Face & Digit Classification
CS520 : Intro To AI - (Final Project)

Sai Mounica Pothuru (RUID: 198008261)
Chaitanya Sharma Domudala (RUID: 198009015)

December 13, 2020

Introduction

OCR known as Optical Character Recognition or Optical Character Reader is a technology developed for converting images of typed, handwritten or printed text into machine-encoded text, the text being from a scanned document, a photo of a document or any relevant sources. It is a common method of digitizing printed texts so that they can be electronically edited, searched, stored more compactly, displayed on-line, and used in machine processes.

In this project, we are going to implement Face & Digit Classification using Naive Bayes, Perceptron and KNN classifiers. Applying these classifiers,both the digit and face recognition is gradually trained with the corresponding images and labels and the accuracy is tested against the complete test data set.

Feature Extraction

A custom function is built to read the train and test data sets of Digit & Face data and implements the following steps:

- Reads through data & split each line at linebreak.
- The pixel size is set to 28x28 for Digit data and 70x60 for Face data. Each line of data is read and a list is generated corresponding to the type of data and its pixel size.
- The final list is composed of the number of images with each image being a list entry.

The above procedure is followed for both train and test data sets and the data is stored in corresponding lists.

Classifiers Implementation

Three classifiers, Naive Bayes, Perceptron and KNN algorithms are executed on the Digit & Face data sets. The implementation is described as below.

Naive Bayes Classifier

The Naive Bayes classifier is devised based on the assumption that the features are independent to each other, the probability equation used is as below.

$$P(Y|X) = P(X = x_i|Y = y) * P(Y = y)$$

Our training function follows the below steps:

- Calculate the priors of the data (P(Y=y))
- Calculate the probability of each pixel being a feature.

The probabilities and priors are used by the testing function and follows the below steps:

- Compute the probability for each test image with the trained probabilities and priors.
- Get the predicted result and compare with the original result.
- Implement the process to all the images and calculate the accuracy

Perceptron Classifier

The Perceptron classifier is implemented using different weights and bias. Below is the linear function used.

$$Z = W^T X + B$$

argmax{Z} is used as the tie breaker.

Our training function follows the below steps:

- Initialize the weights and bias
- Calculate the Z value given the weights and bias.
- Update the weights and bias based on the prediction

The weights and bias are used by the testing function and follows the below steps:

- Compute the Z function value for each test image with the trained weights and bias.
- Get the predicted result and compare with the original result.
- Implement the process to all the images and calculate the accuracy

KNN Classifier

KNN is devised to use the euclidean distance between train image and test image to compute the similarity and predict the image. The implementation is as below:

- Data is modified to contain only numbers for easier implementation of the algorithm.
- The complete training image and a single test image are sent as parameters to the function.The number of nearest neighbors is given as 6.
- The algorithm computes the distance between the test image and all the training images and gives the k nearest neighbors.
- The labels with highest count is given as predicted result
- The result is compared against the original image
- The procedure is repeated for all the test images and the accuracy is computed.

Observations

We tested all three Classifiers on 10%,20%,30%,40%,50%, 60%,70%,80%,90%,100% of the total training data size. Calculated the accuracy and runtime on 100% of the total test data size.

It is observed that the accuracy doesn't necessarily increase with the training size but the runtime is directly proportional to the training size which is expected. So it is not absolutely necessary to take a large training set.

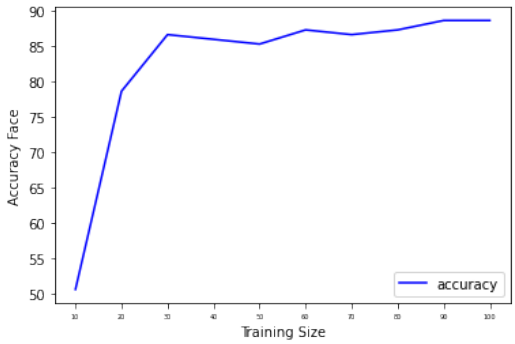

Figure 1: Naive Face Accuracy

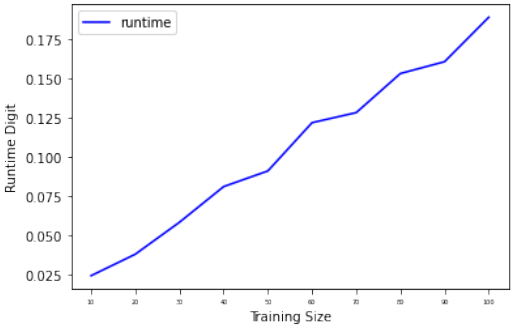

Figure 2: Naive Face Runtime

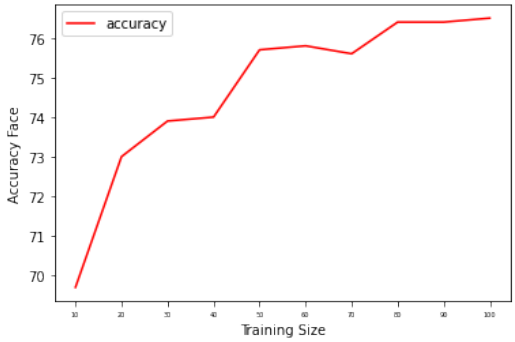

Figure 3: Naive Digit Accuracy

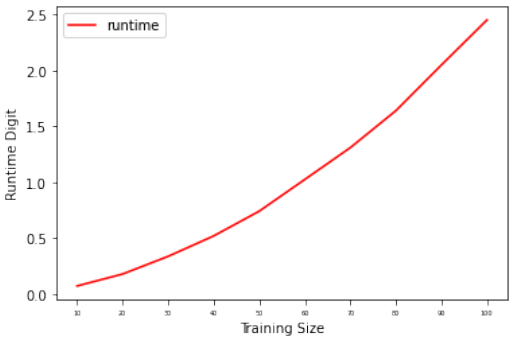

Figure 4: Naive Digit Runtime

Figure 5: **Naive Bayes Observations**

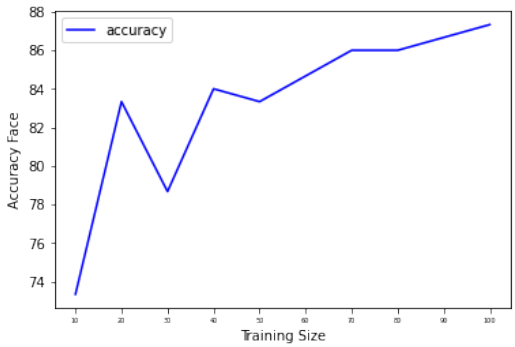

Figure 6: Perceptron Face Accuracy

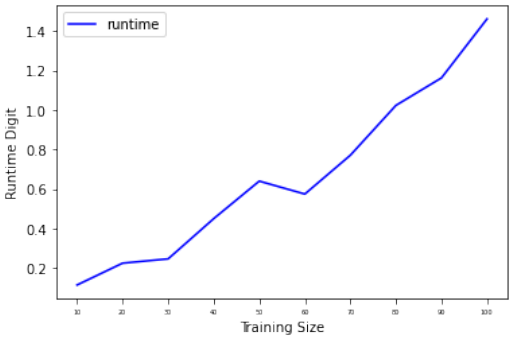

Figure 7: Perceptron Face Runtime

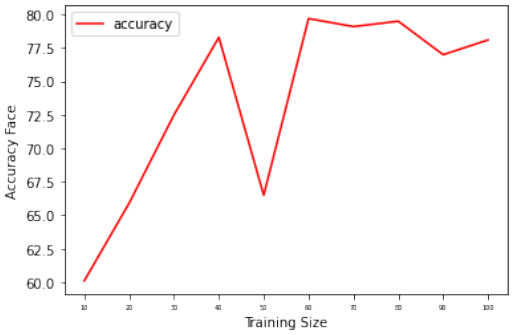

Figure 8: Perceptron Digit Accuracy

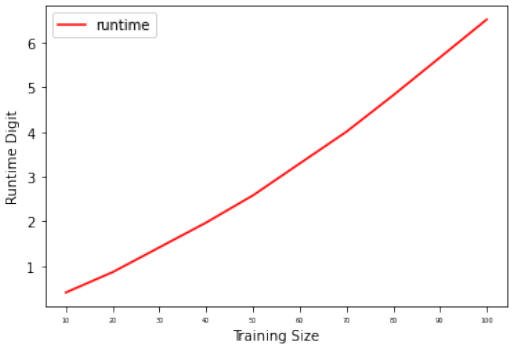

Figure 9: Perceptron Digit Runtime

Figure 10: **Perceptron Observations**

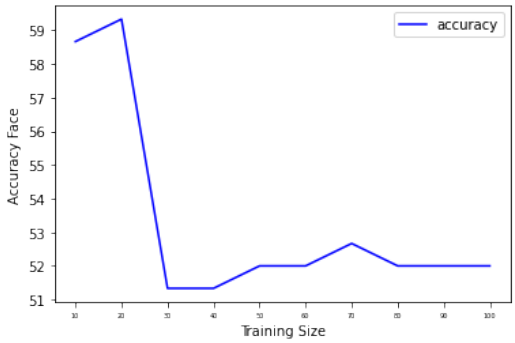

Figure 11: KNN Face Accuracy

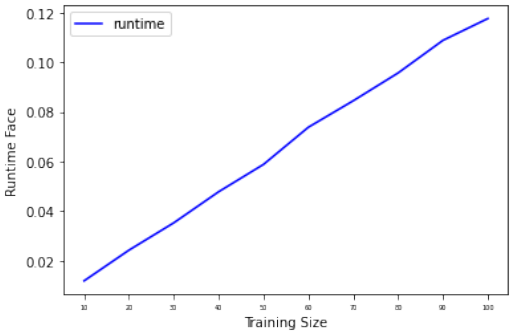

Figure 12: KNN Face Runtime

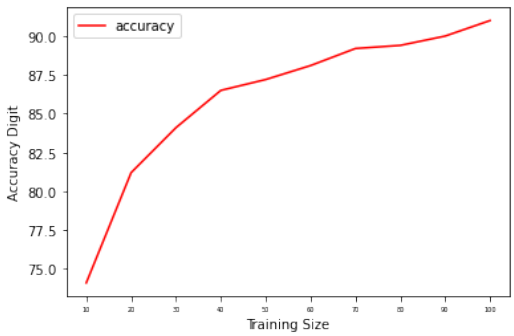

Figure 13: KNN Digit Accuracy

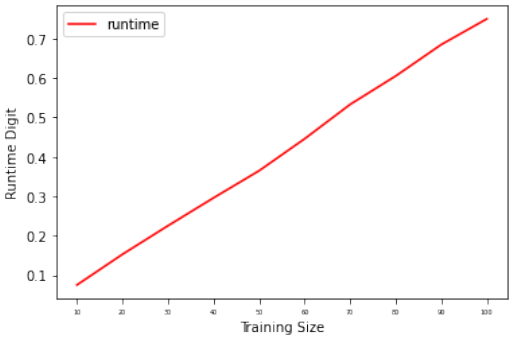

Figure 14: KNN Digit Runtime

Figure 15: **Knn Observations**

References

[1] https://www.youtube.com/watch?v=R2XgpDQro9k

[2] https://medium.com/swlh/image-classification-with-k-nearest-neighbours-51b3a289280

[3] https://medium.com/@YearsOfNoLight/intro-to-image-classification-with-knn-987bc112f0c2

[4] https://towardsdatascience.com/algorithms-from-scratch-naive-bayes-classifier-8006cc691493

[5] https://towardsdatascience.com/implementing-naive-bayes-algorithm-from-scratch-python-c6880cfc9c41

[6] https://medium.com/hackernoon/implementing-the-perceptron-algorithm-from-scratch-in-python-48be2d07b1c0