CS 520 - Introduction to Artificial Intelligence

Project Title: Face and Digit Classification

Pradeep Roy	py160
Pranoy Sarath	ps1279
Satyabrat Bhol	sb2311

TABLE OF CONTENTS

Image Data Extraction

A Data Loader utility module was created for storing the images. Each image is represented using a matrix of size $Image_{height}$ * $Image_{width}$. Based on the image dimensions, we load the image into memory using a file reader. Then we scan through each line and check whether it has a space character or any other valid character to represent the image. All the spaces are encoded as ‘0’s and all other characters are encoded as ‘1’.

Feature Engineering

Black and white pixel count

A new feature matrix is created by summing up the number of black pixels and the number of white pixels. The new feature set has two features(black_pixel_count and white_pixel_count)

Reduced Grid Feature

A reduced feature set of black and white pixels represented with 1's and 0's where 1 represents the black pixel and 0 represents the white pixel is created by traversing a smaller grid size of (2 * 2) with a jump of 2 units to the right and down. This results in a matrix that is exactly half in size

a. First, you start at (0,0) and then you sum all the values of the elements in the first (2 X 2) grid {elements of the grid: [0,0], [1,0], [0,1], [1,1]}. If the sum is greater than 0 then we update the first element of the new feature set with 1 else, we will update it with 0.
b. Next, we jump two units to the right and start at (2,0) and then again sum all the values of the elements in (2 X 2) grid {elements of the grid [2,0], [3,0], [2,1], [3,1]}. Similarly, we update the next element of the new feature set based on the sum. And we repeat this until we reach the end of the row.
c. Now we jump 2 units down and follow the same steps as above. We repeat this until we have read the entire feature set. And we have a reduced feature set.

Other variations for the above feature selection would be to take a grid size of (4X4) with a jump factor of 2 right and down. This results in a much smaller feature set size. The reason for not directly jumping 4 units is to maintain the integrity and information of the feature matrix.

Noise reduction

In the digit data set if we think that the samples of any one underlying label representation in the feature matrix are nothing but a pure consistent representation with some added noise at the borders. We can find a way to reduce the noise as a pre-processing step and train on the cleaned feature set to understand the underlying distribution of the labels better. To perform noise reduction, while summing the values w.r.t the traversing grid we update the value of 1 in the new feature set if the sum is greater than a certain threshold. {For example: In a 4 X 4 traversing grid, we update 1 in the new feature matrix only if the sum of all the 16 elements is greater than 2 else, we update it a 0}

Classification Models

Naïve Bayes

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

To calculate the above expression, we need to know P(x/y) and P(y)

Defining P(x/y) and P(y)

P(y): P (y = i) = Number of images where the true label 'y' is equal to 'i' / The total number of images

$$\hat{P}(y) = \frac{c(y)}{n}$$

where c(y) is the number of training instances with label y and n is the total number of training instances.

P(x/y): a joint distribution over a label 'y' and a set of observed random variables, or *features*, $(F_1, F_2, \ldots, F_n)$using the assumption that the full joint distribution is given as

$$P(F_1, \ldots, F_n, Y) = P(Y) \prod_i P(F_i|Y)$$

The current probability estimates are unsmoothed, i.e. the empirical estimates are not adequate in our systems. We need to make sure that no parameter ever receives zero probability. We use Laplace smoothing to compensate for this.

$$P(F_i = f_i|Y = y) = \frac{c(f_i, y) + k}{\sum_{f_i' \in \{0,1\}} (c(f_i', y) + k)}$$

Black and white Pixel count as a feature set

$$P(\phi_1(x) = a / y = b) = \frac{\text{Number of 'b' labelled images with 'a' black pixels} + k}{\text{Number of 'b' labelled images} + k\,(\text{no. of labels in } y)}$$

$$P(\phi_2(x) = c / y = b) = \frac{\text{Number of 'b' labelled images with 'c' white pixels} + k}{\text{Number of 'b' labelled images} + k\,(\text{no. of labels in } y)}$$

Where,

1. The domain of *a* is [0 ..... (Feature height * Feature width)]
2. The number of white pixels = Total number of pixels - Number of black pixels
3. The domain of *y* is [0...9] for digit data and [0,1] for face data
4. *K* represents the smoothing factor. we will be using the validation data set to identify the best value for K

$$P(x/y='n') = \prod_{j=1}^{2} P(\phi_j(x)/y=n) \quad \forall\, x \in [0 - \text{matrix size}]$$

Where *j* represents the feature size.

We can find the probable label using the below approach,

$$\begin{aligned} P(y|f_1,\ldots,f_m) &= \frac{P(f_1,\ldots,f_m|y)P(y)}{P(f_1,\ldots,f_m)} \\ &= \frac{P(y)\prod_{i=1}^{m} P(f_i|y)}{P(f_1,\ldots,f_m)} \\ \arg\max_y P(y|f_1,\ldots,f_m) &= \arg\max_y \frac{P(y)\prod_{i=1}^{m} P(f_i|y)}{P(f_1,\ldots,f_m)} \\ &= \arg\max_y P(y)\prod_{i=1} P(f_i|y) \end{aligned}$$

Reduced Gird Feature

We perform similar steps as above now using the reduced grid feature set where the features are the reduced pixels.

$$P(\phi_1(x_i)/y=b) = \frac{\text{Number of 'b' labelled images with pixel } x_i=1 \text{ (black)} + k}{\text{Number of 'b' Labelled images} + k \times (\text{no. of true } y \text{ labels})}$$

$$P(\phi_2(x_i)/y=b) = \frac{\text{Number of 'b' labelled images with pixel } x_i=0 \text{ (white)} + k}{\text{Number of 'b' labelled images} + k \times (\text{no. of true } y \text{ label})}$$

$y = [\text{'true'}, \text{'false'}]$ for face

$y = [0, 1, 2 \text{ ——— } 9]$ for digit.

x_i represents i^{th} pixel

multiplying many probabilities together often results in underflow, we will instead compute *log probabilities* that have the same argmax

$$\begin{aligned}\arg\max_y \log P(y|f_1,\ldots,f_m) &= \arg\max_y \log P(y,f_1,\ldots,f_m) \\ &= \arg\max_y \left\{\log P(y) + \sum_{i=1}^{m} \log P(f_i|y)\right\}\end{aligned}$$

Perceptron

Input Feature Used

Raw pixels were used as the input features for both face and digit classification. In addition to this, a node of value 1 will be appended for the bias node

Output nodes

In the case of Digits, we have 10 nodes each, indicating the output node from 0 to 9. For the face, we have 2 nodes indicating whether it is a face or not.

Algorithm overview

- Initialize the weight matrix *W* with all the values as zero. This will be of the dimension *Input feature size * Number of Output Nodes*
- For each example (x_i, y_i) in our training set X_{Train} , do:
    - Compute $f(x_i, w) = w_0 + w_1\varphi_1(x_i) + w_2\varphi_2(x_i) + w_3\varphi_3(x_i) + \cdots + w_l\varphi_l(x_i)$
    - From the output nodes, select the one that has the highest $f(x_i, w)$ value. This node will be classified output. Compare this prediction against the actual y_i.
    - *If* the labels are same (for successful prediction), continue with the next train data input.
    - *Else* (For the wrong prediction)
        - For the label that got predicted, we have to penalize the corresponding node weights and the bias weight
            - $w_j \leftarrow w_j - \varphi_j(x_i)$, *for* $j = 1, \ldots, l$, and $w_0 \leftarrow w_0 - 1$
        - For the label that was supposed to be predicted, we have to increase the corresponding node weights and the bias weight
            - $w_j \leftarrow w_j + \varphi_j(x_i)$, *for* $j = 1, \ldots, l$, and $w_0 \leftarrow w_0 + 1$

Training Configuration Used

- The above algorithm was run for 30 epochs. If There was no weight change (no wrong prediction) for a particular epoch, we don't execute further epochs.
- To update the weights, we have used a learning factor of 0.01

Logistic Regression

Input Feature Used

Raw pixels were used as the input features for face and digit classification.

Output Classes

In the case of Digits, we have 10 output classes, indicating the digits from 0 to 9. For the face, we have 2 classes indicating whether it is a face.

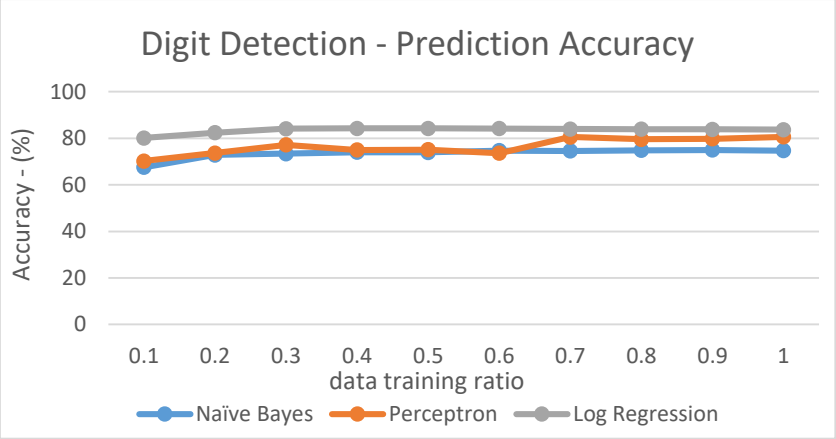
Implementation

Used *LogisticRegression* library part of sklearn.linear_model module to create a Logistic Regression model

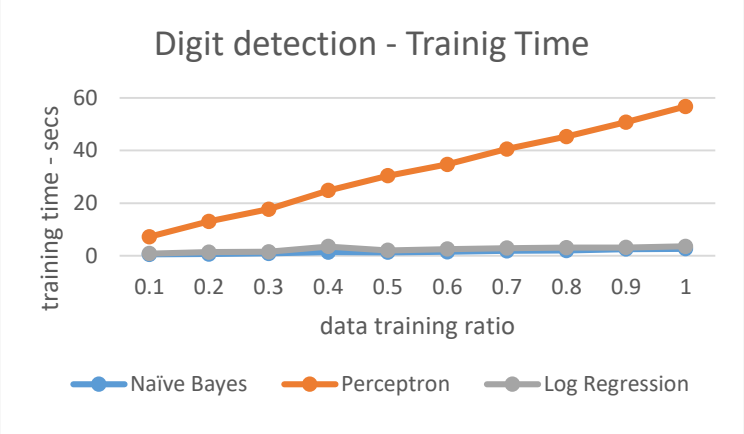
Results

Digit prediction stats

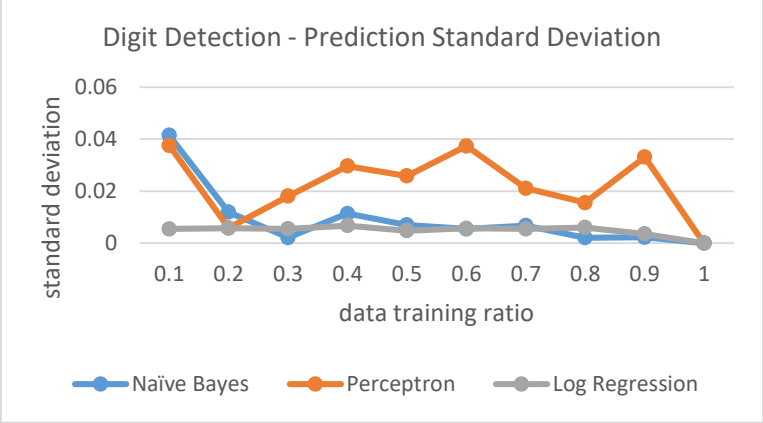
Accuracy



Training Time

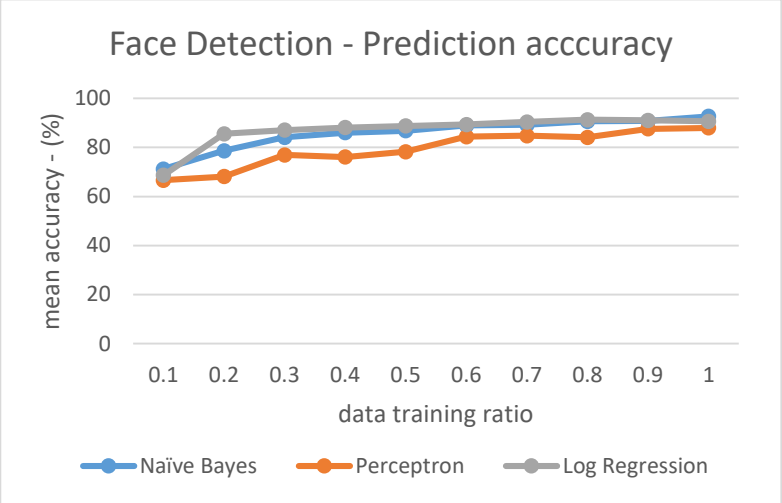


Prediction Standard deviation

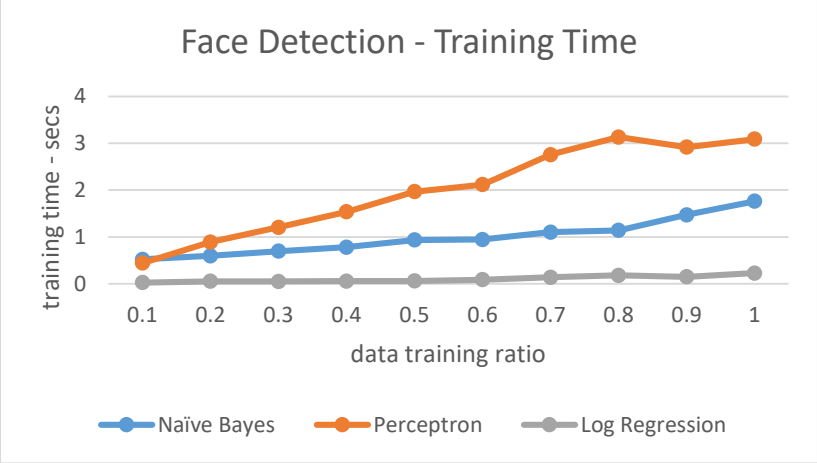


Face Detection

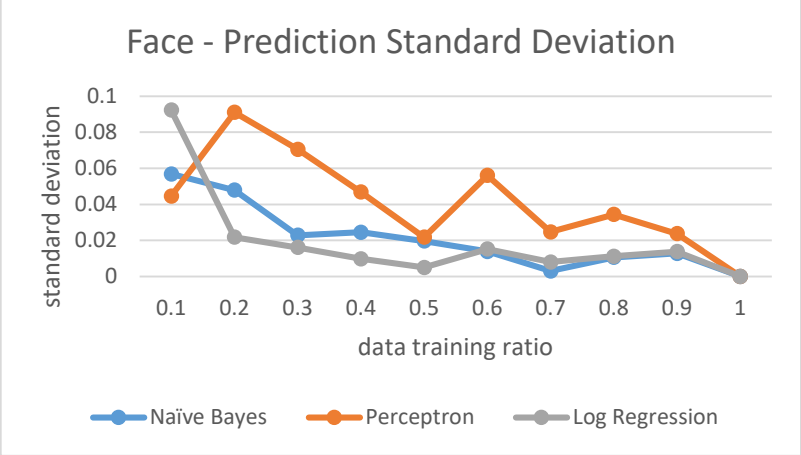
Mean Prediction Accuracy



Training Time



Standard Deviation



Discussions

1. For Naïve Bayes we find the best value of K using a validation data set.
    - After training the model we tested the validation data set using different values of k and found the best value of K for each data set.
    - Best K identified for digit data is 1
    - Best K identified for face data set is 8
2. All our models have given good results,

    Digit data test set accuracies:
    - Naïve Bayes: 74.98
    - Perceptron: 80.6
    - Log regression: 84.28

    Face data test set accuracies:
    - Naïve Bayes: 92.7
    - Perceptron: 88
    - Log regression: 91.3
3. The training time was directly proportional to the amount of training data considered. The perceptron algorithm took the most time among the 3 classification models.
4. Accuracies were not directly proportional for all the iterations due to the randomness introduced on the data set. But we can say that training accuracy did go up when increased training data is used until some point but later converged to a value. Sometimes, the introduction of more data after convergence caused the test accuracy to go down by a small amount.
5. Standard deviation saw inverse proportionality to the amount of training data until it converged to a point.

Conclusion

It has always been a notion that the more the training data more the prediction accuracy but here we see that it is not the case. We see that the accuracy converges after a certain amount of training data. What matters the most is the quality of the data set. The other notion is more the training time the more the accuracy. Perceptron took a lot of time for training but the performance is bad on the face detection data set

References

[1] http://rl.cs.rutgers.edu/fall2019/lecture11.pdf - Lecture slides on Machine Learning in CS 520, Introduction to AI.

[2] https://inst.eecs.berkeley.edu//~cs188/sp11/projects/classification/classification.html