**HAND WRITTEN DIGIT RECOGNITION**

**AIM**

To identify hand written digits ranging from one to nine based on a pre existing data set containing pixel data for the images in the form of a .csv dataset.

**LIBRARIES USED**

The following libraries were used -

*NumPy*: NumPy is a library for the Python programming language, which adds support for large, multi-dimensional arrays and matrices and their associated operations.

*Matplotlib*: Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy

*Pandas*:  It is library written for the Python programming language for data manipulation and analysis

**DATA SET**

The data being used has been taken from Kaggle. The data files train.csv and test.csv contain grey-scale images of hand-drawn digits, from zero through nine. Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive. (Link: https://www.kaggle.com/c/digit-recognizer/data)

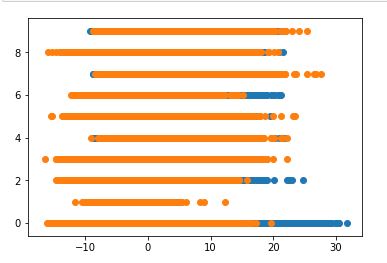
**DATA PROCESSING AND PLOTTING**

The labels and features of the data are extracted separately and stored. *Principal Component Analysis* is used to implement dimensionality reduction of the data. The number of features is reduced from 28000 to 2, for better efficiency and processing of data. This also helps in reduction of the error in the final result.

Principal Component Analysis

Principal Component Analysis (PCA) is a statistical procedure that orthogonally transforms the original n coordinates of a data set into a new set of n coordinates called principal components. As a result of the transformation, the first principal component has the largest possible variance; each succeeding component has the highest possible variance under the constraint that it is orthogonal to (i.e., uncorrelated with) the preceding components.

The dimensionally reduced data is then plotted using the matplotlib library and a graphical representation is obtained (as shown below).

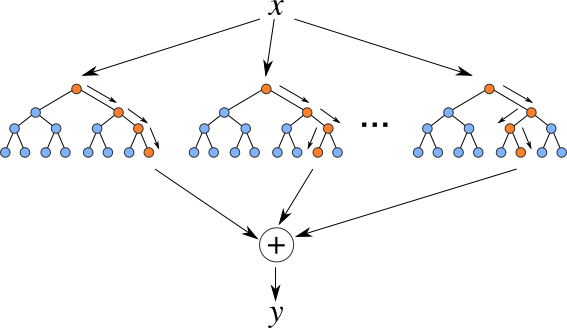


**LEARNING TECHNIQUE**

The training data contains both labels and features. The testing data contains only features and the appropriate labels have to be predicted. *Random Forest Algorithm* is a [supervised learning](https://en.wikipedia.org/wiki/Unsupervised_learning) model and uses labelled data to “learn” how to classify unlabelled data. The Random Forest Classifier is trained upon the training data set (train.csv) and then is used to predict the labels in the testing dataset (test.csv)

Random Forest Algorithm

In the figure given below, three individual decision trees make up a Random Forest. The R[andom Forest](https://en.wikipedia.org/wiki/Random_forest) Algorithm is usually composed of different decision trees, each with the same nodes, but using different data that leads to different leaves. The algorithm uses the final decisions of each node to come to a conclusion of its own. This helps in increasing accuracy as it looks at decisions of different trees and formulates an average value.



This algorithm is extremely helpful with various types of data sets. It can be used easily, is fast to train and finds results with great accuracy.

Key Points

The number of features considered for training the model after reduction is 2

The number of estimators used in the Random Forest classifier is 100

The training accuracy of the model obtained using ‘model.score’ returned 0.999976190476

**RESULT**

The output is obtained in the ‘array([x], dtype = int64)’ format, where x is the number being predicted. The result obtained is very accurate, as we have used PCA and Random Forest Algorithm to reduce the errors in processing and calculation.