My Report on Face and Digit Classification Using Naive Bayes and Perceptron

Pranav Ambulkar

December 21, 2024

1 Introduction

In this project, I tackled two classification tasks:

1. **Digit Classification**: Identifying which digit (0–9) is present in a 28×28 textual image of a handwritten digit.
2. **Face Classification**: Determining whether a given 60×70 textual image represents a face (`1`) or not (`0`).

I implemented two supervised learning algorithms—*Naive Bayes* and *Perceptron*—to solve these tasks. Then, I compared their performance as I varied the amount of training data (from 10% to 100%) in both the digit and face domains.

2 Feature Extraction

For each image (both digits and faces), I converted the data into a *binary feature vector*:

- I read each line of the text-based image.
- I mapped any character `#` or `+` to `1`.
- All other characters (spaces, periods, etc.) were mapped to `0`.

Thus, a $W \times H$ image became a feature vector of length $W \times H$. This simple binary representation served as my input for both classifiers. Although it ignores more advanced cues (like edges or corners), it proved sufficient for achieving moderate classification performance.

3 Learning Algorithms

3.1 Naive Bayes

For Naive Bayes, I computed two core components for each class c:

1. The prior $P(y = c)$, which is the relative frequency of class c in the training data.
2. The likelihood $P(x_i = 1 \mid y = c)$ for each binary feature x_i, assuming conditional independence given y.

To avoid numerical underflow with so many features, I stored probabilities in *log-space*. I also used *Laplace smoothing* to ensure no probability was exactly zero. Once these parameters were learned, I predicted a new sample's label via:

$$\hat{y} = \arg\max_c \Big[\log P(y = c) + \sum_{i=1}^{d} \log P(x_i \mid y = c)\Big].$$

Here, x_i can be either 1 or 0, and I used the appropriate log-likelihood term accordingly.

3.2 Perceptron

I also implemented a Perceptron classifier. In the multi-class case (digits 0–9), I maintained one weight vector $\mathbf{w}_c$ per class c. The training procedure for each example $(\mathbf{x}, y)$ is:

1. **Predict**:
$$\hat{y} = \arg\max_c \ \mathbf{w}_c \cdot \mathbf{x}.$$
2. **Update** if $\hat{y} \neq y$:
$$\mathbf{w}_y \leftarrow \mathbf{w}_y + \mathbf{x}, \quad \mathbf{w}_{\hat{y}} \leftarrow \mathbf{w}_{\hat{y}} - \mathbf{x}.$$

I repeated this update for multiple epochs through the training data. For the face classification (binary) problem, I used the same logic but only had two weight vectors (one for "face" and one for "not face").

4 Experimental Setup

To measure how performance changes with the amount of data, I tried **10%, 20%, . . . , 100%** of the available training set. For each fraction:

1. I *randomly sampled* that fraction of the training set.
2. I *trained* the classifier on that subset.
3. I *tested* on the entire test set.
4. I *repeated* this sampling and training 5 times.

I then reported the *mean* and *standard deviation* of accuracy across those 5 runs.

5 Results and Discussion

5.1 Naive Bayes (Digits)

Train Fraction	Mean Accuracy	Std. Accuracy
10%	0.7402	0.0073
20%	0.7592	0.0106
30%	0.7560	0.0076
40%	0.7558	0.0084
50%	0.7682	0.0066
60%	0.7612	0.0061
70%	0.7638	0.0071
80%	0.7660	0.0044
90%	0.7664	0.0015
100%	0.7710	0.0000

Table 1: Naive Bayes (Digits) Accuracy vs. Training Fraction.

I noticed that accuracy gradually increased with more training data, settling at around 77%. The standard deviations were small, suggesting consistent performance across different random subsets. Overall, Naive Bayes is fairly strong for digit classification under a simple binary representation.

5.2 Perceptron (Digits)

Train Fraction	Mean Accuracy	Std. Accuracy
10%	0.7358	0.0327
20%	0.7608	0.0231
30%	0.7396	0.0531
40%	0.7732	0.0352
50%	0.7792	0.0213
60%	0.7812	0.0131
70%	0.7780	0.0207
80%	0.7986	0.0074
90%	0.8010	0.0159
100%	0.8092	0.0174

Table 2: Perceptron (Digits) Accuracy vs. Training Fraction.

Here, the Perceptron showed more variation for smaller training subsets (notable standard deviations). With sufficient data (80% or more), the Perceptron reached around 80%–81%, surpassing Naive Bayes for the digit task. This suggests that a linear discriminant can capture subtle patterns among the 10 digit classes more effectively, given enough examples.

5.3 Naive Bayes (Faces)

Train Fraction	Mean Accuracy	Std. Accuracy
10%	0.7733	0.0436
20%	0.8187	0.0328
30%	0.8653	0.0290
40%	0.8827	0.0108
50%	0.8773	0.0131
60%	0.8773	0.0116
70%	0.8920	0.0181
80%	0.8787	0.0142
90%	0.8853	0.0078
100%	0.9067	0.0000

Table 3: Naive Bayes (Faces) Accuracy vs. Training Fraction.

For faces, the Naive Bayes classifier reached about 90.7% with the full data. The conditional independence assumption seems to work quite well in distinguishing face vs. non-face. The standard deviations are low at higher fractions, indicating stable performance.

5.4 Perceptron (Faces)

Train Fraction	Mean Accuracy	Std. Accuracy
10%	0.7280	0.0512
20%	0.7960	0.0511
30%	0.7987	0.0251
40%	0.8493	0.0225
50%	0.8387	0.0232
60%	0.8467	0.0094
70%	0.8720	0.0088
80%	0.8627	0.0167
90%	0.8587	0.0115
100%	0.8680	0.0154

Table 4: Perceptron (Faces) Accuracy vs. Training Fraction.

The Perceptron steadily improved with more data but topped out around 86.8%. Overall, Naive Bayes outperformed it slightly in the face task. I suspect the binary face-or-not-face scenario might be well suited for the conditional independence assumption, so Naive Bayes can perform strongly here.

6 Lessons Learned

1. **More Data Helps.** In both digit and face classification, increasing the training set fraction boosted accuracy.

2. **Naive Bayes vs. Perceptron.** For digits, Perceptron eventually overtook Naive Bayes (81% vs. 77%). For faces, Naive Bayes had a slight edge (90.7% vs. 86.8%).

3. **Simple Binary Features.** Even though I used just "on/off" pixels, the performance was surprisingly decent. More sophisticated feature extraction might improve results further, but wasn't required here.

4. **Hyperparameters Matter.** I used 5 epochs for Perceptron and a default Laplace smoothing for Naive Bayes. Tuning might have yielded better results, but this project was more about comparing algorithms under consistent settings.

5. **Variance in Small Data.** With smaller subsets (10%–20%), the standard deviation sometimes spiked, especially for Perceptron. This highlights that random subsets can cause more fluctuations when data is scarce.

7 Conclusion

In this assignment, I successfully implemented and compared Naive Bayes and Perceptron classifiers on digit (multi-class) and face (binary) data using simple binary features. The main conclusions are:

- **Digits:** Perceptron can eventually outperform Naive Bayes with enough training data.
- **Faces:** Naive Bayes is slightly superior for a 2-class face-or-not-face classification.

These results underscore how different algorithms perform better under certain conditions (number of classes, amount of data, and so on). I also saw how simple feature representations can work surprisingly well, especially when paired with relatively robust classifiers. Overall, I learned the importance of methodical experimentation, the impact of training set size, and the need to consider variance when evaluating performance.