Final Exam Report

Ritwika Das - 214003221

December 16, 2024

1. Implementing Each Algorithm

Naive Bayes:

Since we need to use Bayes Rule equation to predict the probabilities of each outcome(label), we have to implement it in the coding. For Baye Rule, probability of outcome(specific label) given a feature set of a sample is equal to the product of probability of outcome(specific label) and probability of this data given outcome is a specific label divided by the probability of the data.

Because it is only needed to know which outcome(label) has the greatest probability, it compares the value of each outcome(label) for the same data(sample). Since the probabilities of the data are always the same, it just needs to compare the value of probability of outcome(specific label) times probability of this data given outcome is a specific label.

Because multiplying many probabilities together often results in underflow, it will instead compute log probabilities.

For probability of outcome(specific label), it uses the number of samples with specific label divided by total number of samples. Then, calculate its log value.

For probability of this data given outcome is a specific label, it is the same as the product of the probability of each feature given a specific outcome(label). It calculates each probability of a single feature given a specific probability(label), and then multiply their log values.

For probability of a single feature given a specific probability(label) it counts the total times a specific feature has value 0 with a specific label, and divided by the total number of samples with this specific label. Since it only has two possible values for each feature(0 and 1), if a specific feature has value 1, then it uses 1 minus the probability it has above to get the answer. Lastly, it converts the value to log value.

It sums all log values it gets above to get the log probability of a specific label for a data. The label with greatest probability is the final prediction.

When doing the training process, it uses a smoothing parameter k to improve the accuracy of prediction. It has a set of 10 different k values, and each iteration runs with one k value. After 10 iterations, it assigns k to the value with best accuracy.

Perceptron:

For all training data, it picks them one by one to do the following training process:

(1) For each training data, there are A features vectors and B labels. Then it initializes the weight as a BxA matrix which contains small values (0.5 in our project). Multiplying the weight matrix by the features vectors will yield a vector that contains a value corresponding to each label. Among these values, it chooses the label with the largest value as our prediction

(2) It compares the prediction with the actual answer and take different actions depending on the comparison result.

   (a) If the label it predicts is right, then it doesn't modify the weight and jump to the next picture.

   (b) If the label it predicts is not the same with the actual label,

      (i) it increases the weight value used for the real label if the current result for the real label is less than zero.

      (ii) it decreases the weight value used for the prediction label if the current result for that label is greater than zero.

(3) it keeps doing the above two steps until the algorithm reaches the iteration limits we predefined or there is no weight value change for all training data.

2. The Results

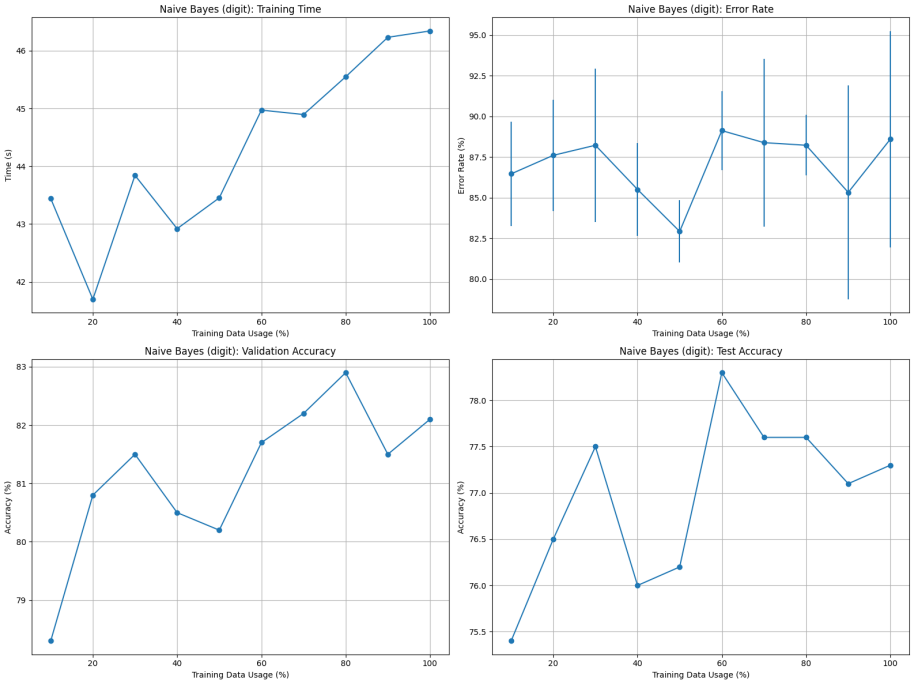

Figure 1: Naive Bayes (Digit) Metrics

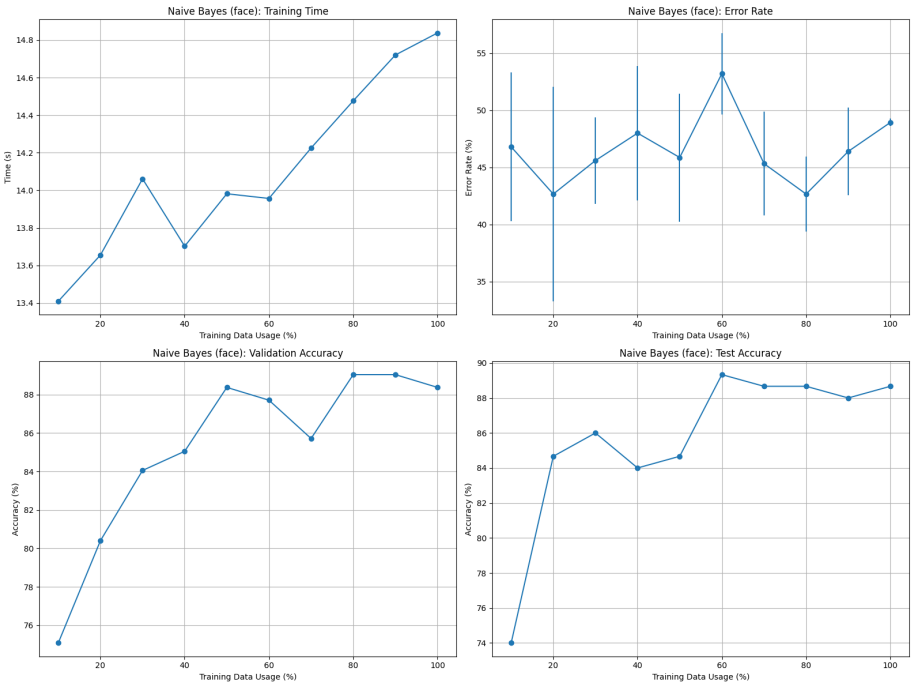

Figure 2: Naive Bayes (Face) Metrics

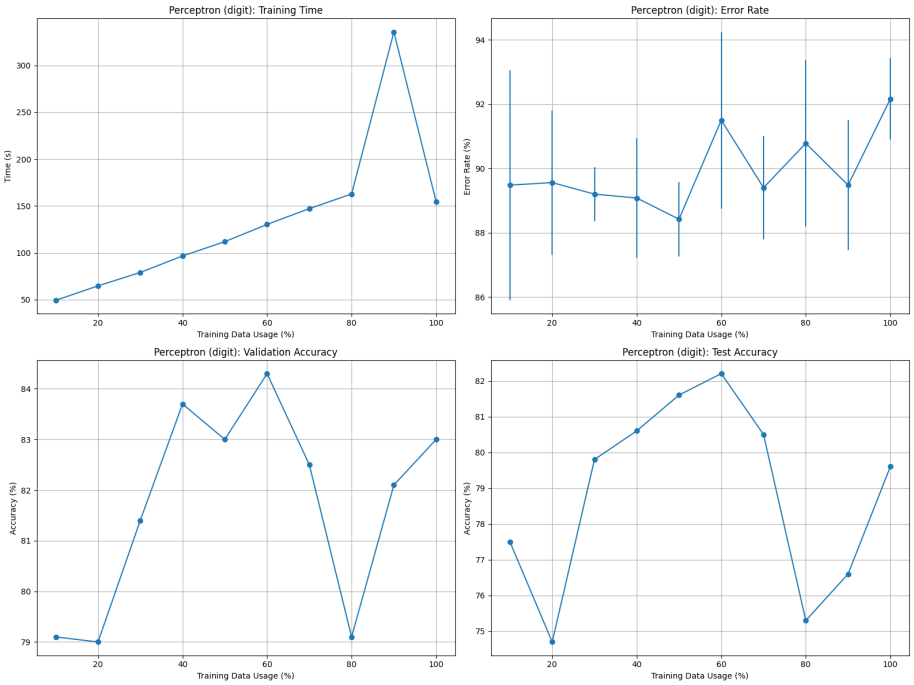

Figure 3: Perceptron (Digit) Metrics

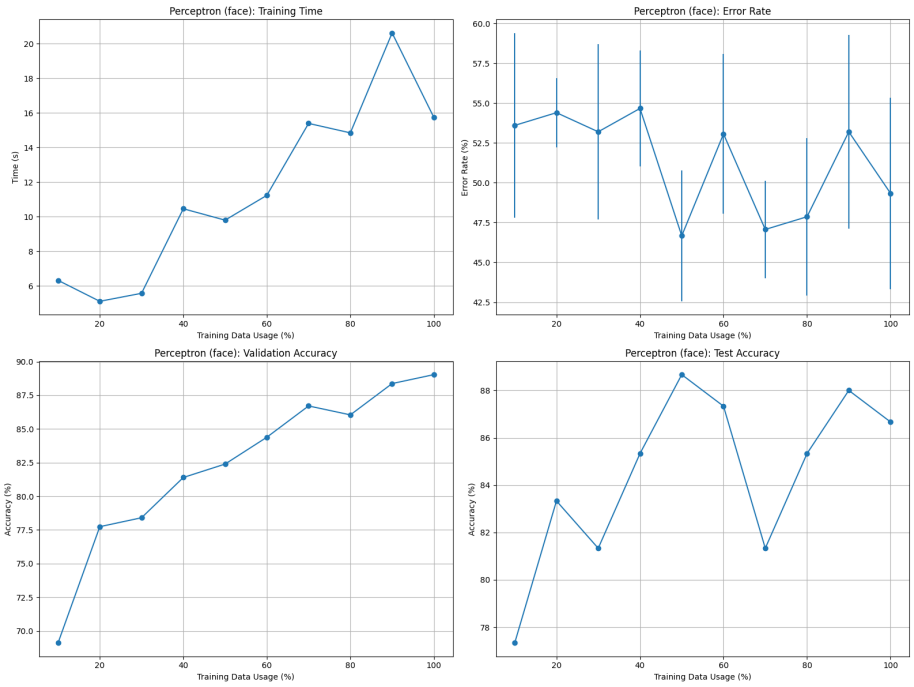

Figure 4: Perceptron (Face) Metrics

As above Figure 1-4 show, we can see that the accuracy goes up as we are using more and more training data and we can see a general trend that the amount of data needed to increase the accuracy is increasing as the accuracy keeps increasing. While there are a few outliers, in general the accuracy is increasing with more training data. As we are getting higher and higher accuracy, it's harder and harder to increase the accuracy by adding more training data.

Specifically, in the top right image of Figure 1-4, the vertical lines show the standard deviation as the number of data points increases during training. We can see a general pattern that as the size of training data increases, the prediction error is getting smaller and smaller.

3. Learned Lessons

Through the implementation and enhancement of this classification code, several important concepts and techniques in machine learning were reinforced. First, the project highlighted the strengths and limitations of Naive Bayes and Perceptron classifiers when applied to image data.

Naive Bayes, while computationally efficient, demonstrated the need for thoughtful feature engineering, such as grid-based features and binarization, to better handle the spatial dependencies of image pixels. On the other hand, the Perceptron classifier underscored the importance of non-linearity and feature transformations, where methods like kernel approximation and data augmentation played a critical role in improving performance.

Additionally, techniques such as balancing datasets, tuning hyperparameters, and utilizing advanced augmentation methods (e.g., rotations, noise injection) emphasized the practical challenges of preparing data for robust machine learning models. Finally, evaluating metrics like training time, error rate, validation accuracy, and test accuracy provided insights into the

trade-offs between model complexity and performance, helping identify areas for further refinement and learning.