Handwriting Recognition

-VISION QUEST
Manne Naga Nithin 14EE10026
Himanshu Chaudhary 14IE10013
Rajasekhar Mekala 14EE10027
Ashish PVJS 14EE10052
Arun Chandra Ghantasala 14EE10017
Kasaraneni Sai Hemanth 14EE30012
Vaddi Veda Vyshnavi 14EC10058

Problem Definition

Recognize handwritten digits and alphabets from a sequence of combined data and compare the results using various standard machine learning algorithms.

Methodology

In this project, for recognizing handwritten data, we decompose the problem statement into two main parts which are segmenting the sequence and recognizing individual character or digit. For digit recognition, we used the standard MNIST[1] data set and built in OpenCV dataset, and for character recognition we used the UCI Letter Recognition Dataset[2] and The Chars 74K dataset[3].

Dataset Description

MNIST Dataset

The MNIST database of handwritten digits 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.The original black and white (bilevel) images from NIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. the images were centered in a 28x28 image by computing the center of mass of the

pixels, and translating the image so as to position this point at the center of the 28x28 field.

The Chars74K dataset

In this dataset, symbols used in both English and Kannada are available. In the English language, Latin script (excluding accents) and Hindu-Arabic numerals are used. Our dataset consists of:

- 64 classes (0-9, A-Z, a-z)
- 7705 characters obtained from natural images
- 3410 hand drawn characters using a tablet PC
- 62992 synthesised characters from computer fonts

This gives a total of over 74K images (which explains the name of the dataset), but we are only using the hand drawn characters.

UCI Letter Recognition Dataset

Data Set Characteristics:	Multivariate	Number of Instances:	20000	Area:	Computer
Attribute Characteristics:	Integer	Number of Attributes:	16	Date Donated	1991-01-01
Associated Tasks:	Classification	Missing Values?	No		

The character images were based on 20 different fonts and each letter within these 20 fonts was randomly distorted to produce a file of 20,000 unique stimuli. Each stimulus was converted into 16 primitive numerical attributes (statistical moments and edge counts) which were then scaled to fit into a range of integer values from 0 through 15.

Attribute Information

1. lettr capital letter (26 values from A to Z)
2. x-box horizontal position of box (integer)
3. y-box vertical position of box (integer)
4. width width of box (integer)
5. high height of box (integer)
6. onpix total # on pixels (integer)
7. x-bar mean x of on pixels in box (integer)
8. y-bar mean y of on pixels in box (integer)
9. x2bar mean x variance (integer)
10. y2bar mean y variance (integer)
11. xybar mean x y correlation (integer)
12. x2ybr mean of x * x * y (integer)
13. xy2br mean of x * y * y (integer)
14. x-ege mean edge count left to right (integer)
15. xegvy correlation of x-ege with y (integer)
16. y-ege mean edge count bottom to top (integer)
17. yegvx correlation of y-ege with x (integer)

OpenCV Sample Dataset

OpenCV comes with an image digits.png (in the folder opencv/samples/python2/data/) which has 5000 handwritten digits (500 for each digit). Each digit is a 20x20 image. So our first step is to split this image into 5000 different digits. For each digit, we flatten it into a single row with 400 pixels.

Input Preprocessing

Image Segmentation

Here we are using simple segmentation methods for recognizing the characters in the given image which is the main difference or challenge in recognizing cursive handwriting.

The image is first converted to grayscale image followed by the threshing technique, which make the image become binary image. The binary image is then completed for small holes through dilation technique and sent through connectivity test in order to check for the maximum connected component, which is, the box of the form. After locating the box, the individual characters are then cropped into different sub images that are the raw data for the following feature extraction routine.

The size of the sub-images are not fixed since they are exposed to noises which will affect the cropping process .This will causing the input of the network non-standard. So we ensure that all the images are of fixed size.Thus we would be able to extract the most important features of each characters or use these images for Neural networks where each of these pixels of the image are taken as input in a proper vector form.
Steps in image segmentation are:
1- Read the image to MATLAB workspace
2- Convert to grayscale
3- Convert to binary image
4- Edge detection
5- Morphology
At the step image dilation and Image filling is performed.
6- Blob analysis
At this step, all the objects on the image and all the properties of each object are found.
7- Plot the object location.
At this step, the location of each object is plotted.

Algorithms Implemented

k-Nearest Neighbour

KNN has some nice properties: it is automatically non-linear, it can detect linear or nonlinear distributed data, it tends to perform very well with a lot of data points. On the minus side KNN needs to be carefully tuned, the choice of K and the metric (distance) to be used are critical. If you are in a very low dimensional space you can use a RP-Tree or KD-Tree to improve performance, if you have a higher number of dimensions then you need an approximation to the nearest neighbors problems and whenever we use an approximation we have to think if KNN with the NN approximation is still better than other algorithms. KNN is also very sensitive to bad features (attributes) so feature selection is also important. KNN is also sensitive to outliers and removing them before using KNN tends to improve results.
Here we built the kNN algorithm from the scratch and the distance measured is the euclidean distance.For each test data we check all the neighbouring distances and then sort all of them to find the k nearest neighbours.A part of the dataset is used for training and the remaining for testing which can be manually

assigned in the knn function.we observed as the number of nearest neighbours are increased we observed a decrease in the accuracy.

Results

MNIST Dataset

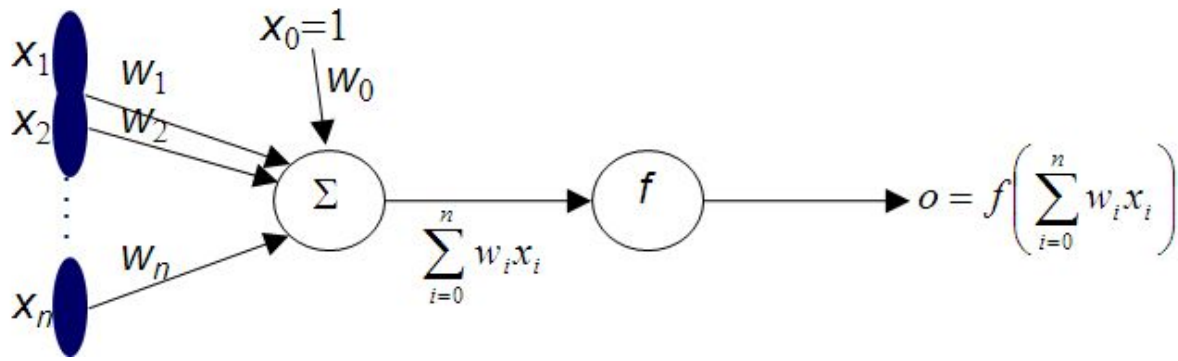
Accuracy:-

Char 74 Dataset

Accuracy on Combined (Alphabets+Digits):-**67.49%**
Accuracy on just Capital Aplhabets:-**76.64%**
Accuracy on just Small Alphabets:-**73.54%**

Neural Networks

As we are working with recognition problem i.e too much features are involved we need to have more complex algorithms compared to basic algorithms like KNN.



An Artificial Neural Network (ANNs) form the basis of an OCR which is trained using the Backpropagation algorithm. After converting the handwritten English characters into 28x28 matrices as explained earlier, these matrices can be fed to the ANN as input. After the Feed Forward Algorithm which gives workings of a neural network, the Backpropagation Algorithm performs Training, Calculating Error, and Modifying Weights. For Backpropagation, we are using Stochastic Gradient Descent method. The BP algorithm starts with computing the output layer, which is the only one where desired outputs are available. The error rate in the output layer is calculated based on the difference between the desired output and the actual output.

Structure

The Back Propagation Neural Network implemented for the purpose of this project is composed of 3 or more layers, one input, one or more hidden and one output. For the 28x28 matrices, the input layer has 784 neurons, the hidden layers have 30 neurons, (the number of neurons in the hidden layer has been determined by trial and error) and the output layer has 10 or 26 neurons. The output layer is in fact a competitive layer (only one bit of the output becomes 1 for each class). For this project, the sigmoid function has been used as a non-linear neuron activation function :

$$f(x) = \frac{1}{1+e^{-x}}$$

Bias terms with trainable weights were also included in the network structure.The biases and weights for the network are initialized randomly, using a Gaussian distribution with mean 0, and variance 1.

Results

MNIST Dataset

Batch size=10 Hidden layer=1, Number of neurons=30

Learning Rate	Accuracy(after 10 epochs)

0.01	45.86%
0.1	80.44%
1	94.89%
3	95.86%
10	91.25%

Chars74 Dataset

(Batch Size=5 ,Hidden Layers=3,Number of neurons=30,Learning Rate = 2)

For capital letters
:Accuracy=**86.45%**
For small letters
:Accuracy=**81.58%**
For combined data
:Accuracy=**67.97%**

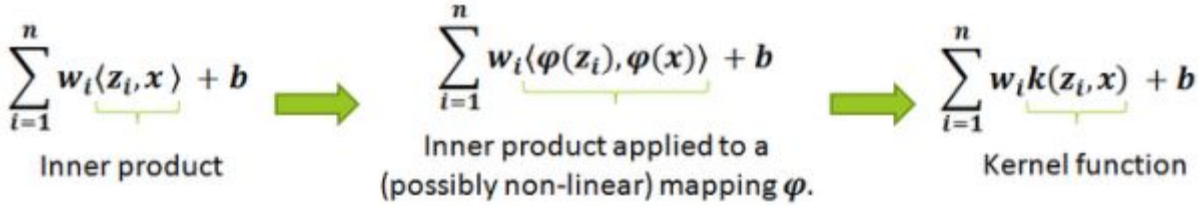
SVM(Support Vector Machines)

SVM is the use of supervised learning with associated learning algorithms that analyze data used for classification and regression analysis. Using SVM training on different datasets we adopted a standard algorithm for dividing them into various categories.We used Non Linear Classification based on RBF kernel. Using SVM model we represented the examples as points in space, mapped them into categories with a good margin by providing large number of Data Sets. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier and increasing the accuracy when Test Data sets are used. A linear support vector machine is composed of a set of given support vectors **z** and a set of weights **w**. The computation for the output of a given SVM with N support vectors z_1, z_2, ... , z_N and weights w_1, w_2, ... , w_N is then given by:

$$F(x) = \sum_{i=1}^{N} w_i \ \langle z_i, x \rangle + b$$

A decision function is then applied to transform this output in a binary decision. Usually, sign(.) is used, so

that outputs greater than zero are taken as a class and outputs lesser than zero are taken as the other.Using a Kernel function, the algorithm can then be carried into a higher-dimension space without explicitly mapping the input points into this space. This is highly desirable, as sometimes our higher-dimensional feature space could even be infinite-dimensional and thus infeasible to compute. Since the original formulation of SVMs mainly consists of dot products, it is straightforward to apply the Kernel trick. Even if the resulting classifier is still a hyperplane in the high-dimensional feature space, it may be nonlinear in the original input space. The use of the Kernel trick also gives a much stronger theoretical support to the underlying classifiers when in comparison with methods which have different source of inspiration, such as biology.



Support Vector Machines do not generalize naturally to the multi-class classification case. SVMs are binary classifiers, and as such, in their original form, they can only decide between two classes at once. However, many approaches have been suggested to perform multi-class classification using SVM's.

Results

MNIST Dataset

-:Accuracy=**94.35%**

Chars74

For Capital letters
:Accuracy-95.68%
For Small letters
:Accuracy-94.21%
For combined data
:Accuracy-75.94%