CS520 - Introduction to Artificial Intelligence

Project: Face and Digit Classification

am3372 - Adithya Murugadass, rr1268 - Raghav Ramkumar, Sivakalyan Subbiah ss4356

1 Perceptron

1.1 Basics of the Algorithm

The perceptron is an algorithm used to generate a binary classifier by learning a linear decision function f defined as :

$$f(x_i, w) = w_0 + w_1\phi_1(x_i) + w_2\phi_2(x_i) + ... + w_l\phi_l(x_i) \tag{1}$$

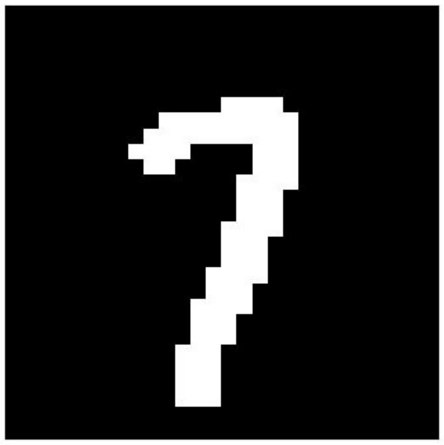
Given a data point x_i, the predicted label(y) is True if $f(x_i, w) \geq 0$ and False if $f(x_i, w) < 0$.
The algorithm is as follows:

1. Initialize the weights w_j to 0.
2. For each example (x_i,y_i) in our training data set X_{train}, do:
   (a) For Binary Classification
      - Compute $f(x_i, w) = w_0 + w_1\phi_1(x_i) + w_2\phi_2(x_i) + ... + w_l\phi_l(x_i)$.
      - if $f(x_i, w) \geq 0$ and y_i is True or if $f(x_i, w) < 0$ y_i is False.
         – Then do nothing and continue to the next iteration.
      - Else, Update the weights w_j
         – if $f(x_i, w) < 0$ and y_i is True then $w_j \leftarrow w_j + \phi_j(x_i)$, for j=1,...,l,and $w_0 \leftarrow w_0$+1
         – if $f(x_i, w) >= 0$ and y_i is False then $w_j \leftarrow w_j - \phi_j(x_i)$, for j=1,.,l,and $w_0 \leftarrow w_0$-1

   (b) For Multi-Label Classification
      - Compute pred = Arg max $f(x_i, w[k])$ for k = 1....10.
      - if pred = y_i
         – Then do nothing and continue to the next iteration.
      - Else, Update the weights w
         – For $w[y_i]$, $w[y_i]_j \leftarrow w[y_i]_j + \phi_j(x_i)$, for j=1,...,l,and $w[y_i]_0 \leftarrow w[y_i]_0$+1
         – For $w[pred]$, $w[pred]_j \leftarrow w[pred]_j + \phi_j(x_i)$, for j=1,...,l,and $w[pred]_0 \leftarrow w[pred]_0$+1
3. Exit the loop when a pass on all examples in X_{train} is made without making any updates.

1.2 Features to Train the Algorithm

Here, we have taken the individual pixels as features for the image. The face image is of size 70 x 60 px giving us 4200 features. Similarly, the digit image is of size 28 x 28 px, giving us 784 features. Figure 1 shows images with their subdivisions


(a) Digit image

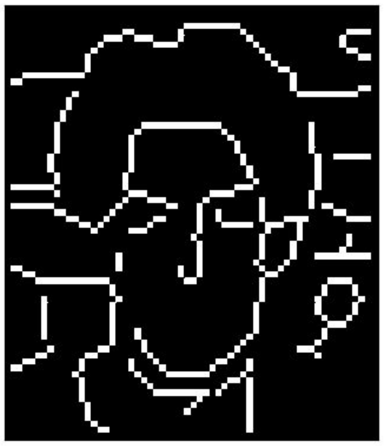
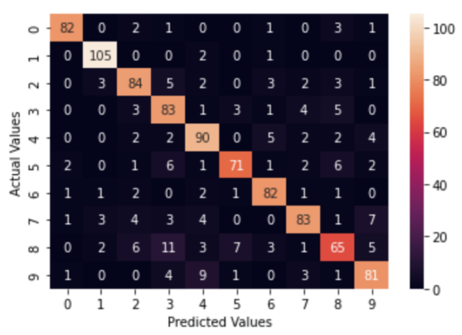

(b) Face image

Figure 1: Images with subdivisions

1.3 Performance of the algorithm

After training the classifier over the entire training and validation data set and testing using the test data set, the Perceptron gave the following results

1. Digit Data
   - Accuracy - *82.6%*
   - Training time - *6.57s (Average)*
2. Face Data
   - Accuracy - *89.3%*
   - Training time - *2291.32s (Average)*


(a) Digit data

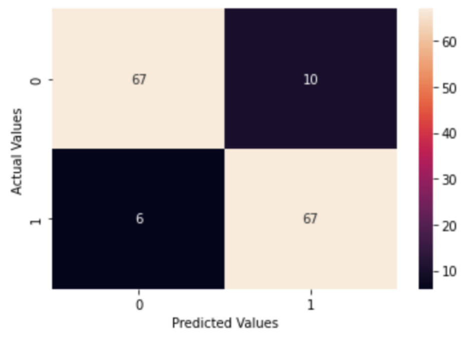

(b) Face data

Figure 2: Confusion matrix

Figure 2 shows the confusion matrices of the data. The matrix of digit data shows us that the most mislabeled digit is 8, which is predicted to be 3 eleven times. This behavior is due to the similarity in their shape, which causes the weight distribution of their features to be similar. However, the confusion matrix of the face data is balanced primarily.

2 Naive Bayes Classifier

2.1 Basics of the Algorithm

The Naive Bayes Classifier is used to classify data based on probability. At its core, the algorithm works on the principles of Bayes' rule

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)} \tag{2}$$

While we can calculate $p(x|y)$ and $p(y)$ relatively easily, calculating $p(x)$ is a bit more complicated. Thus, instead of calculating $p(y|x)$, we calculate $p(x|y)p(y)$ and select the label which has the highest value.
Thus, for an image x, the label y can be obtained from

$$y = arg \max_{y_i \in labels} p(x|y_i)p(y_i) \tag{3}$$

For a given label y_i

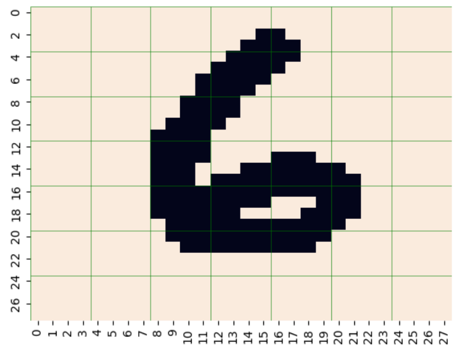
$$p(y_i) = \frac{\text{No. of times label } y_i \text{ is present in the training data}}{\text{Length of training data}} \tag{4}$$

$$p(x|y_i) = \prod_{j=1}^{l} p(\phi_j(x)|y_i) \tag{5}$$

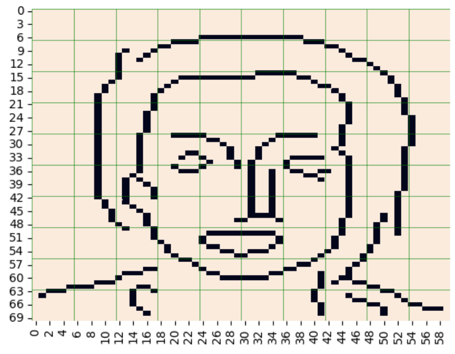
Where $\phi_j(x)$ is a feature of image x.

2.2 Features to Train the Algorithm

Here, we have taken the feature as the number of black pixels in a subsection of the image. The face image of size 70 x 60 px has been divided into subsections of size 7 x 6 px, giving us 100 features. Similarly, the digit image of size 28 x 28 px has been divided into subsections of size 4 x 4 px, giving us 49 features. Figure 3 shows images with their subdivisions



(a) Digit image



(b) Face image

Figure 3: Images with subdivisions

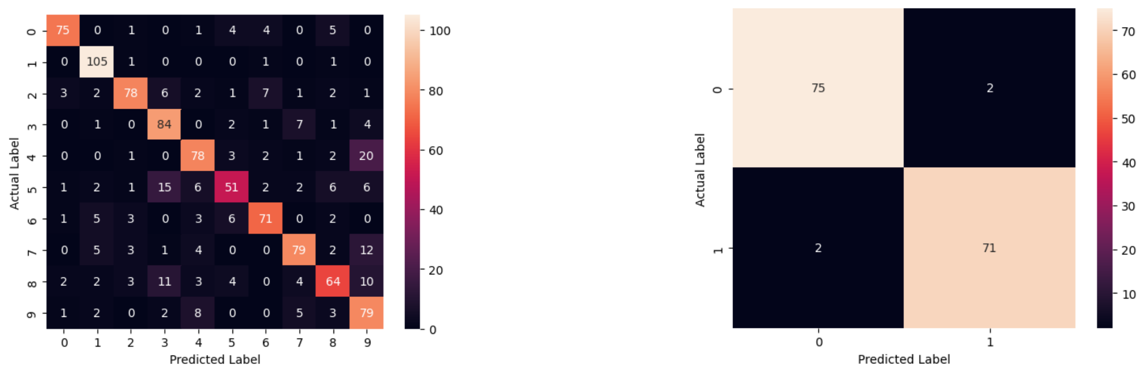
Thus, for a subsection $\phi_j(x)$ with c black pixels, we can calculate its probability given a label y_i

$$p(\phi_j(x) = c|y_i) = \frac{\text{Number of images with } c \text{ black pixels in } \phi_j \textit{ and } \text{label } y_i}{\text{Number of images with feature } y_i} \tag{6}$$

2.3 Performance of the algorithm

After training the classifier over the entire training and validation data set and testing using the test data set, the Naive Bayes Classifier gave the following results

1. Digit Data
   - Accuracy - *76.4%*
   - Training time - *0.0554s (Average)*
2. Face Data
   - Accuracy - *97.3%*
   - Training time - *0.0229s (Average)*



(a) Digit data

(b) Face data

Figure 4: Confusion matrix

Figure 4 shows the confusion matrices of the data. The matrix of digit data shows us that the most mislabeled digit is 4, which is predicted to be 9 twenty times. This behavior is due to the similarity in their shape, which causes the probability distribution of their features to be similar. However, the confusion matrix of the face data is mostly balanced. Hence we can say that in the data set, the images where the face is present and where they are absent are distinct, which results in higher accuracy.

3 Neural Networks

3.1 Basics of the Algorithm

The neural networks are called artificial neural networks (ANNs). Artificial neural networks (ANNs) are comprised of node layers containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

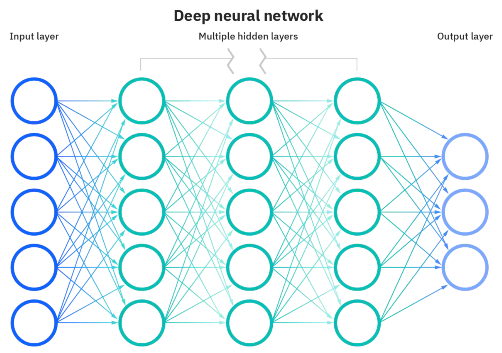


Figure 5: Neural Network

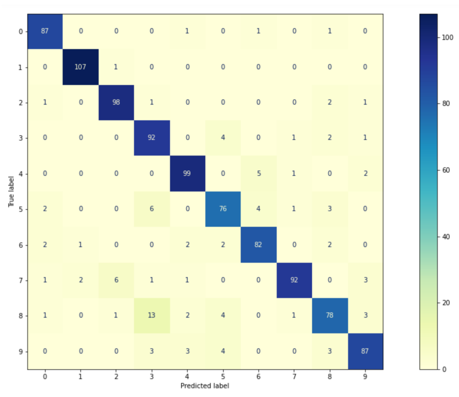
3.2 Algorithm

1. Input - The face image of size 70 x 60 px has been divided into subsections of size 7 x 6 px, giving us 100 features. Similarly, the digit image of size 28 x 28 px has been divided into subsections of size 4 x 4 px, giving us 49 features.
2. Model definition - 5 hidden layers for the digit image and 4 hidden layers for the face image. The final hidden layer has the activation function as softmax.
3. Compile the model - We have compiled the model with categorical cross-entropy for the digit and binary cross-entropy for the face classification. The model also requires us to select an algorithm to perform the optimization procedure; a modern variation, such as Adam Optimizer, can be used. It may also be required to select a performance metric - Accuracy to keep track of during the model training process.
4. Fitting the model - We need to select the training configuration; for instance, the number of epochs = 100 and the batch size = 32.
5. Model Evaluation - The accuracy and confusion matrix are calculated and discussed in the further subsections.

3.3 Performance of the algorithm

After training the classifier over the entire training and validation data set and testing using the test data set, the neural networks gave the following results.

1. Digit Data
   - Accuracy - *88.8%*
   - Training time - *27.125s (Average)*
2. Face Data
   - Accuracy - *85.3%*
   - Training time - *7.192s (Average)*


(a) Digit data

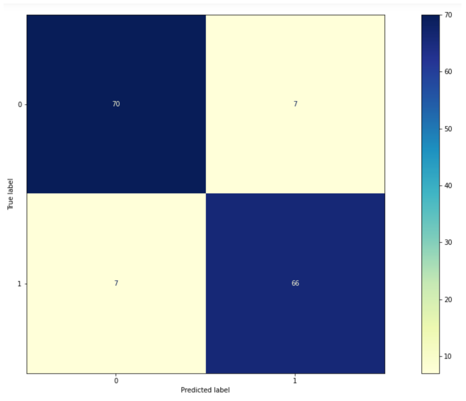

(b) Face data

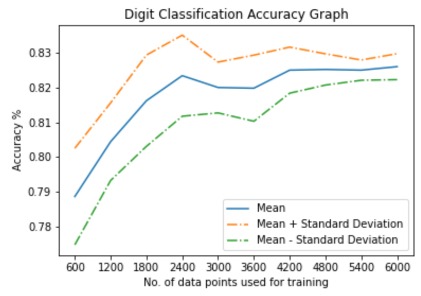
Figure 6: Confusion matrix

Figure 6 shows the confusion matrices of the data. The matrix of digit data shows us that the most mislabeled digit is 8, which is predicted to be 3 thirteen times. This behavior is due to the similarity in their shape, which causes the probability distribution of their features to be similar. However, the confusion matrix of the face data is mostly balanced. 7 face labels which are face are marked as not face. Similarly, the 7 non-face are marked as a face.

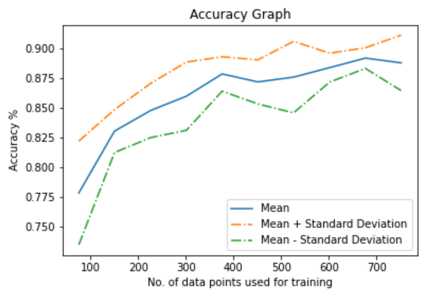
4 Comparison of the Algorithms

4.1 Perceptron

Figure 7 plots the mean accuracy and the standard deviation from the mean with respect to the size of the training data. We can observe that in the cases of both the digit and face data, the improvement in accuracy as we increase the training size is initially significant. However, after a certain point, the rate of increase in accuracy starts to decrease and starts to flatten. We can also see that there is an increase in standard deviation at 100% dataset size, which tells us that the order of data processed by the algorithm affects the final model generated.
From figure 8, we can see that the graph for face data is almost linear, but for digit data, the growth is exponential. This means that as the size of the training data increases, the training time also increases analogously. Thus, we have to choose the training data size wisely because as the size of the training data the time to train will increase exponentially while the improvement in accuracy starts to plateau.

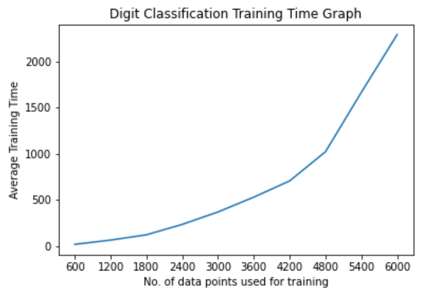


(a) Digit Data

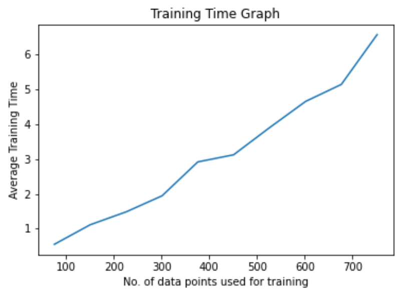


(b) Face Data

Figure 7: Accuracy vs. Size of Train Data



(a) Digit Data



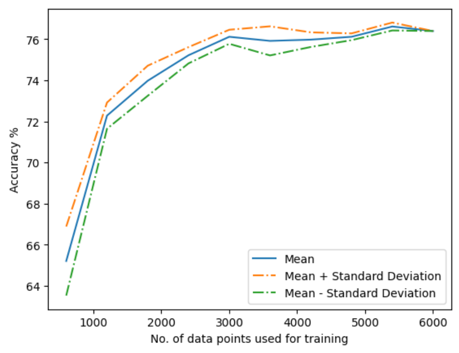
(b) Face Data

Figure 8: Training Time vs. Size of Data

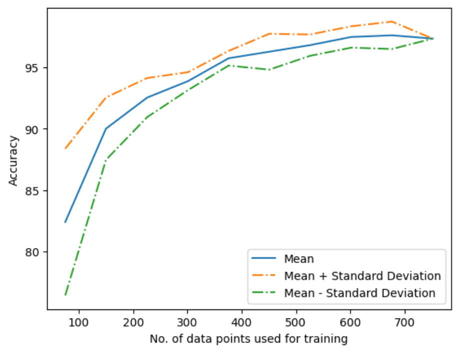
4.2 Naive Bayes

Figure 9 plots the mean accuracy and the standard deviation from the mean with respect to the size of the training data. We can observe that in the cases of both the digit and the face data, the accuracy improvement as we increase the training size is initially significant. However, after a certain point, the graph starts to plateau, and it also drops a tiny bit towards the end (but in this case the standard deviation was also pretty high which might explain the boost in accuracy). To understand the accuracy trend, we also have to compare it with the time taken to train the data, shown in figure 10.

We can see that both graphs are almost linear. This means that as the size of the training data increases, the training time also increases. Thus, we have to choose the training data size wisely because as the size of the training data the time to train will increase linearly. At the same time, the improvement in accuracy will slowly reduce, similar to the law of diminishing returns. However, in the case of Naive Bayes', the penalty is pretty small as training on 6000 data points took less than a second; thus, we can use larger data sets. But, in that case, we also need to consider the possibility of overfitting the training dataset.

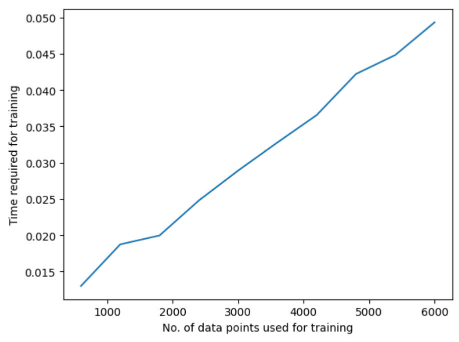


(a) Digit Data

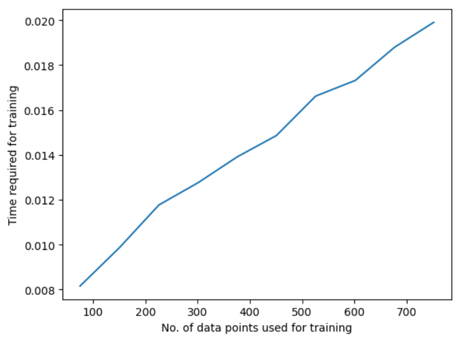


(b) Face Data

Figure 9: Accuracy vs. Size of Train Data



(a) Digit Data



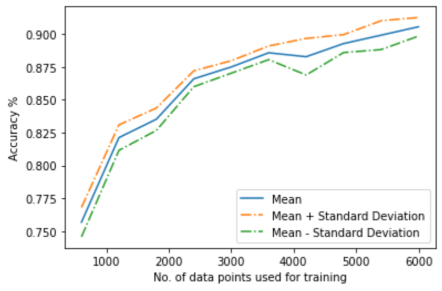
(b) Face Data

Figure 10: Training Time vs. Size of Data

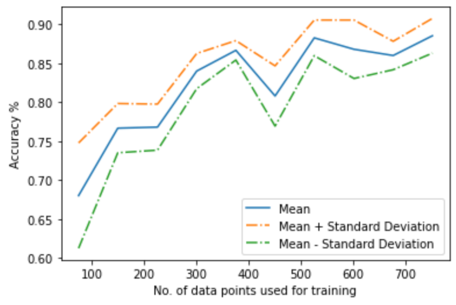
4.3 Neural networks

Figure 11 plots the mean accuracy and the standard deviation from the mean with respect to the training data size. We can observe a steady increase in accuracy in the case of digit data. Whereas the face data, there is a slight dip in the accuracy when we train over 60% of the training data. This might be a case of sample bias in one particular case of the training dataset, as we can also see that the standard deviation is also high. But in general, the number of training data points is directly proportional to the accuracy. However, after a certain point, the graph starts to plateau, dropping slightly towards the end (with the standard deviation still being high).
To understand the accuracy trend, we also have to compare it with the time taken to train the data, shown in figure 12. We can see that both graphs are almost linear. The Digit model trains considerably faster, even when the data is the entire sum of training and validation. We can also see that the accuracy graph still has a mild upward slope, so there is still a possibility of improving the accuracy with a larger dataset. However, in the case of the Face model, the time is linearly increasing with respect to the number of data. Training over 90% of training data gives us better

results as compared to training over 100% of the dataset. So we should be careful about selecting the size of our training dataset as there might be cases where the accuracy drops with a larger training dataset.

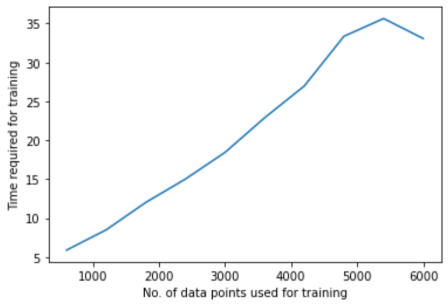


(a) Digit Data

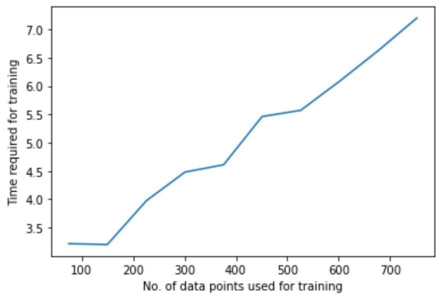


(b) Face Data

Figure 11: Accuracy vs. Size of Train Data



(a) Digit Data



(b) Face Data

Figure 12: Training Time vs. Size of Data

5 Conclusion

As we saw through our examples, it is not necessary to use complex Neural Networks to get an accurate model to solve a problem. Training on Naive Bayes' algorithm itself resulted in a 97% accuracy in the case of face detection, and Perceptron gave us an accuracy of 89%, which are both better than that of the Neural Network. However, in the case of digit classification, Naive Bayes' and Perceptron performed poorly compared to the Neural Network.

We also saw through the plots that the accuracy of the models varied vastly based on the distribution of the dataset. When we trained on random subsections of the dataset, we could see that the standard deviation also changed considerably. This is because when we select a random subsection, the data we get in the subsection doesn't need to be distributed similarly to that of the entire dataset. For instance, in the case of Face Detection, we might get more face images than non-face images, which also affects the algorithm's accuracy. Thus, it is essential that if we want to improve the accuracy of our model, we don't just focus on the size of the dataset but also its distribution.

Lastly, the decision of which model gives us better results boils down to the core problem we are trying to solve. In some cases, Neural Networks, despite taking more time to train, result in a better model overall. There are also cases where using a basic algorithm like Perceptron or Naive Bayes' can give us equally good results with shorter training times.