbuilt classifier by 2.88%.

We applied PCA on the data to reduce dimensions. We scaled the data as the dimensions were reduced simultaneously. As this was done, our classifier was outperformed by the inbuilt functions by 1%.

Application of Random Forest yielded an accuracy less than 10% than that of the inbuilt functions.

VII. FUTURE WORK

There is definitely scope for improvements. We can determine a way in which we can recognize stray scratches and dots and eliminate them. This can be considered as a vital preprocessing step.

This can be done by analyzing various people's handwriting as determine the difference between a stray scratch and something important like the tittle on top of an 'i'. We must delve into structural recognition to resolve this issue. Also techniques more useful than gray-scale can be used to enhance character recognition, be it a digit or a letter.

As we know, English is not the only language spoken or this planet. Hence it is not going to be the only language forms and checks are written in. We can delve into languages like Arabic and Mandarin and perform character recognition. Both those languages entail some form of drawing rather than simple letters that we have used. It would be very interesting to see the results in these languages.

As an enthusiast, we can also use Sanskrit and Hindi as the base language that can be used for recognition and classification. Both of them use a unique script and the results obtained can also be very interesting.

We could use fill-hole density, profile distance and advanced structural distance in order to achieve positive results. We can also include a more effective scanning technique that will allow us to retain on the digits and letters excluding the stray markings.

VIII. REFERENCES

[1] Dr. Y Venkateswarlu, U Ravi Babu, Aneel Kumar Chintha, “Handwritten Digit Recognition Using K-Nearest Neighbour Classifier”

[2] Nyssa Aragon, William Lane, Fan Zhang, “Classification of Hand-Written Numeric Digits”

Links:

[1]http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6755106

[2]http://cs229.stanford.edu/proj2013/AragonLaneZhang-ClassifyHandWrittenNumericDigits.pdf

[3]http://yann.lecun.com/exdb/mnist/

[4]https://www.stat.berkeley.edu/~breiman/random-forests.pdf

[5]http://www.analyticsvidhya.com/blog/2015/09/random-forest-algorithm-multiple-challenges/

[6]http://sebastianraschka.com/Articles/2014_pca_step_by_step.html

[7]http://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/