(ENGLISH HAND-WRITTEN DIGIT RECOGNITION)

Project report submitted
in partial fulfillment of the requirement for the degree of

Bachelor of Technology

By

Rishab (1505137)

Anish Hota (1505094)

Pranesh Biswas (1505232)

Champak Sinha (1505205)

Anand Kumar (1505365)

Under the guidance of
Ms. Jyotismita Chaki

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY

Deemed to be University, BHUBANESWAR

(April 2018)

Acknowledgement

We would like to thank our mentor Ms. Jyotismita Chaki for providing constant support and giving us a broader picture of the whole scenario without which the completion of the project would not have been possible. Her constant endeavour and guidance has helped us complete the project within the stipulated time. Last but not the least, we would like to thank our fellow project mates for helping us out with all the problems that we faced and making our experience an enjoyable and educative one.

Declaration

We declare that this written submission represents our ideas in our own words and where others ideas or words have been included, we have adequately referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misinterpreted or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will cause for disciplinary action by the Institute and can also evoke penal action from the sources and which have thus not been properly cited or from whom proper permission has not been taken when needed.

Rishab 1505137

Anish Hota 1505095

Anand Kumar 1505361

Champak Sinha 1505205

Pranesh Biswas 1505232

Date: 30th April, 2018

1.Introduction

The significant task of handwritten digit recognition has great importance in the recognition of postcodes sort mail, bank check amounts and so on. Since three decades, there is no single classifier performs the best for all pattern classification problems consistently. There are different challenges faced while attempting to solve this problem. The handwritten digits are not always of the same thickness, size, orientation or position.

There have been several techniques used to counter this problem. Many people have tried using several types of classifiers. Most popular types of the classifier as of now have been the Support vector classifier and the convolutional neural networks. In SVMs for digit classification problems, the training set is large and is comparatively slow. Neural Networks have widely been used to counter this drawback of SVM's. However, in our project we have considered only the support vector machines. A benchmark of handwritten digit recognition with several optimal approaches, datasets, and feature representations have been developed.

Several classifiers and features are evaluated on MNIST handwritten digit database. Some of the other datasets that have been readily used for this purpose are Cenparmi and Semion datasets. In this paper, our work is focused on an accurate and feasible method applied and tested on training set (15,000 images) and test set (7,500 images) of MNIST handwritten digit database and additional custom digits. Even though the MNIST dataset is already standardized in our project we found that some further preprocessing divided into several stages was required and played a very crucial role in the success if and it reflects the accuracy of the classification process.

The custom digits were entered through a self-made User Interface which had black background and the digits were drawn in white over the black canvas. The custom digits were the preprocessed to make them fit for the recognition system. The classifiers that were used were of 5 types which included the SVM'S linear classifier, polynomial classifier and RBF kernel classifier along with two other classifiers namely KNN classifier and Random Forest classifier.

2.1 Previous researches:

2.1.1

Youssouf Chherawala, Partha Pratim Roy and Mohamed Cheriet in their paper “Feature Set Evaluation for Offline Handwriting Recognition Systems: Application to the Recurrent Neural Network” stated that handwriting recognition system is dependent on the features extracted from the digit image. There are various methods to extract the features but there are no method that have been proposed to identify the most promising of these other than a straightforward comparison based on the recognition rate. So, they proposed a framework for feature set evaluation. They use a weighted vote combination of recurrent neural network (RNN) classifier.

2.2.2

Nurul Ilmi, Tjokorda Agung Budi W and Kurniawan Nur R in their paper “Handwriting Digit Recognition using Local Binary Pattern Variance and K-Nearest Neighbors Classification” using Local Binary Pattern (LBP) as feature extraction and K-NN classification on their handwriting recognition system on the C1 form used by General Elections Commission in Indonesia. The testing result is LBP variance can recognize handwriting digit character on MNIST dataset with accuracy 89.81% and for data from C1 form, the accuracy is 70.91%

2.2.3

In paper “Online Handwriting Verification with Safe Password and Increasing Number of Features”, present a solution to verify user with safe handwritten password using Bayes Net, Naïve Bayes, K-NN and Multi-layer Perceptron (MLP) classifier. The main goal is to reduce the features to achieve the same or better result after ranking and reduce the processing time for classification.

3 Report on present work

3.1 Goal

Our goal for this project is quite simple—we want to take pixel data from images of hand-drawn digits and classify them as a number from 0 to 9.

3.2 Database

The data for our project was taken from the MNIST dataset. As written on a Kaggle competition using the MNIST dataset, "The MNIST ('Modified National Institute of Standards and Technology') dataset is a classic within the Machine Learning community that has been extensively studied. More detail about the dataset, including Machine Learning algorithms that have been tried on it and their levels of success, can be found at http://yann.lecun.com/exdb/mnist/index.html." Each handwritten digit is vectorized. Each vector is composed of a label which represents the label classification (i.e. numbers from 0 to 9) and 783-pixel features of integer values between 0 and 255. Since all of the data is already normalized and centered, we did not need to take those steps ourselves to start building models against. The training data set consists of 60,000 data points, while the test set consists of 10,000 data points. The specific flavor of the MNIST dataset that we used came from https://pjreddie.com/projects/mnist-in-csv/ where the data is formatted as CSV files.

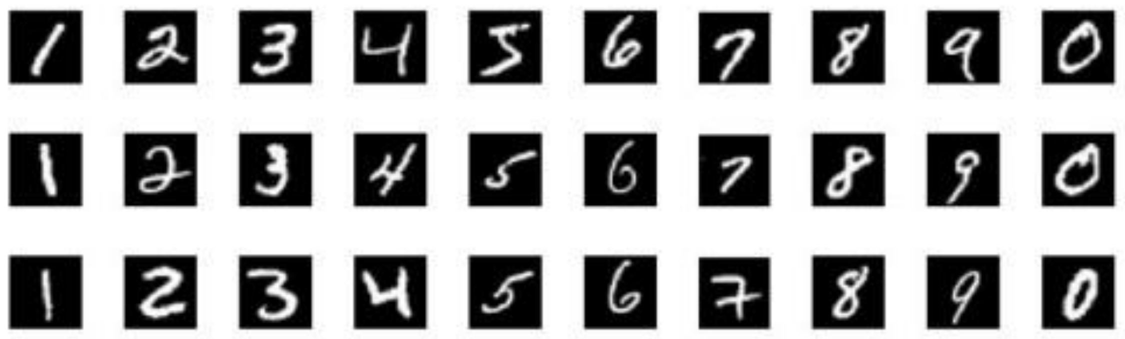


Fig.1: Sample images from MNIST dataset

3.3 Preprocessing required for the project

3.3.1 The techniques used

Firstly, a preprocess step is started with threshold in the gray-scale digit image into a binary image, and then noise removal and thinning are performed. Secondly, by reducing the search space, the region-of-interest (ROI) is cropped from the preprocessed image, Experimental results on MNIST database will be reported in the paper to support the feasibility of our method. The steps followed by us is listed and described in detail in the sections that follow.

3.3.2 Taking the input

Firstly, the images were read from the user interface that was designed using Tkinter in Python. The UI had a black canvas and allowed the digits to be drawn only in white on the black background. This allowed us an easy way to threshold the image. The image drawn on the canvas was by hand on a touch enabled laptop. The thickness of the drawing line was about 20px.The prediction process started by clicking on the "Predict" button specified in the UI.

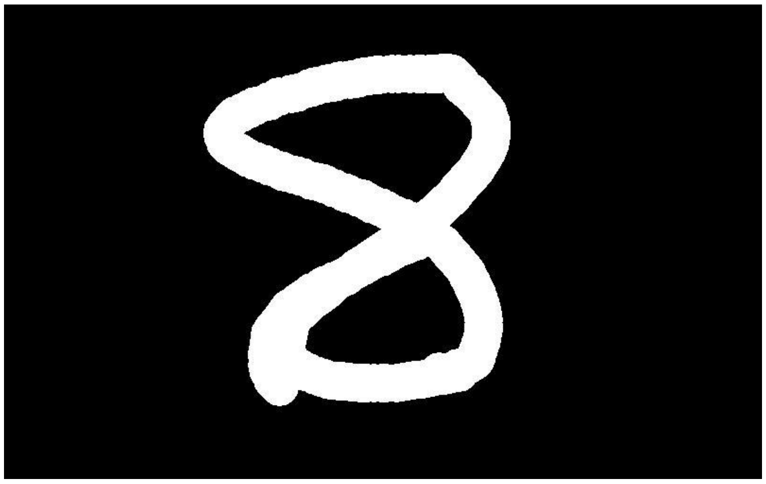


Fig 2.: The custom digit fed to the recognizer

3.3.3 Resizing

Differences in digit size are due to different image resolutions and handwriting styles. Image resolutions and handwriting styles. To minimize the effect of these differences, all digits are resampled/rescaled to the same size in digits are resampled/rescaled to the same size in pixels, before being fed to the recognizer. Pixels, before being fed to the recognizer the canvas is of 800x500 pixels, the image would have been too large it was necessary for the images to be resized into proper size to ensure proper processing of the image further in the system. Every image that was input was resized to a 28x28 pixels i.e. 784 pixels keeping the aspect ratio same.

3.3.4 Thresholding

Image thresholding is a simple, yet effective, way of partitioning an image into a foreground and background. This image analysis technique is a type of image segmentation that isolates objects by converting grayscale images into binary images. This was a technique applied by us to make sure that each pixel in the image after preprocessing was either black or white and nothing else. Since the picture is hand drawn, it may contain some pixel values which do not correspond to the two colors that we need that are black and white. The curved regions may have slightly less intensity than the white pixel or even greyish. So to overcome this problem what we did was after resizing the image, for each of the 784 pixels of the image we calculated the mean of the RGB value of each individual pixel. Then we found the summation of the RGB values and took out the average of the RGB values of all the pixels. If a pixel's RGB value was greater than the calculated average intensity, its value was set to 255 which corresponds to the white color in RGB system, else if the RGB value of the pixel was less than the average value, that pixel's value was set to 0, which corresponds to the black color in RGB system. This way we could change the greyscale values of the image only to the two values that are black and white which made it more fit for our system.

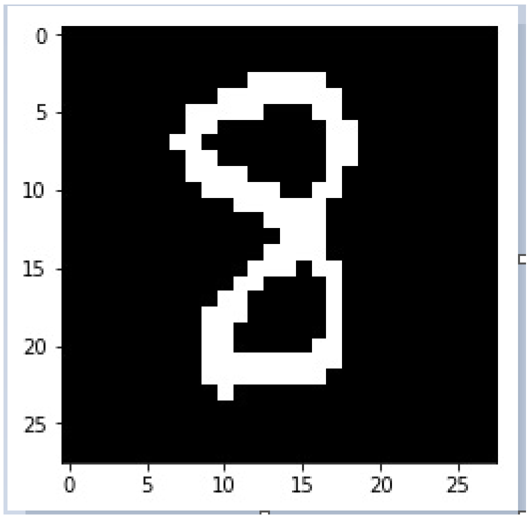


Fig 3.: Custom digit after thresholding

3.3.5 Cropping and antialiasing

Cropping is the removal of unwanted areas from a photographic or illustrated image. The process often consists the removal of the outer parts or background of an image to improve framing or change aspect ratio to accentuate or isolate the subject matter.

In our digit recognition system, the input image was UI generated, since it was possible that the digit could be drawn anywhere on the canvas, it was necessary that we get only those desired parts which was required by our system and nothing else that could cause disturbance. So, since the image could be drawn anywhere on the canvas, for example if the image was drawn at center then it would leave blank black spaces in the top, bottom, left and right side of the image. If the image was drawn at the left bottom of the canvas, it would leave blank black spaces in the top and right side of the image. This is a problem, we do not want to have any unwanted space in our image. So in order to fit the image perfectly, we chose to crop the image. This was done in Python.

Since the image only contained the values 0 and 255 it became easy for us to crop the image. As the canvas was black, the intensities of the blank spaces was always 0,and moreover all the pixel values that correspond to the image was already set to 255 by thresholding previously. So we developed an algorithm which searched all the horizontal pixel rows and the vertical pixel columns for a white pixel. What we mean by that is for every horizontal pixel row collection we looked for a white pixel, as soon as we encounter the first white pixel we stopped scanning and discarded the previous empty black pixel rows. Similarly, for every vertical pixel column we kept scanning until we found a white pixel that meant the input image has been reached and our algorithm stopped scanning further and discarded all the blank columns before the image.

All these processing and the natural character of the image caused aliasing in our image. Aliasing caused sharp edges with kind of distorted our image a bit. So, our algorithm also prevented aliasing using a built-in function provided by Python's PIL for antialiasing an image.

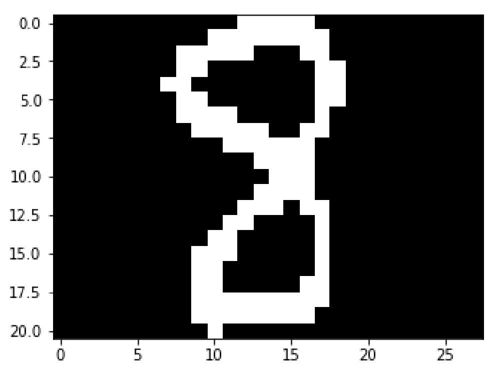


Fig 4.1.:After removal of rows

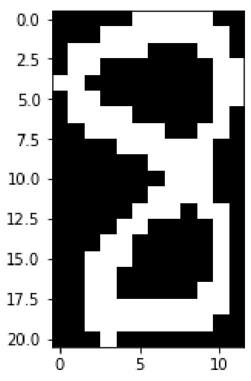


Fig 4.2.:After removal of both rows and columns

This proved to be an effective solution to remove unwanted pixels in the image background, but this caused another problem to arise which was the 28x28 image was now reduced to something less than 28x28, this was now out of standard for our system. The way we dealt with this is discussed in the next section.

3.3.6 Resizing again

Since our classifier can predict an image of dimensions[1,784], we have to resize the image to 28x28 keeping the aspect ratio. The aspect ratio is kept using the pillow module. After it is resized, the image is then converted into a numpy array which contains the value 0 or 255 denoting Black and White respectively. The numpy array is then reshaped to dimensions [1,784] using the numpy's reshape function. This numpy array is then put into our classifier for prediction.

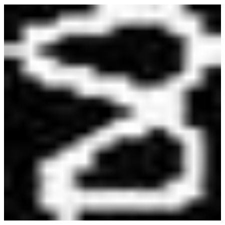


Fig 5.: Cropped image resized to 28x28 px

3.4 The Classifiers

Our English Handwritten Digit Recognition System used various classifiers for the prediction of the results. The classifiers used were Linear SVM, Polynomial SVM of degree 5, RBF kernel SVM, KNN classifier and the Random Forest Classifier. The classifiers were trained on MNIST dataset for handwritten digits and then tested in two phases-

The first test set was from the MNIST dataset itself. The train-test split was done in 50:50 ratio. The test cases were passed into the classifier and the accuracy was measured and noted.

The second test was not a test set but showed real time prediction using our UI. The image was passed through the UI to the recognition script and the results were displayed instantly as soon as the user clicked on the "Predict" button on the RHS of the UI.

The various classifiers and their working principles are shown in the subsequent sections of this report.

3.4.1 KNN Classifier

In digit recognition, the k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification. In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

In k-NN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors.

The k-NN algorithm is among the simplest of all machine learning algorithms

The neighbors are taken from a set of objects for which the class (for k-NN classification) or the object property value (for k-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.

Query point $\boldsymbol{X_i} = x_1, x_2, x_3, \ldots\ldots\ldots, x_n$

Training Sample $\boldsymbol{X_j} = x_1, x_2, x_3, \ldots\ldots\ldots, x_n$

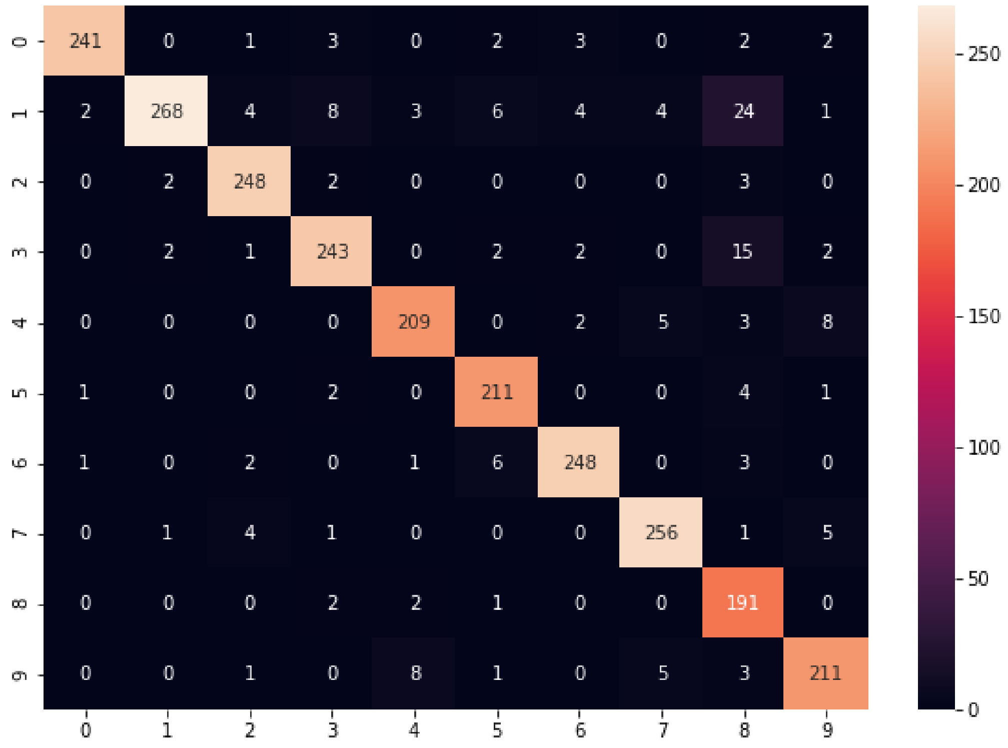
$$Dist(c_1, c_2) = \sqrt{\sum_{i=1}^{N} \left(attr_i(c_1) - attr_i(c_2)\right)^2}$$

$$k - NearestNeighbors = \{k - MIN(Dist(c_i, c_{test}))\}$$

$$prediction_{test} = \frac{1}{k}\sum_{i=1}^{k} class_i \; (or \frac{1}{k}\sum_{i=1}^{k} value_i)$$

Fig 6.: Working principle of K-NN classifier

When the KNN classifier was used on 5,000 training images and 2,500 test images, the results were displayed in a confusion matrix which is shown as follows:



	0	1	2	3	4	5	6	7	8	9
0	241	0	1	3	0	2	3	0	2	2
1	2	268	4	8	3	6	4	4	24	1
2	0	2	248	2	0	0	0	0	3	0
3	0	2	1	243	0	2	2	0	15	2
4	0	0	0	0	209	0	2	5	3	8
5	1	0	0	2	0	211	0	0	4	1
6	1	0	2	0	1	6	248	0	3	0
7	0	1	4	1	0	0	0	256	1	5
8	0	0	0	2	2	1	0	0	191	0
9	0	0	1	0	8	1	0	5	3	211

Fig 7.: Confusion matrix

We see that the classifier predicted a lot digits correctly, yet it was not good enough and could be improved using other classification techniques.

Cons: Since KNN classifier is sensitive to a lot of irrelevant attributes which affects the distances, it is not fit to use for every condition. Moreover it is computationally expensive as "n" i.e, the number of training examples grows, d also grows and system will become slower and slower. The time needed to compute distances to all examples will be of order O(nd) which is too high. In real world such a long time to classify is not feasible.

3.4.2 The Random Forest Classifier

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.Random decision forests correct for decision trees' habit of overfitting to their training set.

Random Forest is a supervised learning algorithm. We can already see from it’s name, it creates a forest and makes it somehow random. The “forest” it builds, is an ensemble of Decision Trees, most of the time trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result. To say it in simple words: Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.

One big advantage of random forest is, that it can be used for both classification and regression problems, which form the majority of current machine learning systems. I will talk about random forest in classification, since classification is sometimes considered the building block of machine learning. Below you can see how a random forest would look like with two trees:

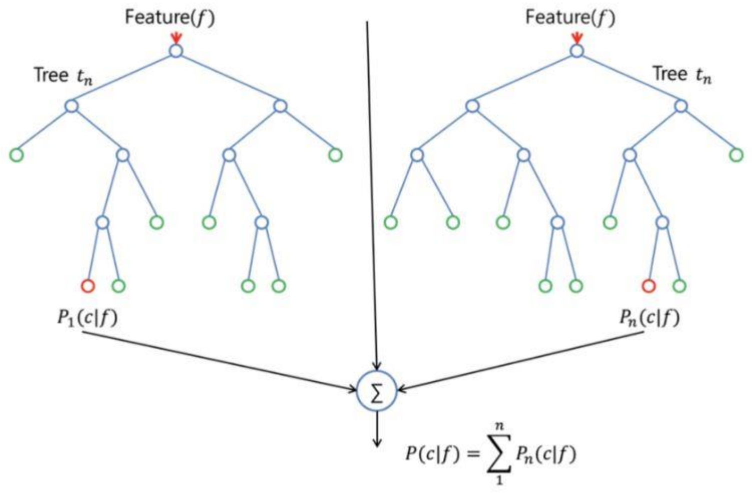


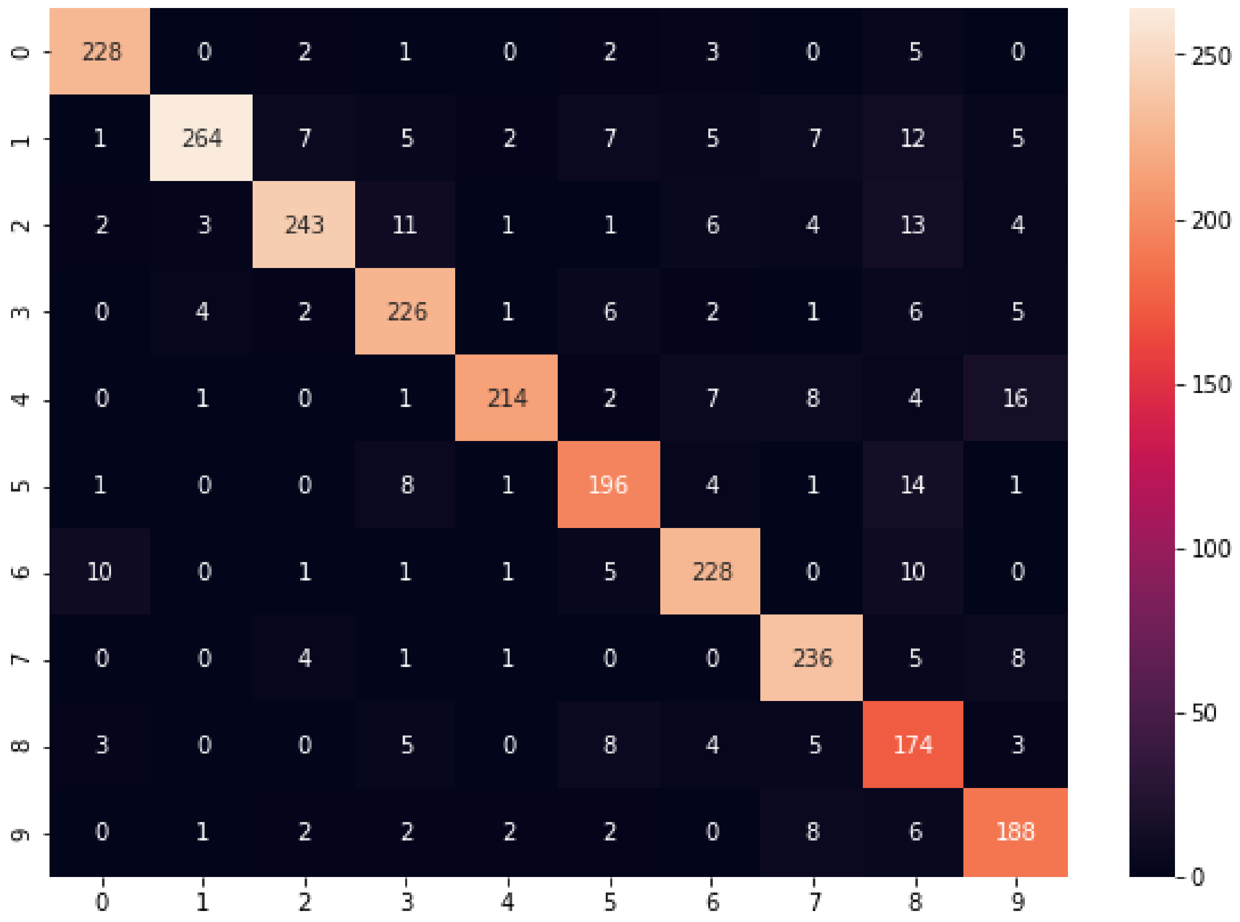
Fig 8.: The figure shows how the decision trees are created and classify image using Bayes theorem

With a few exceptions a random-forest classifier has all the hyperparameters of a decision-tree classifier and also all the hyperparameters of a bagging classifier, to control the ensemble itself. Instead of building a bagging-classifier and passing it into a decision-tree-classifier, we can just use the random-forest classifier class, which is more convenient and optimized for decision trees.

The random-forest algorithm brings extra randomness into the model, especially when it is growing the trees required. Instead of searching for the best feature while splitting a node, it searches for the best feature among a random subset of features. This process creates a wide diversity, which generally results in a better model.Our system when trained and tested on the Random Forest Classifier gave average results and the results were plotted on the confusion matrix as shown below:

In a decision tree each internal node represents a “test” on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). A node that has no children is a leaf.

Cons: The main limitation of Random Forest is that a large number of trees can make the algorithm quite slow and ineffective for real-time predictions. Even though these algorithms are fast to train, but quite slow to create predictions once they are trained. A more accurate prediction requires more trees, which results in a slower model. In most real-world applications the random forest algorithm is fast enough, but there can certainly be situations where run-time performance is important and other approaches would be preferred.



	0	1	2	3	4	5	6	7	8	9
0	228	0	2	1	0	2	3	0	5	0
1	1	264	7	5	2	7	5	7	12	5
2	2	3	243	11	1	1	6	4	13	4
3	0	4	2	226	1	6	2	1	6	5
4	0	1	0	1	214	2	7	8	4	16
5	1	0	0	8	1	196	4	1	14	1
6	10	0	1	1	1	5	228	0	10	0
7	0	0	4	1	1	0	0	236	5	8
8	3	0	0	5	0	8	4	5	174	3
9	0	1	2	2	2	2	0	8	6	188

Fig 9.: Confusion matrix for Random Forest Classifier

3.4.3 Support Vector Machines

The classifier that we used thoroughly is the Support Vector Machine.

A Support Vector Machine (SVM) performs classification by constructing an N-dimensional hyperplane that optimally separates (classifies) the data into two categories. SVM models are closely related to neural networks. In fact, a SVM model using a sigmoid kernel function is equivalent to a two-layer, perceptron neural network.

Support Vector Machine (SVM) models are a close cousin to classical multilayer perceptron neural networks. Using a kernel function, SVM's are an alternative training method for polynomial and radial basis function classifiers in which the weights of the network are found by solving a quadratic programming problem with linear constraints.

In SVM , a predictor variable is called an attribute, and a transformed attribute that is used to define the hyperplane is called a feature. The task of choosing the most suitable representation is known as feature selection. A set of features that describes one case (i.e., a row of predictor values) is called a vector. So the goal of SVM modeling is to find the optimal hyperplane that separates clusters of vector in such a way that cases with one category of the target variable are on one side of the plane and cases with the other category are on the other size of the plane. The vectors near the hyperplane are the support vectors. The figure below presents an overview of the SVM process.

Classifying data is a common task in machine learning. Suppose some given data points each belong to one of two classes, and the goal is to decide which class a new Data point will be in. In the case of support vector machines, a data point is viewed as a {p} p-dimensional vector, and we want to know whether we can separate such points with a (p-1)-dimensional hyperplane. This is called a linear classifier. There are many hyperplanes that might classify the data. One reasonable choice as the best hyperplane is the one that represents the largest separation, or margin, between the two classes. So we choose the hyperplane so that the distance from it to the nearest data point on each side is maximized. If such a hyperplane exists, it is known as the maximum-margin hyperplane and the linear classifier it defines is known as a maximum margin classifier; or equivalently, the perceptron of optimal stability.

A) The Linear Support Vector Machine

If the training data is linearly separable, we can select two parallel hyperplanes that separate the two classes of data, so that the distance between them is as large as possible. The region bounded by these two hyperplanes is called the "margin", and the maximum-margin hyperplane is the hyperplane that lies halfway between them. With proper dataset rescaling these hyperplanes can be described by the following equations:

$\vec{w} \cdot \vec{x} - b = 1$ (anything on or above this boundary is of one class, with label 1)

and

$\vec{w} \cdot \vec{x} - b = -1.$ (anything on or below this boundary is of the other class, with label -1)

The distance is computed using the distance from a point to a plane equation. We also have to prevent data points from falling into the margin, we add the following constraint: for each i either:

$$\vec{w} \cdot \vec{x}_i - b \geq 1, \text{ if } y_i = 1$$

or

$$\vec{w} \cdot \vec{x}_i - b \leq -1, \text{ if } y_i = -1.$$

These constraints state that each data point must lie on the correct side of the margin.This can be rewritten as:

$$y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1, \quad \text{for all } 1 \leq i \leq n.$$

We can put this together to get the optimization problem:

"Minimize $\|\vec{w}\|$ subject to $y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1$, for $i = 1, \ldots, n$"

The $\vec{w}$ and b that solve this problem determine our classifier, $\vec{x} \mapsto \operatorname{sgn}(\vec{w} \cdot \vec{x} - b)$.

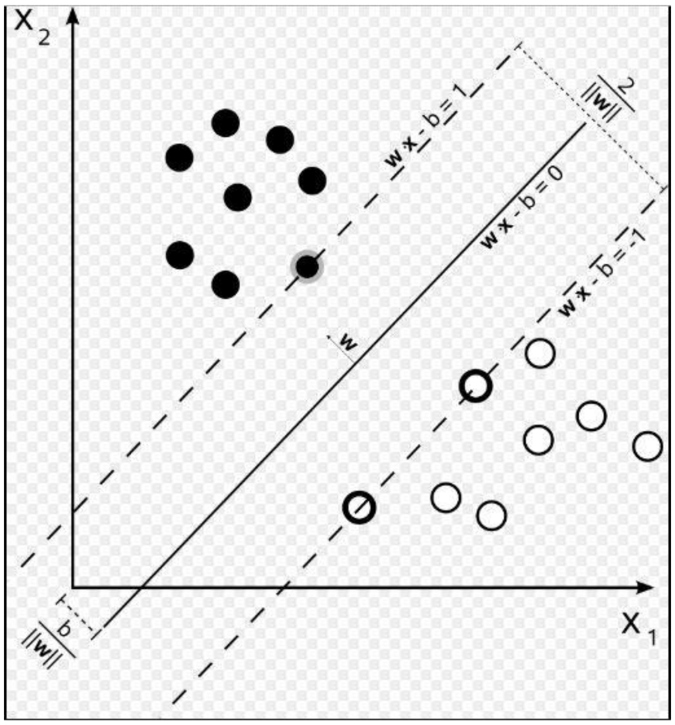


Fig 10.: Linear SVM

The confusion matrix obtained after the linear SVM classifier was fit with 5000 training images and 2500 test images is shown as follows:

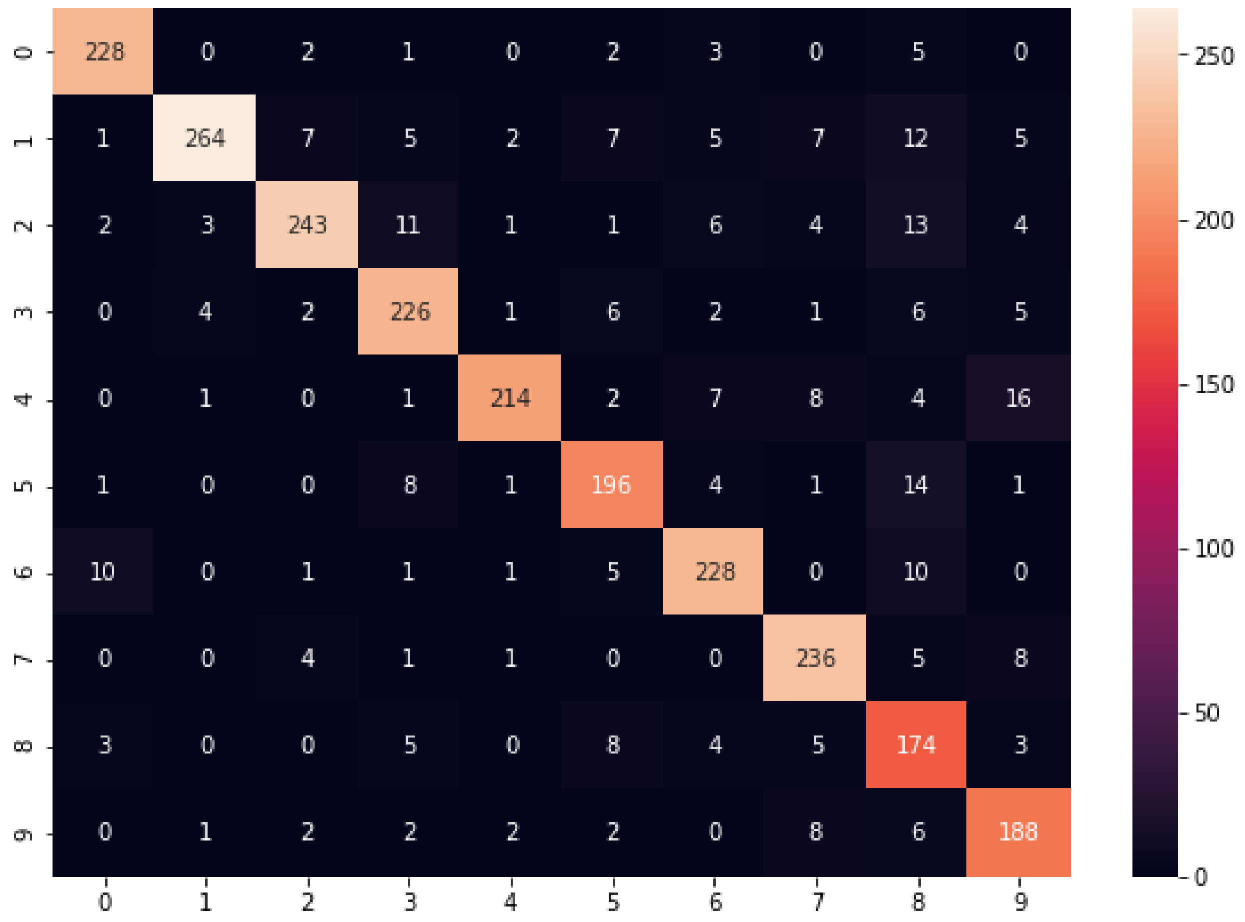


Fig 11.: Confusion matrix for Linear SVM

B) The RBF kernel

In machine learning, the radial basis function kernel, or RBF kernel, is a popular kernel function used in various kernelized learning algorithms. In particular, it is commonly used in support vector machine classification. The RBF kernel on two samples, represented as feature vectors in some input space.

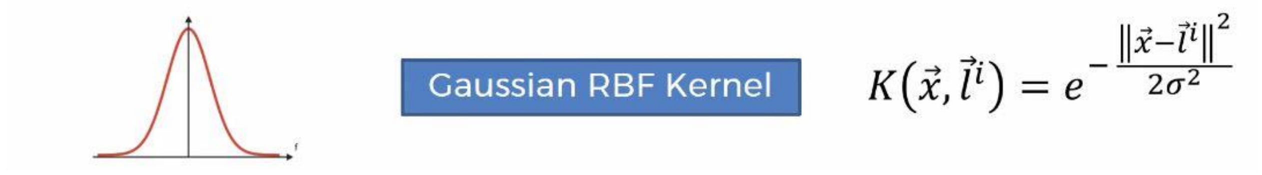

Gaussian RBF Kernel
$K(\vec{x},\vec{l}^i) = e^{-\frac{\|\vec{x}-\vec{l}^i\|^2}{2\sigma^2}}$

Fig 11.: The working principle of RBF kernel SVM

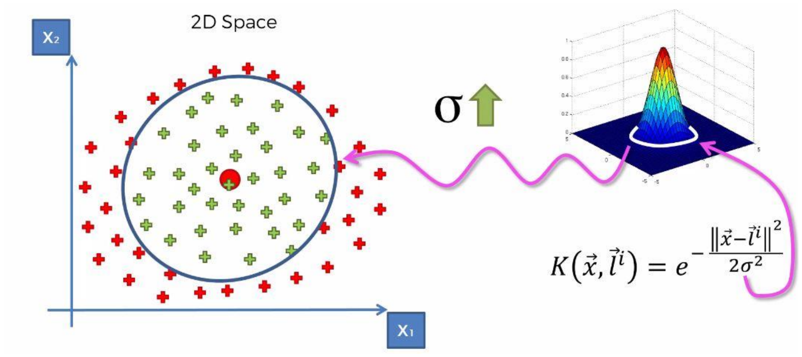

2D Space
X₂
X₁
σ
$K(\vec{x},\vec{l}^i) = e^{-\frac{\|\vec{x}-\vec{l}^i\|^2}{2\sigma^2}}$

Fig 12.: The mapping of the Gaussian with the samples

C) The Polynomial SVM

There are many instances when the linear classifiers fail to fit and classify the data when they are not linearly separable, so a polynomial classifier is required. The polynomial classifier fits as perfectly as possible around the data.

The polynomial classifier's working principle and the how it separates the data is shown below as follows:

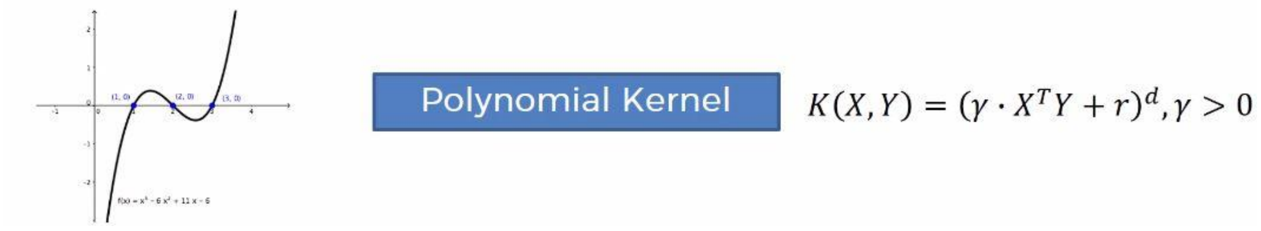
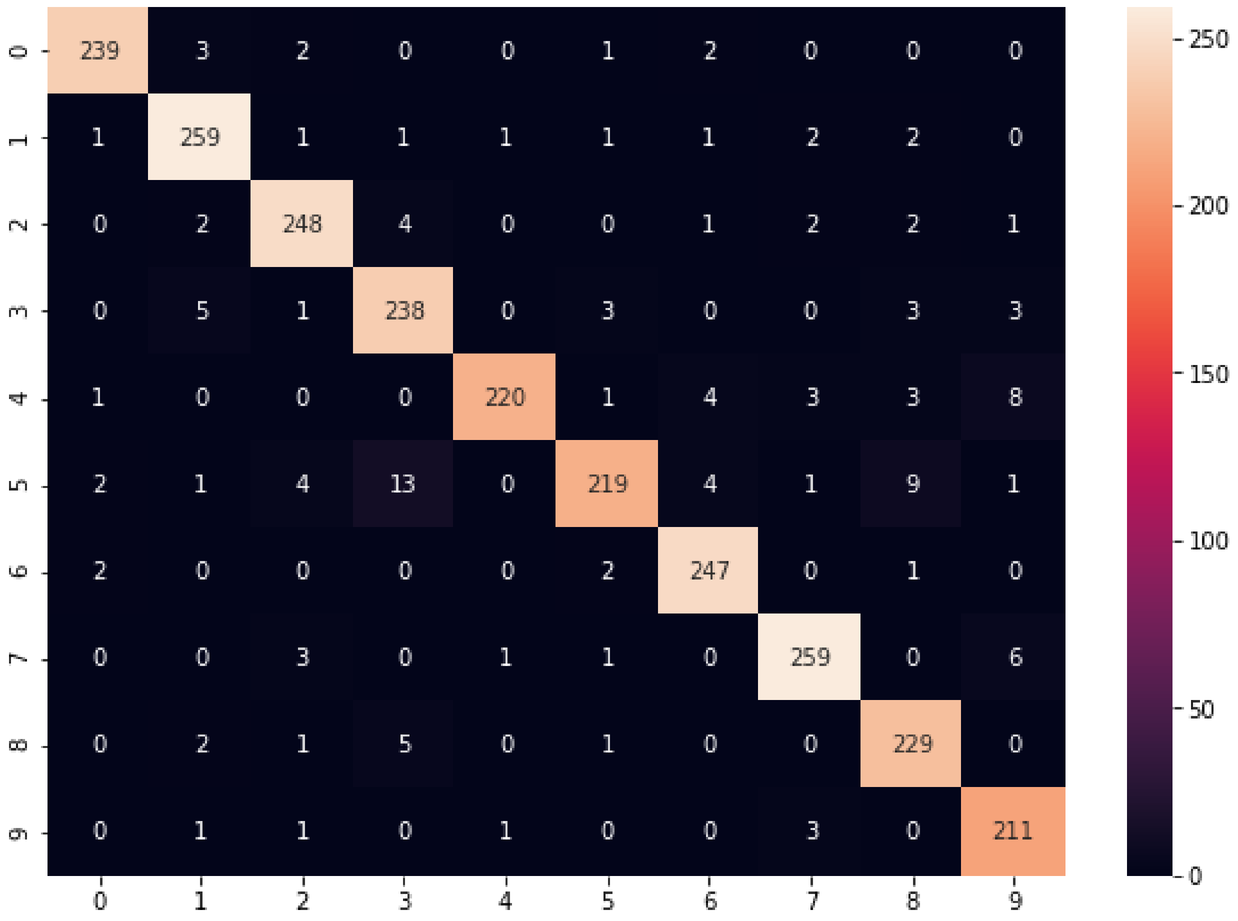

(1, 0)
(2, 0)
(3, 0)
f(x) = x³ − 6x² + 11x − 6
Polynomial Kernel
$K(X,Y) = (\gamma \cdot X^T Y + r)^d, \gamma > 0$

Fig 12.: Working principle of Polynomial SVM

When we used the SVM polynomial classifier of degree 5 and trained and tested our MNIST dataset, we got the following results which was shown in the confusion matrix:



	0	1	2	3	4	5	6	7	8	9
0	239	3	2	0	0	1	2	0	0	0
1	1	259	1	1	1	1	1	2	2	0
2	0	2	248	4	0	0	1	2	2	1
3	0	5	1	238	0	3	0	0	3	3
4	1	0	0	0	220	1	4	3	3	8
5	2	1	4	13	0	219	4	1	9	1
6	2	0	0	0	0	2	247	0	1	0
7	0	0	3	0	1	1	0	259	0	6
8	0	2	1	5	0	1	0	0	229	0
9	0	1	1	0	1	0	0	3	0	211

Fig 13.: Confusion matrix for Polynomial SVM

4.Results and Discussions

After testing and running our custom digits and MNIST dataset we found that the best classifier for the Handwritten Digit Recognition to be the Polynomial Support Vector Classifier of degree 5. The accuracy was found to be 94.76% which was the highest amongst the classifiers compared. The following table shows the results comparing the various classifiers with the accuracy and error rate:

We found that as previously known the KNN and Random Forest Classifiers were slower and took more time to fit and predict the test results, whereas almost all the SVM classifiers were faster and gave above 90% accuracy. The best classifier was the SVM polynomial classifier with degree 5.

5. Summary and Conclusions

In this paper, we have described a simple approach for processing and classification for English Handwritten digit recognition. Five classifiers were used for the recognition process each of which classified the digits into 9 categories based on their respective results. The overall performance was 90% for the recognition of custom digits and 94.7% for the 2,500 MNIST test digit images. In the future work, an online implementation of the same and we will build an application that could recognize a string of digits, not just a single digit. It will be an implementation of complete Optical Character Recognition system

References

● [1] https://homes.cs.washington.edu/~dericp/assets/projects/digit-recognition/writeup.pdf

● [2] http://yann.lecun.com/exdb/mnist/

●[3]https://www.researchgate.net/publication/234791450_Multiclass_SVM_classifier_for_english_handwritten_digit_recognition_using_manual_class_segmentation

● [4] https://thesai.org/Downloads/Volume2No10/Paper%206-A%20Statistical%20Approach%20For%20Latin%20Handwritten%20Digit%20Recognition

● [5] https://scikit-learn.org/stable/modules

● [6] https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd

● [7] https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

● [8] https://www.kaggle.com/c/digit-recognizer

● [9] Maji, Subhransu, and Jitendra Malik. ”Fast and accurate digit classification.” EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-159 (2009).