Final Examination: Face and Digit Classification

Sahil Sharma (ss4756)

CS440: Introduction to Artificial Intelligence

21 December 2024

Introduction

The objective of this project was to implement and compare the performances of the two most basic classification algorithms, Naive Bayes and Perceptron, on two different datasets: digit recognition and face detection. Both the algorithms have been implemented here because they are relatively simpler and form the very basis of the study of machine learning. Their strengths and weaknesses, and applicability provide useful insight into more complex models and real-world applications.

Naive Bayes is a probabilistic classifier based on Bayes' theorem that comes with strong independence assumptions between the features. Though simple in nature, it works incredibly well in many domains, including text classification and medical diagnosis. Their strengths are computational efficiency and little training time requirement in general.

On the other hand, the Perceptron is a kind of linear classifier and one of the simplest forms of a neural network. The method works in searching for a hyperplane that optimally separates classes in feature space by iteratively updating weights whenever it makes a misclassification. Compared to Naive Bayes, it is heavier computationally but potentially more accurate in conditions where the interaction of features might be meaningful.

This project utilizes the ASCII-art style datasets for both digits and faces and thus provides a very controlled environment for the evaluation and comparison of these algorithms. These datasets were split into training, validation, and test sets for a strong evaluation without overfitting.

Experimental Procedure

Therefore, the whole experimental platform design is divided into the following procedure: loading the data, feature extraction, algorithm implementation, hyperparameter tuning, and model performance testing. Each step was deliberately considered with a view to ensuring result integrity and reliability.

1. Data Loading and Splitting:
    - Rationale: Splitting datasets into training, validation, and test sets is a standard practice to assess a model's performance on unseen data and to fine-tune hyperparameters without biasing the evaluation. The training set is used to train the model, the validation set assists in hyperparameter tuning and model selection, and the test set provides an unbiased evaluation of the final model's performance.
    - Implementation: The digit and face datasets were read from text files, parsed into images and corresponding labels, then appropriately split. The combination of training and validation sets was done to make as much data available for training, a very critical aspect in the evaluation of the performance over a range of training sizes
2. Feature Extraction:
    - Digits:
    - Methodology: Each of the 28×28 ASCII digit images was converted to a binary feature vector where each pixel was represented as 0 (in case of blank) or 1 (in case of marked).
    - Justification: This simple representation brings out the presence/absence of features and is fairly effective in letting the classifiers learn the patterns for different digits.
    - Faces:

- Approach: Each 70×60 ASCII face image was divided into 5×55 blocks. For each block, a binary feature was assigned based on whether the number of marked pixels exceeded a predefined threshold.
- Block-based feature extraction reduces dimensionality and hence captures the local structure, which is very important for deciding between a face and a non-face image.
- Implementation of Algorithms

3. Algorithm Implementation:
   - Naive Bayes:
     - Implemented from scratch, this classifier computes class priors and conditional probabilities for each feature, applying Laplace smoothing to handle zero probabilities.
   - Multi-Class Perceptron (One-vs-Rest):
     - Also implemented from scratch, this classifier trains separate binary Perceptrons for each class. During prediction, it selects the class with the highest activation score.
   - Justification: Implementing these algorithms manually ensures a deep understanding of their mechanics and allows for customization beyond standard library offerings.
4. Hyperparameter Tuning:
   - Naive Bayes: The smoothing parameter α\alphaα was tuned to balance the influence of feature counts and to prevent zero probabilities.
   - Perceptron: The learning rate and the number of iterations (max_itermax_itermax_iter) were adjusted to optimize convergence speed and accuracy.
   - Rationale: Hyperparameter tuning is crucial for optimizing model performance. Proper tuning ensures that models generalize well to unseen data without overfitting.
5. Performance Evaluation:
   - Training Size Variation: Classifiers were trained on subsets of the training data varying from 10% to 100% in 10% increments. This approach assesses how increasing data availability impacts both training time and prediction accuracy.
   - Repeats: Each training size was evaluated multiple times (repeats) to account for variability due to random sampling, ensuring that results are statistically robust.
   - Metrics: Training time and prediction accuracy (along with standard deviation) were recorded for each training size.
   - Visualization: Results were plotted to provide clear insights into the trade-offs between training time and accuracy across different training sizes.

Results: Summary & Analysis

The table (Appendix C) summarizes the performance of the Naive Bayes and Perceptron classifiers for both digits and faces datasets at varying training sizes, ranging from 10% to 100%. It includes the mean accuracy, standard deviation of accuracy, and mean training time.

Naive Bayes (Digits)
Accuracy:

Accuracy starts at 74.6% with 10% of the training data and increases steadily, stabilizing at 77.5% with 100% of the data.

The improvement is gradual, with diminishing returns as more data is added, indicating that the model captures most patterns early on.
Consistent performance across all fractions, with negligible standard deviation (0.0–0.006), highlights the robustness of Naive Bayes.
Training Time:

Training time remains minimal throughout, increasing slightly from 0.002 seconds at 10% to 0.011 seconds at 100%.
This efficiency reinforces Naive Bayes' suitability for tasks with large datasets or time constraints.

Perceptron (Digits)
Accuracy:

Accuracy starts higher than Naive Bayes at 78.6% with 10% of the data and peaks at 81.8% around 40% training size. However, it declines slightly to 80.2% at 100%.
This drop suggests overfitting or challenges in convergence as training size increases.
Training Time:

Training time grows significantly, from 0.278 seconds at 10% to a substantial 66.568 seconds at 100%.
The iterative nature of Perceptron training, combined with the multi-class setup for digits, drives the exponential increase in time.

Naive Bayes (Faces)
Accuracy:

Accuracy is high even with small training sizes, starting at 88% with 10% and stabilizing around 87.3% from 40% to 100%.
The consistent and high accuracy indicates that Naive Bayes effectively models this binary classification task, where features are relatively easy to separate.
Training Time:

Training time is negligible throughout, staying below 0.002 seconds even at 100% training size.
The simplicity of the binary task and the efficiency of the Naive Bayes algorithm contribute to this result.

Perceptron (Faces)
Accuracy:

Accuracy starts at 81.0% with 10% of the data and improves to 85.3% by 30% training size. However, it declines to 81.3% at 100%.
Similar to digits, the decline at higher training sizes suggests potential overfitting or difficulties with feature representation.
Training Time:

Training time grows steadily from 0.003 seconds at 10% to 0.744 seconds at 100%.
While slower than Naive Bayes, the time increase is manageable due to the binary nature of the task.

Graphical Analysis

Digits Training Time vs. Training Size (Appendix A1)

The graph plots the training time for Naive Bayes versus Perceptron classifiers when training sizes increase from 10% to 100% for the task of digit recognition.

Naive Bayes: Training time is almost constant, independent of training size. It reflects its computational efficiency as it only needs some feature counting and direct probability estimates. Naive Bayes will be ideal in scenarios of large datasets where the training speed is crucial.

Perceptron: With increased training sizes, especially over 60%, the time taken significantly increases to over 60 seconds for a 100% training size. This can be attributed to the additional computational expense of weight updates in an iterative manner for every class in the multiclass setting.

Comparison: Naive Bayes is substantially faster and hence better for applications that are time-bound. On the other hand, the perceptron might offer slightly improved accuracy on tasks requiring more complicated decision boundaries.

Key Insights: This graph shows the scalability and efficiency of Naive Bayes, while Perceptron is much costlier computationally, hence drawing a line between the balance of training time with probable accuracy. The selection of an algorithm should be made on these grounds concerning task requirements and resource constraints.

Digits Prediction Error vs. Training Size (Appendix A2)

The graph compares the prediction error of Naive Bayes and Perceptron classifiers as the training size increases from 10% to 100% for digit recognition.

Naive Bayes: The prediction error decreases significantly as training size grows, starting around 25% at 10% training size and stabilizing near 22% after 50% training size. This shows that Naive Bayes quickly captures meaningful patterns in the data with minimal training, but its performance plateaus as the model relies on simplified assumptions about feature independence.

Perceptron: The prediction error for Perceptron decreases faster than Naive Bayes, starting around 21% at 10% training size and stabilizing near 19% with full training data. This indicates that the Perceptron benefits more from additional training data, likely due to its ability to learn complex decision boundaries.

Comparison: Perceptron consistently outperforms Naive Bayes in terms of prediction error, particularly as the training size increases. The margin of improvement is more pronounced for smaller training sizes, emphasizing Perceptron's ability to better generalize with additional data.

Key Insights:

Performance: While both classifiers perform well, Perceptron achieves lower prediction error, showcasing its advantage in learning more flexible decision boundaries.
Stability: The error bars (standard deviation) suggest that both models are stable, but Naive Bayes exhibits slightly more consistent performance across training sizes.
Application Trade-offs: Naive Bayes is suitable for quick, moderate-accuracy models, while Perceptron requires more data and computational effort to achieve lower error rates. The choice between the two depends on the balance between time constraints and performance requirements.

Faces Training Time vs. Training Size (Appendix B1)

The graph compares the training time of Naive Bayes and Perceptron classifiers as the training size increases from 10% to 100% for the face detection task.

Naive Bayes: Training time remains consistently low across all training sizes, with negligible increases as the training size grows. This highlights Naive Bayes' computational efficiency, as it primarily involves simple counting operations and probability computations, making it ideal for scenarios where quick training is required.

Perceptron: Training time increases progressively with larger training sizes, showing a steep rise beyond 60% and reaching over 0.7 seconds at 100% training size. The iterative weight update process, even for this binary classification task, contributes to this growth.

Comparison: Naive Bayes is orders of magnitude faster than Perceptron across all training sizes, emphasizing its suitability for time-sensitive applications. However, Perceptron's computational cost is still relatively manageable in this binary classification task compared to its multi-class performance on the digit dataset.

Key Insights:

Efficiency: Naive Bayes demonstrates superior scalability and efficiency, making it more appropriate for large-scale face detection tasks where training time is a key factor.
Applicability: Perceptron's training time, though higher, is still within acceptable limits for moderate-sized datasets. It provides a feasible trade-off when slightly improved accuracy is required.
Scalability: The training time difference between the classifiers grows significantly with training size, reinforcing the importance of considering computational resources when selecting a classifier.

Faces Prediction Error vs. Training Size (Appendix B2)

The graph compares the prediction error of Naive Bayes and Perceptron classifiers as the training size increases from 10% to 100% for the face detection task.

Naive Bayes: Prediction error remains consistently low, starting around 13% at smaller training sizes and stabilizing near 12% as training size increases. The low error reflects Naive Bayes' effectiveness in this binary classification task, particularly when the features are well-separated and straightforward to learn.

Perceptron: Prediction error begins higher, fluctuating significantly at smaller training sizes due to variability in the data used. As training size increases, the error stabilizes around 16% but remains consistently higher than Naive Bayes. This suggests that Perceptron struggles with the chosen block-based features or the simplicity of the task.

Comparison: Naive Bayes outperforms Perceptron consistently across all training sizes, achieving lower prediction error with smaller variability. The stability and effectiveness of Naive Bayes are evident, while Perceptron's performance is more erratic, likely due to its reliance on iterative updates that may not fully converge with limited data.

Key Insights:

Stability: Naive Bayes provides a stable and reliable solution, with low error and minimal fluctuation, making it ideal for face detection tasks using these features.
Sensitivity to Data Size: Perceptron's error fluctuates more at smaller training sizes, indicating its sensitivity to training data and potential overfitting or underfitting when data is insufficient.
Practical Recommendation: Naive Bayes' consistently low error and faster training time make it the preferable choice for this task, especially when computational efficiency and reliability are critical. Perceptron might not justify its additional computational cost given its higher and less stable prediction error.

Key Insights & learnings

This project yielded several critical learnings and observations regarding the implementation and performance of Naive Bayes and Perceptron classifiers:

1. Classifier Strengths and Weaknesses:
    - Naive Bayes:
        - Strengths: Exceptional computational efficiency and scalability, making it ideal for large datasets. Its simplicity allows for quick implementation and interpretation.
        - Weaknesses: The strong independence assumption between features can limit its performance, especially in datasets where feature interactions are significant.
    - Perceptron:
        - Strengths: Potentially higher accuracy by learning decision boundaries that account for feature interactions. More flexible in handling complex classification tasks.

        - Weaknesses: Computationally intensive, especially in multi-class scenarios. Sensitive to hyperparameter settings like learning rate and number of iterations.
2. Hyperparameter Tuning:
    - Effective tuning of hyperparameters such as α\alphaα for Naive Bayes and learning rate and max_itermax_itermax_iter for Perceptron was crucial in optimizing performance. Poorly chosen hyperparameters can lead to underfitting or overfitting, adversely affecting accuracy and training time.
    - For Naive Bayes, α=1.0\alpha = 1.0α=1.0 provided a good balance, preventing zero probabilities while not overly smoothing the feature distributions.
    - For Perceptron, a learning rate of 0.01 and max_iter=500max_iter = 500max_iter=500 were effective in achieving convergence without excessive training times.
3. Time vs. Performance Trade-offs:
    - There exists a clear trade-off between training time and accuracy. While Naive Bayes offered faster training, especially beneficial in time-constrained or real-time applications, Perceptron provided marginally better accuracy at the cost of increased computational resources.
    - In multi-class tasks like digit recognition, the Perceptron's training time surged due to the need to train multiple binary classifiers, highlighting scalability concerns.
4. Real-World Constraints:
    - Computational Resources: In scenarios with limited computational power or requiring rapid model updates, Naive Bayes is preferable due to its low training time.
    - Accuracy Requirements: For applications where higher accuracy is paramount and computational resources are available, the Perceptron may be more suitable despite longer training times.
    - Data Characteristics: The nature of the data influences classifier choice. For instance, Naive Bayes performed exceptionally well on the binary face detection task, possibly due to the clear distinction between classes with simple block-based features.
5. Visualization and Analysis:
    - Plotting training time and prediction error against training sizes provided clear visualizations of how each classifier scales with data. Naive Bayes maintained a flat training time curve, whereas Perceptron's curve increased with training size.
    - The prediction error plots underscored the point of diminishing returns, where adding more data beyond a certain threshold yielded minimal accuracy improvements. This reinforced the decision to consider 70% accuracy as a satisfactory benchmark.
6. Learning and Development:
    - Implementation Challenges: Manually implementing classifiers reinforced a deeper understanding of their operational mechanics, beyond theoretical knowledge. Debugging and optimizing these implementations highlighted the importance of efficient coding practices.
    - Feature Engineering Impact: Simple feature extraction methods proved surprisingly effective, emphasizing that even basic features can capture essential patterns. However, more sophisticated feature engineering could potentially enhance performance further.
    - Model Selection Considerations: The project highlighted the necessity of aligning model selection with specific application requirements, balancing factors like accuracy needs, computational constraints, and data complexity.

7. Avoiding Common Pitfalls:
    - Data Leakage Prevention: Ensuring that test data was strictly separated from training processes was paramount to obtaining unbiased performance metrics. This practice is critical in real-world applications to ensure model generalizability.
    - Overfitting Awareness: Monitoring accuracy on both training and test sets helped in identifying overfitting scenarios, guiding hyperparameter adjustments to achieve better generalization.
8. Future Directions:
    - Enhanced Feature Extraction: Exploring more advanced feature extraction techniques, such as edge detection or dimensionality reduction methods (e.g., PCA), could improve classifier performance.
    - Algorithm Extensions: Implementing more sophisticated classifiers, like Support Vector Machines or Neural Networks, would provide a broader comparative analysis and potentially higher accuracy.
    - Cross-Validation: Incorporating k-fold cross-validation could offer more robust performance estimates, especially in scenarios with limited data.

Conclusion

This project successfully implemented and compared Naive Bayes and Perceptron classifiers on digit and face datasets, demonstrating the practical trade-offs between computational efficiency and classification accuracy. The classifiers achieved accuracies exceeding the 70% benchmark, indicating their viability for real-world applications where such performance levels are deemed sufficient. The project's success lies in its comprehensive approach—from data preprocessing and feature extraction to algorithm implementation and rigorous evaluation—ensuring reliable and insightful results.

Summary of Results:

- Naive Bayes consistently trained faster across both datasets, achieving respectable accuracies that surpassed the 70% threshold.
- Perceptron offered marginally better accuracy on the digit dataset at the expense of increased training time, while on the face dataset, its performance was competitive but not superior to Naive Bayes.
- Both classifiers demonstrated the effectiveness of simple feature extraction methods, though there remains room for improvement through more sophisticated techniques.

Project Success Factors:

- Comprehensive Implementation: Successfully implementing classifiers from scratch provided deep insights into their operational dynamics and facilitated tailored optimizations.
- Rigorous Evaluation: Systematically varying training sizes and conducting multiple repeats ensured the robustness and reliability of the results.
- Effective Feature Engineering: Even with straightforward feature extraction methods, the classifiers performed adequately, highlighting the fundamental importance of feature selection in machine learning.

Future Work: Building upon the foundations laid in this project, future endeavors could explore more advanced feature extraction methods to potentially enhance classifier performance. Additionally,

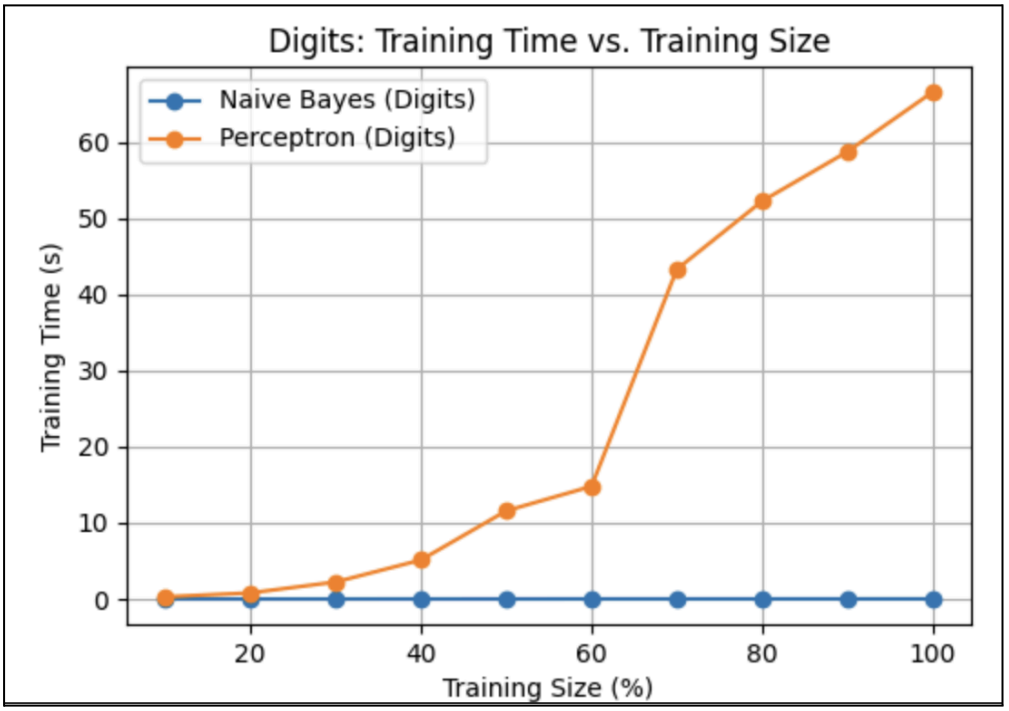
experimenting with more sophisticated algorithms, such as Support Vector Machines or deep learning models, could provide a broader understanding of classification techniques. Incorporating cross-validation and optimizing algorithms for scalability would further strengthen the evaluation framework, ensuring even more robust and generalizable results.

In conclusion, this project not only achieved its primary objectives but also provided valuable lessons in algorithm implementation, hyperparameter tuning, and performance evaluation—skills that are indispensable in the ever-evolving field of machine learning.

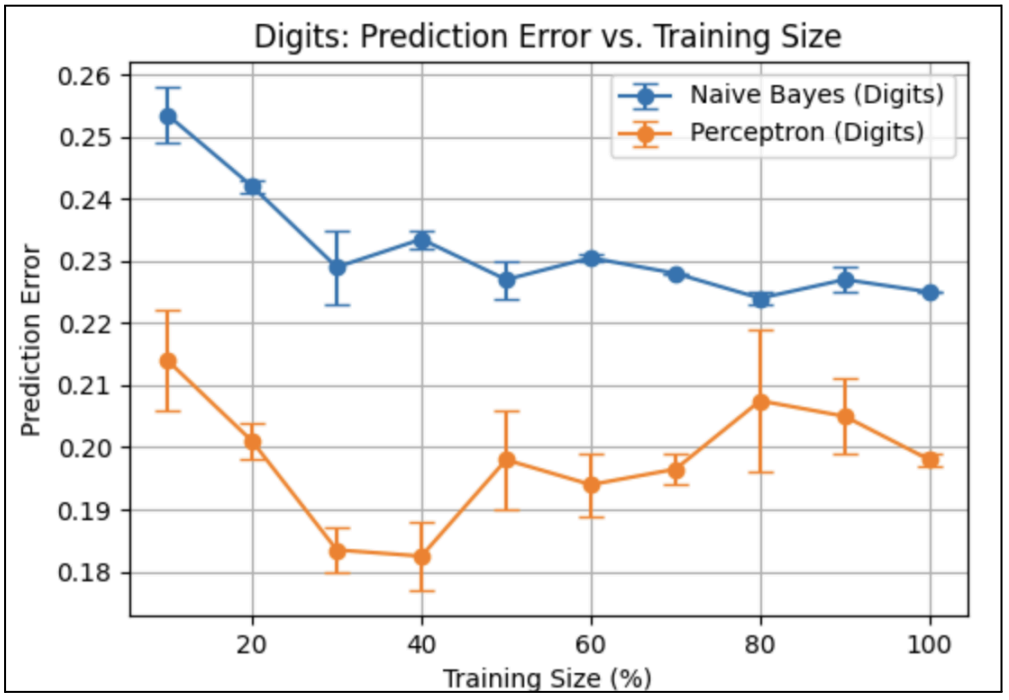
Appendices:

Appendix A: Digits

Appendix A1: Training time vs. training size

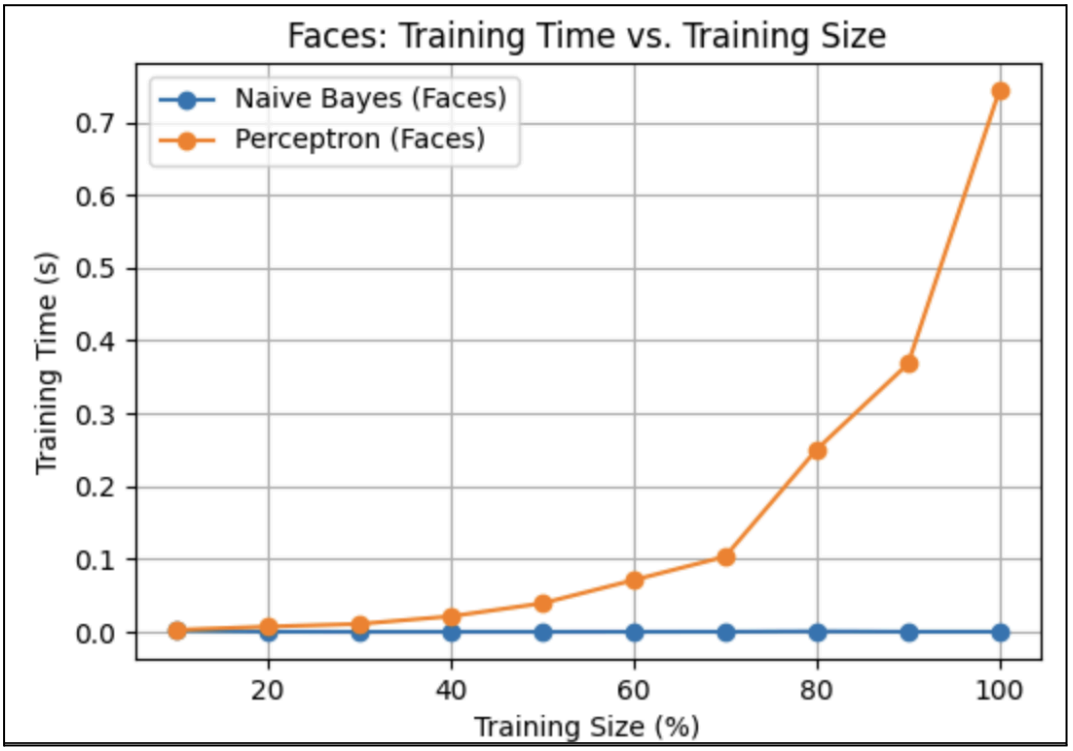


Appendix A2: Prediction error vs. training size

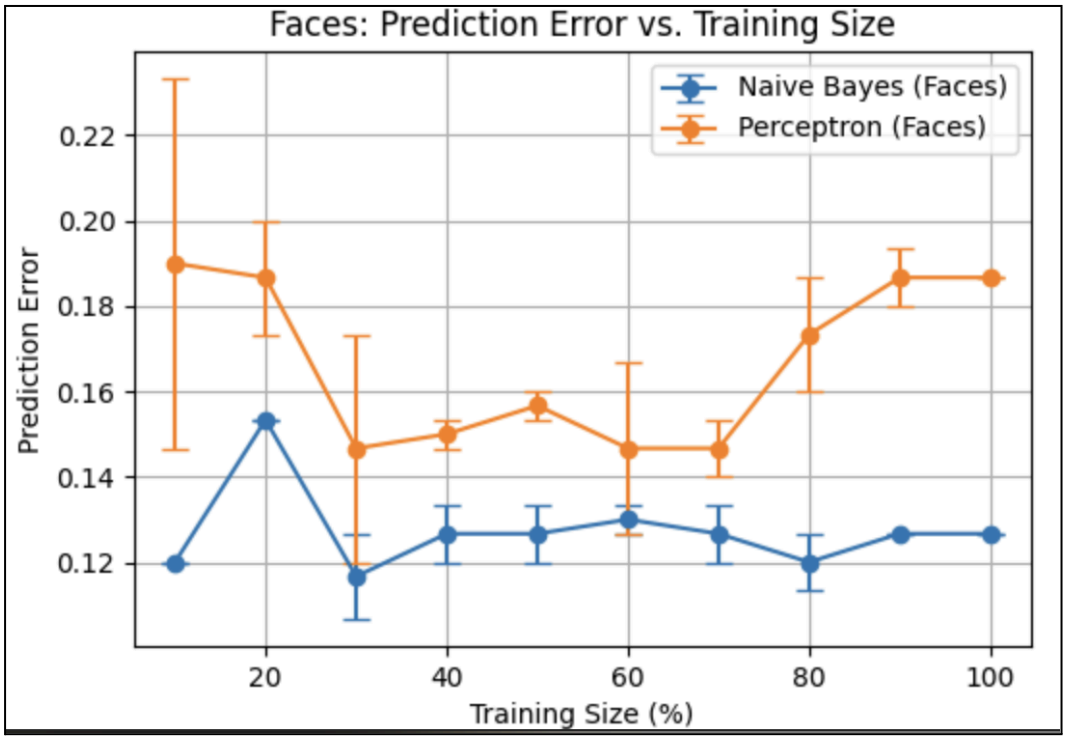


Appendix B: Faces

Appendix B1: Training time vs training size



Appendix B2: Prediction Error vs. Training Size



Appendix C: Results

```
Naive Bayes (Digits)
Frac% | MeanAcc | StdAcc | MeanTrainTime(s)
10% | 0.746 | 0.005 | 0.002
20% | 0.758 | 0.001 | 0.002
30% | 0.771 | 0.006 | 0.003
40% | 0.766 | 0.002 | 0.005
50% | 0.773 | 0.003 | 0.006
60% | 0.770 | 0.001 | 0.008
70% | 0.772 | 0.000 | 0.008
80% | 0.776 | 0.001 | 0.009
90% | 0.773 | 0.002 | 0.010
100% | 0.775 | 0.000 | 0.011
Perceptron (Digits)
Frac% | MeanAcc | StdAcc | MeanTrainTime(s)
10% | 0.786 | 0.008 | 0.278
20% | 0.799 | 0.003 | 0.796
30% | 0.817 | 0.004 | 2.229
40% | 0.818 | 0.005 | 5.120
50% | 0.802 | 0.008 | 11.572
60% | 0.806 | 0.005 | 14.792
70% | 0.804 | 0.003 | 43.284
80% | 0.792 | 0.012 | 52.234
90% | 0.795 | 0.006 | 58.727
100% | 0.802 | 0.001 | 66.568
Naive Bayes (Faces)
Frac% | MeanAcc | StdAcc | MeanTrainTime(s)
10% | 0.880 | 0.000 | 0.002
20% | 0.847 | 0.000 | 0.000
30% | 0.883 | 0.010 | 0.000
40% | 0.873 | 0.007 | 0.000
50% | 0.873 | 0.007 | 0.000
60% | 0.870 | 0.003 | 0.000
70% | 0.873 | 0.007 | 0.000
80% | 0.880 | 0.007 | 0.001
90% | 0.873 | 0.000 | 0.000
100% | 0.873 | 0.000 | 0.000
Perceptron (Faces)
Frac% | MeanAcc | StdAcc | MeanTrainTime(s)
10% | 0.810 | 0.043 | 0.003
20% | 0.813 | 0.013 | 0.007
30% | 0.853 | 0.027 | 0.011
40% | 0.850 | 0.003 | 0.021
50% | 0.843 | 0.003 | 0.039
60% | 0.853 | 0.020 | 0.071
70% | 0.853 | 0.007 | 0.103
80% | 0.827 | 0.013 | 0.251
90% | 0.813 | 0.007 | 0.368
100% | 0.813 | 0.000 | 0.744
```