RUTGERS

Project

CS520

Face and Digit Classification

Presented By:

Vraj Shah

Parva Shah

May 3rd, 2020

Introduction

Code and dataset can be found here: https://github.com/vraj152/projectcs520

In this project, we implemented 3 algorithms on two data sets to generate a model that would fit the data with least cost function possible and predict the output for new unseen data. The algorithms are Bayesian Network, Perceptron and MIRA, which is an updated and improved version of Perceptron. In the process, we learnt the pros and cons of each algorithm.

In the first data set, we were given images of hand written digits and their corresponding labels. Our task was to design a model that would predict the digit (from 0 to 9) based on the image. The other data set comprised of images of face and other random things. The corresponding labels were 1 if the image was of a face and 0 if not. The main idea was to implement algorithm that would accurately predict through image if it was a face or not.

Also, we studied the effect of the training data on the cost function and the predicted accuracy. We played around the training data by randomly selecting only few percentages of the data and passing it through the model. This had a huge effect on the predicted results.

Preprocessing

The training and testing data were given in a text file where one file contained the images and the other had corresponding labels. The first step was to extract features from the images that would be used to train the 3 models.

The features were grey and black pixel values. We create a list that would store the value for each pixel. If the pixel was black, the black pixel value would be 1 and the grey pixel value would be 0 and vice-versa.

Similarly, the program for feature extraction is written in such a way that based on the input and dimension of features required, it can return the grey and black values for each feature. For example: If feature dimension is set to 2x2 grid then in such a grid, it would count the total number of grey pixels and return the sum. A similar task would be performed for black pixels.

Algorithms:

1 Naïve Bayesian Network

According to Bayes rule,

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

where, x represents the features or the pixel values, c is the label values.

The main idea is to find the probability of a label given some set of features. Our job is to find the largest probability among all the probabilities. This largest probability will give us a predicted label value. It will be a digit in case of digit recognition system and predict if it is a face or not in face detection system.

$$c^* = argmax\ P(c|x) = P(c|x) = argmax\ \frac{P(x|c)P(c)}{P(x)}$$

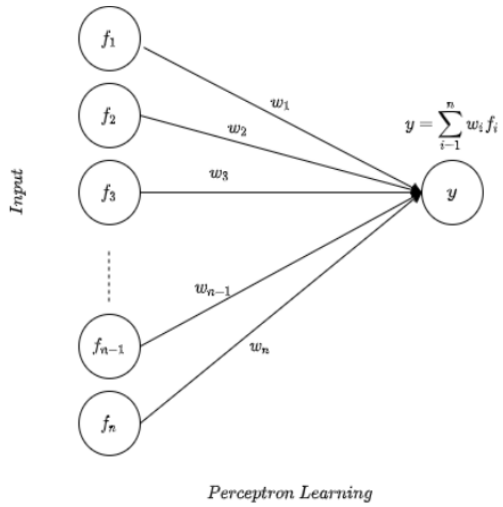
Since $P(x)$ will be the same for all the probability computations, we can ignore it will as it is divided in all the equations. This will not affect our choice for selecting the highest probability.
In the above equation $P(c)$ can be calculated easily by simply counting the number of times a particular label occurs and dividing it with the total number of examples we have.

2 Perceptron Neural Network

We also can use Perceptron for classification. It is used when classes which we want to classify are linearly separable. When classes are not linearly separable (for example: XOR), we should use Neural Network.

For the digit data we have 10 classes in total and for the face data we have only 2 classes. So, in order to classify digits we need to create 10 perceptrons and for face classification only two classes are needed.

2.1 Architecture:



Perceptron Learning

In above image, architecture of perceptron network with single perceptron is illustrated. As shown, input of network is features of a particular image. Each feature has its own weight. Hence, we have n features and n weights. The output of network will be summation of multiplication of each feature and its corresponding weight.
In other words,

$$Y = \sum_{i=1}^{n} w_i \cdot f_i$$

As we stated earlier perceptron is used when classes are linearly separable, we can predict label based on the value of Y.

$$Y = \begin{cases} 1, & \text{if } \sum_{i=1}^{n} w_i \cdot f_i > 0 \\ 0, & \text{otherwise} \end{cases}$$

Now, if we predict the correct label, we won't make any changes into weights associated with the features but if we mistakenly identify incorrect label, we will have to make appropriate changes into weights. We will make changes in following manner.
Let's say Y is predicted label and Y^* is actual label;

If $Y < 0$ and Y^*=$True$:

$$W(Y) = W(Y) + f(j), \ where \ j \in \{1, \ldots, n\}$$

If $Y > 0$ and Y^*=$False$:

$$W(Y) = W(Y) - f(j),\ where\ j \in \{1, \ldots, n\}$$

2.2 Weight updates in case of multiclass perceptron:

In digit classification we have 10 classes *i.e.* from 0 to 9 so we will have 10 perceptrons. Again we do not make changes in weights if network does not make mistake. We only make changes where network makes any mistake.

If correct label is Y^* and we have predicted Y then:

Increase weight of actual label:

$$W(Y^*) = W(Y^*) + f(j),\ where\ j \in \{1, \ldots, n\}$$

Decrease weight of wrongly identified label:

$$W(Y) = W(Y) - f(j),\ where\ j \in \{1, \ldots, n\}$$

When we make changes to weights, it slowly gets converged where it makes very minimal training error. We randomly initialize weights. Irrespective of initial weights, it gets converged. However, if initialization is not done properly, then it may get longer time to get converged.

3 MIRA

The Perceptron suffers from the below issues:

- Due to the presence of some noise or anomalous data, the weights might thrash.
- Sometimes the Perceptron model finds barely separable solutions.
- Overtraining of training data may result in test accuracy falling down as shown in the figure.

The main aim is to try to mitigate the above issues by choosing an update size and minimizing the updates on the weights. Margin Infused Relaxed Algorithm achieves the above task by introducing a variable named τ.

τ is defined as below:

$$\tau = \frac{(w'_y - w'_{y^*}) \cdot f + 1}{2f \cdot f}$$

where, w'_y = Weights of Predicted Label
w'_{y^*} = Weights of True Label
f = Features

Unlike perceptron, where the weights of the predicted label are updated by by simply subtracting themselves with the features, in MIRA they are updated as below:

$$w_y = w'_y - \tau \cdot f(x)$$
$$w_{y^*} = w'_{y^*} + \tau \cdot f(x)$$

Here, w_y = Updated weights of predicted labels
w_{y^*} = Updated weights of true labels

Experiments

We randomly selected different percentages of training data of each of the data sets and algorithms. This helped to plot different accuracy and error rates with respect to the training data selected. We began by choosing only 10 % of the data and move towards 100 % of the data.
This study shows the importance of training data on the accuracy. We came to conclusion that if you decrease the training data, accuracy gets decreased. However, if model tries to overfit then increasing the training data can help improve the accuracy of the testing data.

1 Naïve Bayesian Network

The below plot shows how the accuracy varies with the amount of training data used.

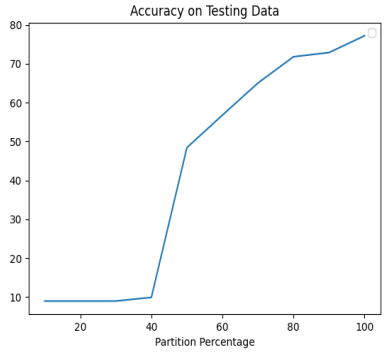


Figure: Digit Recognition Model

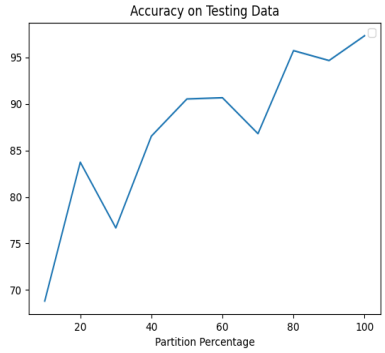


Figure: Face Detection Model

It can be clearly observed from both the graphs that the accuracy is a function of training data set. The more training data, the more accuracy is attained on the testing data.

Also, when only 10 % training data is used, the accuracy is the lowest. This happens due to the fact that many features are not available in such a small data set. Hence, the probability of such features becomes 0 and has to be rounded off to smaller values like 0.001. This results in wrong predictions.

2 Perceptron Neural Network

Same as Bayesian Network, we also fed different size of training data to Perceptron Network, but unlike Bayesian we could not see drastic change in accuracy

as data gets increased. Here are the plots.

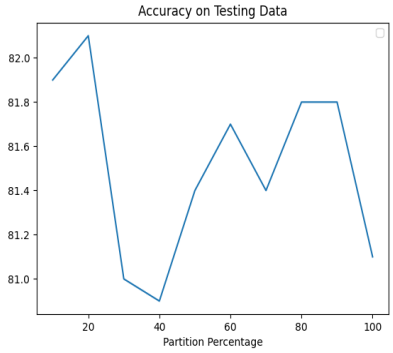


Figure: Digit Recognition Model

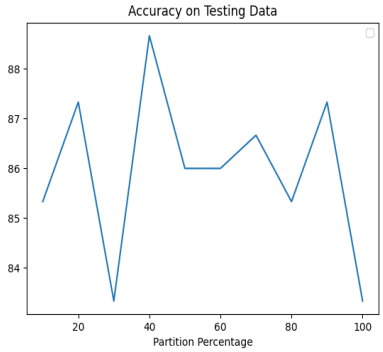


Figure: Face Detection Model

As we can see from the graphs, accuracy gets decreased when training data is increased. Argument for this is model is trying to overfit the training data, hence we get minimal error on training data but more error on testing data.

3 MIRA

As we know MIRA works similarly as Perceptron, the only difference is how we update weights in both cases. Hence, we got almost similar result in this case as well. The graphs are as following.

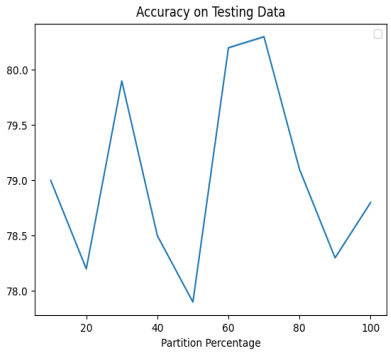


Figure: Digit Recognition Model

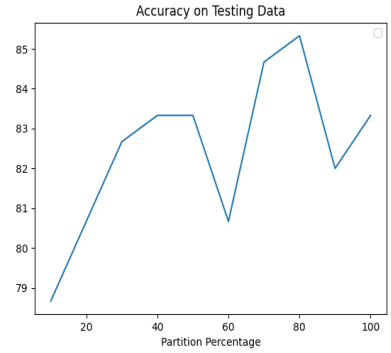


Figure: Face Detection Model

As we can see, accuracy increases until some point, but after that if we increase training data accuracy decreases because model tries to overfit.

Results

Table 1: Accuracy

Algorithm Name	Digit Data	Face Data
Bayesian Netwrok	77.2 %	97.33 %
Perceptron Neural Network	81.4 %	84.66 %
MIRA	80.6 %	87.33 %

1 Regarding Accuracy

- Above mentioned accuracy for each algorithm, for each dataset is measured when we fed 100 % of training data.
- We tried different feature size for both data. We got the best result for following feature size.
  - Digit Data: 1x1
  - Face Data : 5x5
- In Bayesian Network the final accuracy does not change. However, it changes for Perceptron and MIRA because of the fact that every time we run program, the weights are initialized by random numbers. This random assignment makes it subject to change even if the training data used is same.

2 Accuracy with each Epoch

One epoch is when an entire dataset is passed through the network only once. In Bayesian Network we only iterate training data once and we calculate our prior probability and likelihood probability. So, multiple epochs are not required. However, in Perceptron Neural Network and MIRA, we do need multiple epochs in order to learn weights of each feature.

We have plotted graph of accuracy vs. every epoch. Below are the graphs for perceptron learning, but similarly we can plot for MIRA as well. But it would be following similar pattern.

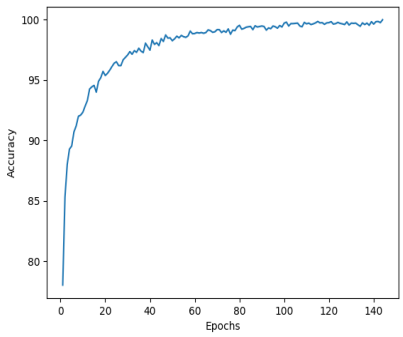


Figure: Accuracy Vs. Epochs for Digit Data

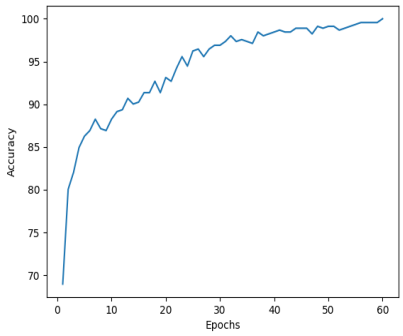


Figure: Accuracy Vs. Epochs for Face Data

3 Presentation for digit data

In order to evaluate our model for digit classification on the images that have never been seen (neither in training nor in testing), we developed web application for the same. Where you can draw a digit in given space and also you can select algorithm, using which you want to predict the digit, and it will call respective model and predicted digit will be displayed.

This is simple client-server architecture. Which is built using flask module available in python. Canvas code was adapted from http://myselph.de/.

3.1 Working:

It takes input from the canvas of size (280x280) and it will convert into the dimensions (28x28) which is understood by our model. Input will be sent to server and server will feed incoming data to pre-trained model and will send response back to client. Below are the screenshots of our front-end.

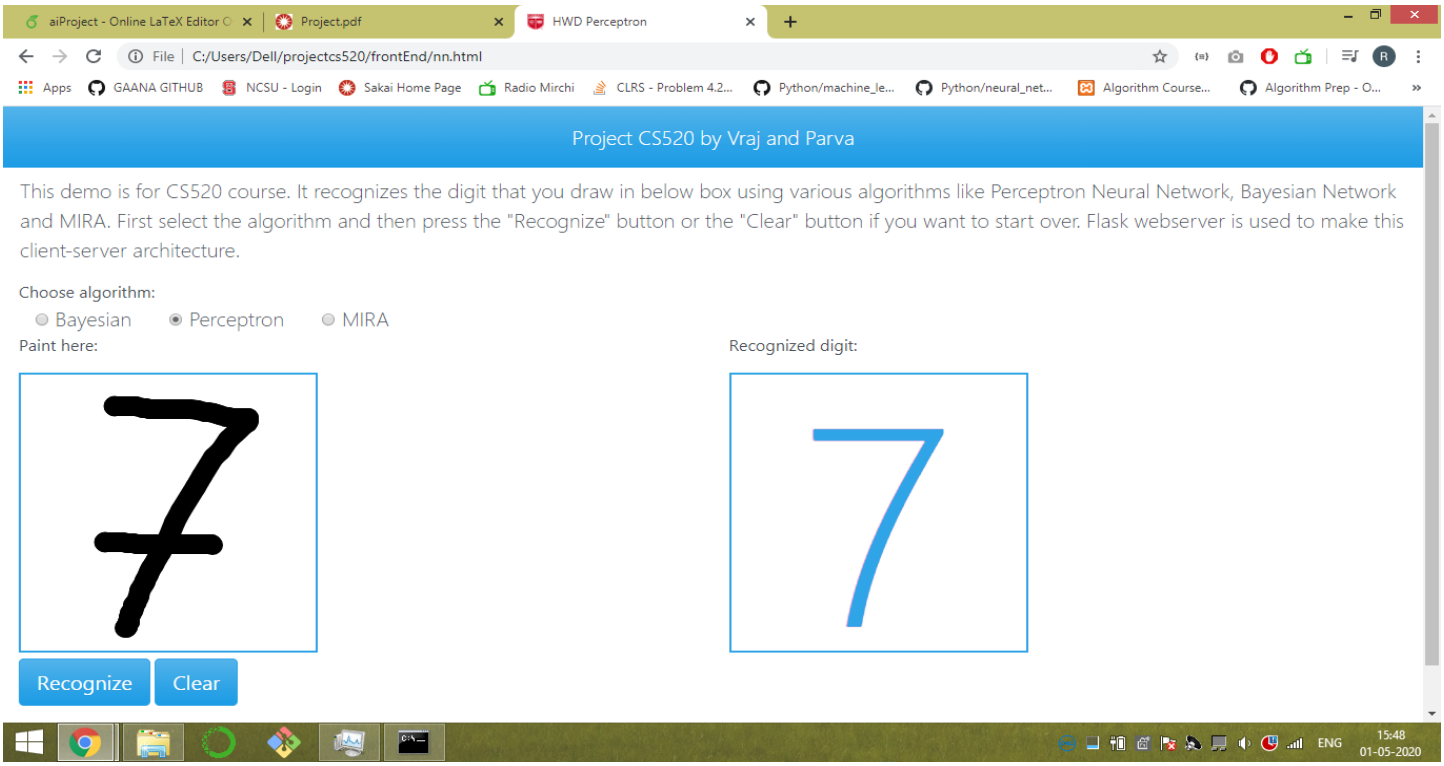


Figure: Digit Recognition using perceptron algorithm

Conclusion

From the above experiments and results, we understood the importance of training data. The portion of training data decides the learning of the model and how it effects the accuracy on the testing and never seen before data.

Also, in perceptron and MIRA algorithms, we found the importance of number of neurons and the random initialization of weights. By simply initializing all the weights to zero always leads to the same results and the model may get stuck in the local minima of the cost function plot. However, by randomly initializing weights we may find other local minimas which are actually better or if we are lucky, we may get to the global minimum.
The other important observation was how the number of features changed the way the model worked. By simply changing few features used, the model can show vast differences in the training data and testing data accuracy.
The realisation that this field is huge and how small changes can make lots of difference, we are excited to try different models and play around with data and features to achieve better results and converge to them in lesser time.