CS440 - Final Project Report: Face and Digit Classification

Professor Abdeslam Boularias

Authors: Stephanie Lin (sl1747), Katie Sidebotham (kgs87), Jasper Vanderree (jpv65)

May 03, 2023

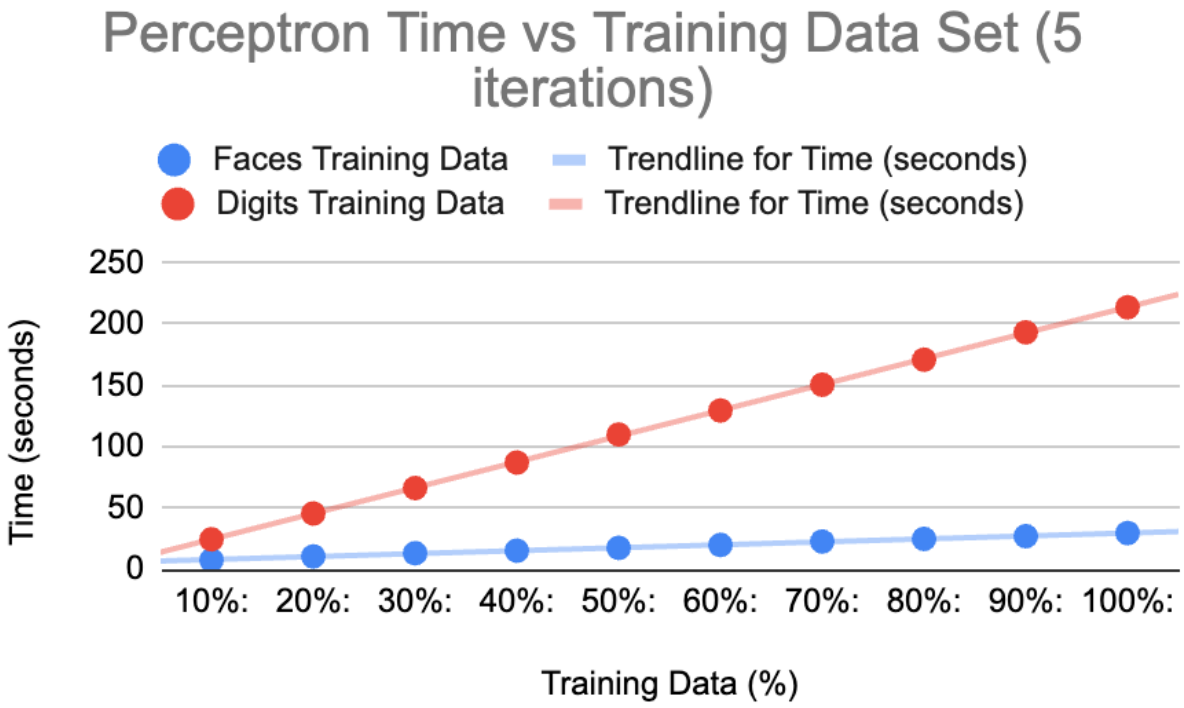
Part A: Perceptron Classifier

In our implementation of perceptron, we first initialize the class with the legal labels, maximum number of 3 iterations, and the weight values for each label as a Counter object.

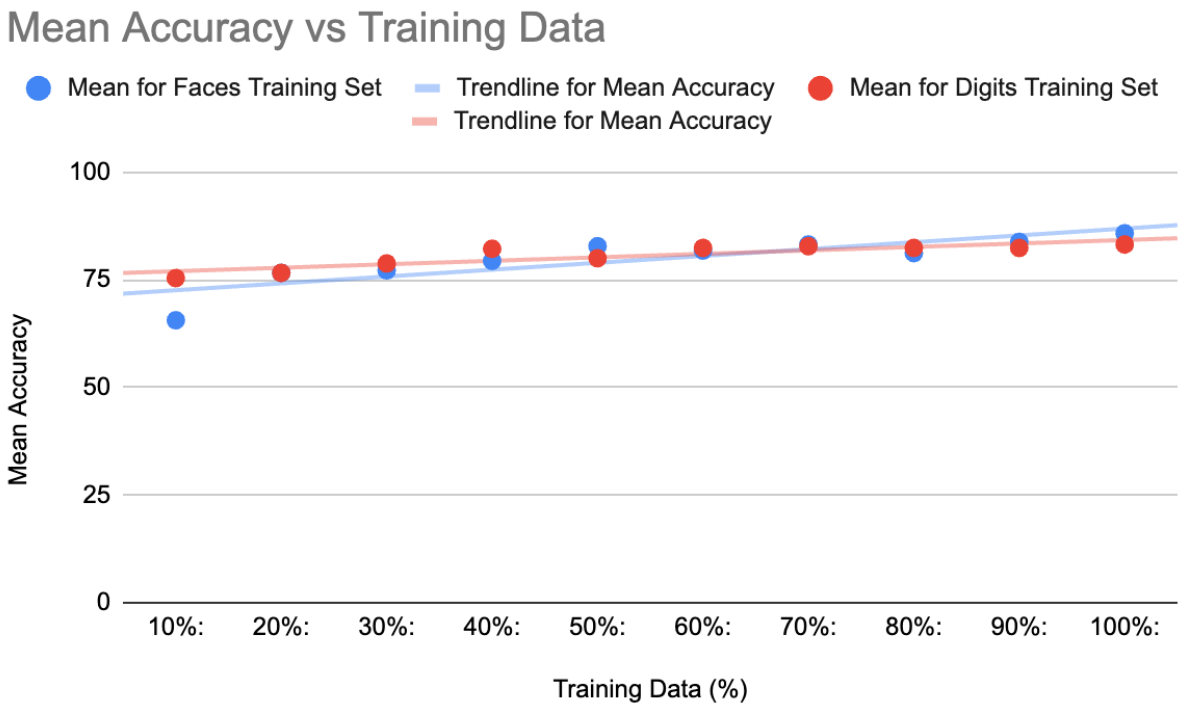
Training:
The train() method iteratively updates the weight values based on the training data and labels. The for loop goes through each training datum and calculates the dot product of the feature vector and weight vector. If the predicted label is incorrect, then the weight vector is updated. The goal of this method is to achieve a higher accuracy on the validation data by updating the weights of the algorithm.

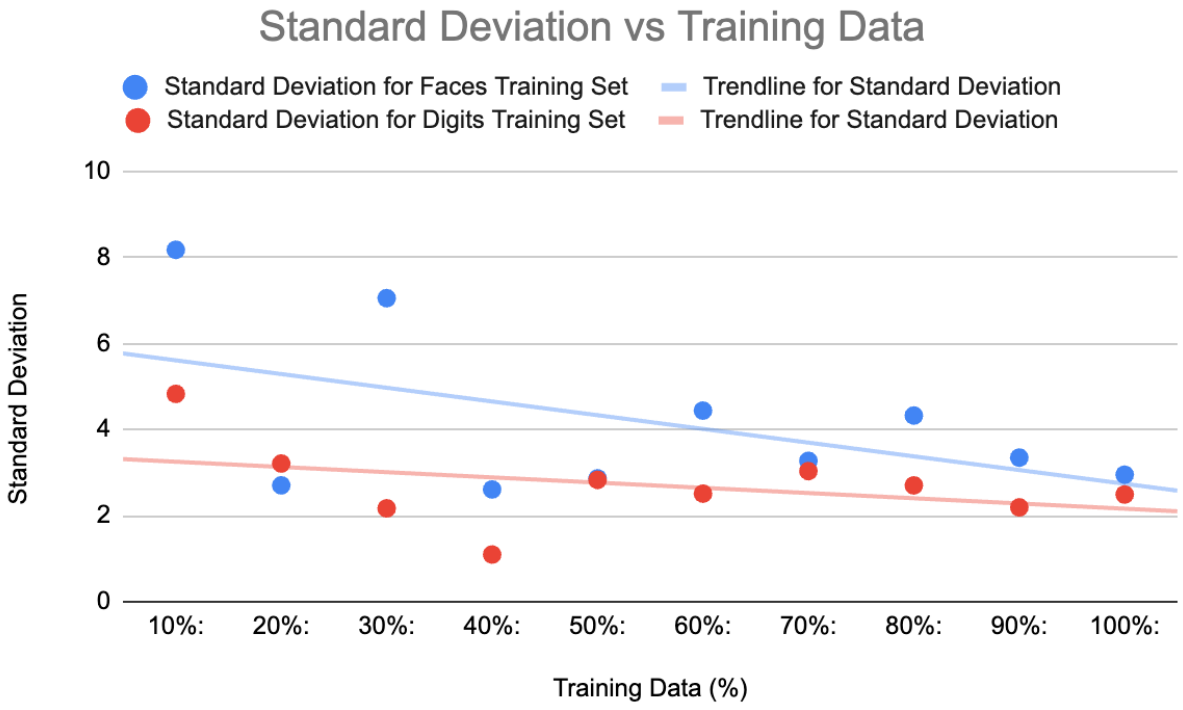
Classify:
The classify() method returns a list of predicted labels for each datum in data. The method calculates the dot product between the weight vector and the feature vector then finds the highest dot product to add to the list of guesses. This is a form of linear classification that generates a line to separate the classes containing the data.

Time Performance: The time performance is based on training the model on maximum of 3 iterations.



Accuracy Performance:





Conclusion:
As we increased the amount of training data used for our model, the time it took to run the program increased (as shown in the graph). Especially when calculating mean and standard deviation on 5 iterations, the runtime went from taking 7.78 seconds to run 5 iterations for 10% of the training data to 29.5 seconds for 5 iterations on 100% of the data. When observing performance, we were able to see an increase in accuracy, as well on both sets of data.

Part B: Naive Bayes Classifier

In our implementation of naive bayes, we first initialize the class with the legal labels, type, and the k variables. The k variable is the smoothing parameter for the training method.

Train:
The train method converts the trainingData to a list and initializes the features variable to a list of different features in the training set. The kgrid is either set to self.k or a range of k values based on the value of automatic tuning. For the sake of this project, we have automatic tuning set to false, and manually implemented the k-value to equal 0.1.

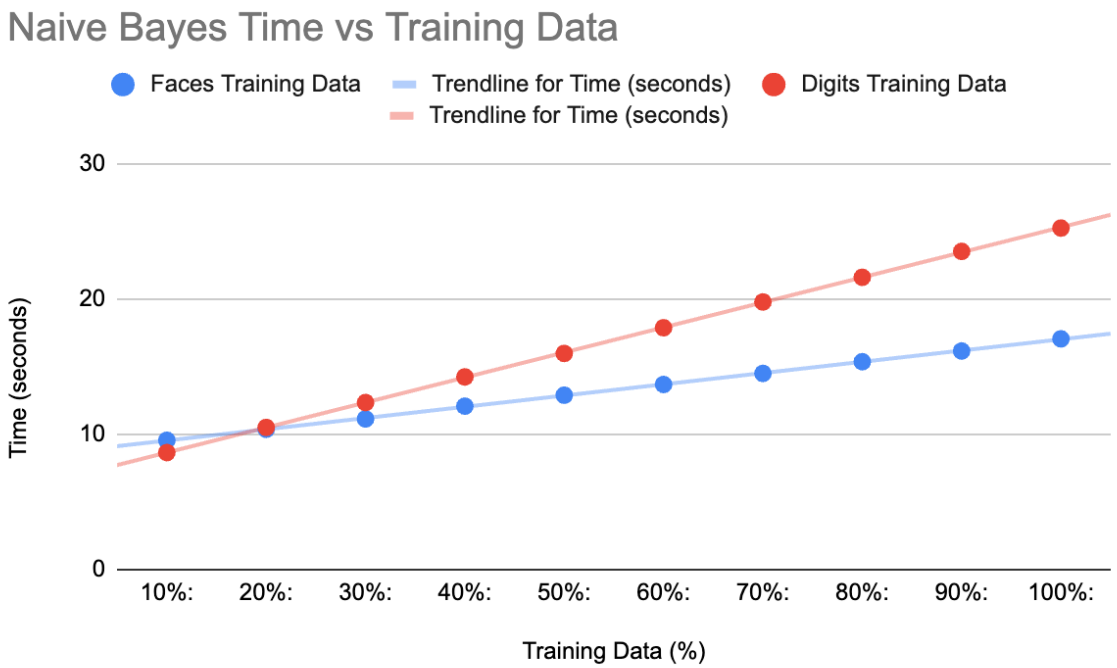
Train and Tune:
This method uses for loop to iterate over the training data and stores the number of features for each label in the featureCounts dictionary. It returns the trained classifier with smoothing using the k as a smoothing parameter to avoid zero probabilities.

Classify:
The classify method computes the log-posterior probability for each label by calling a helper method, calculateLogJointProbabilities(). The highest log-posterior probability is selected as the predicted label and its index is added to the guesses list and returned.

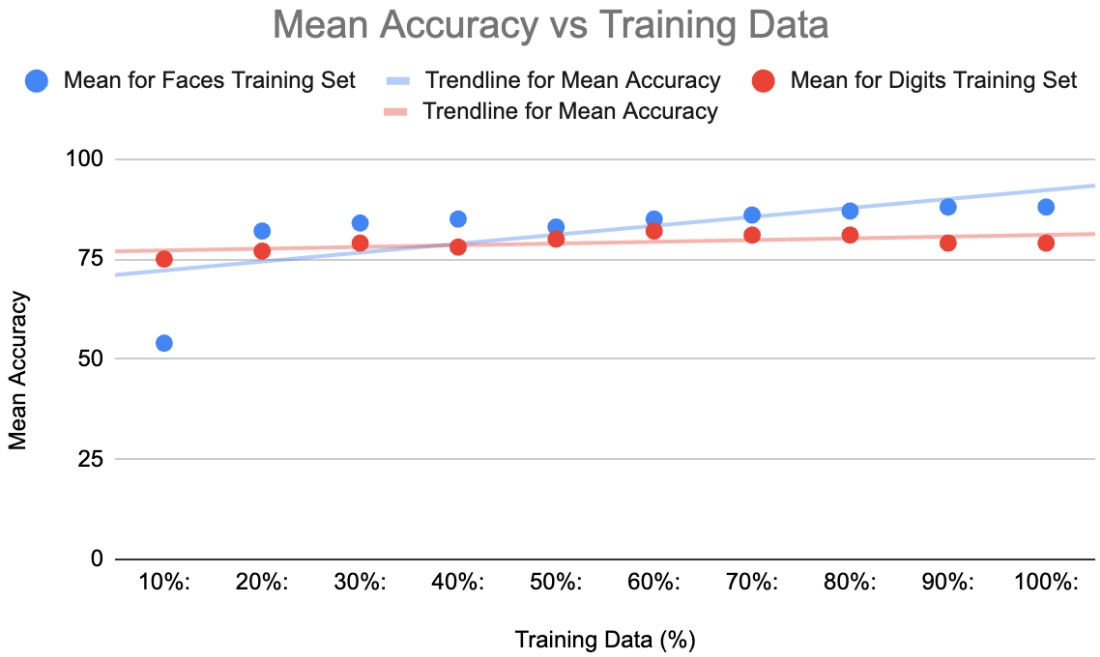
Calculate Log Joint Probability:
For each feature in the datum, the log-prior probability is calculated using featureCounts, count_labels, and k smoothing parameter to avoid zero probabilities. The computation is complete by adding the log-prior probability of feature and label to obtain the log-joint probability.

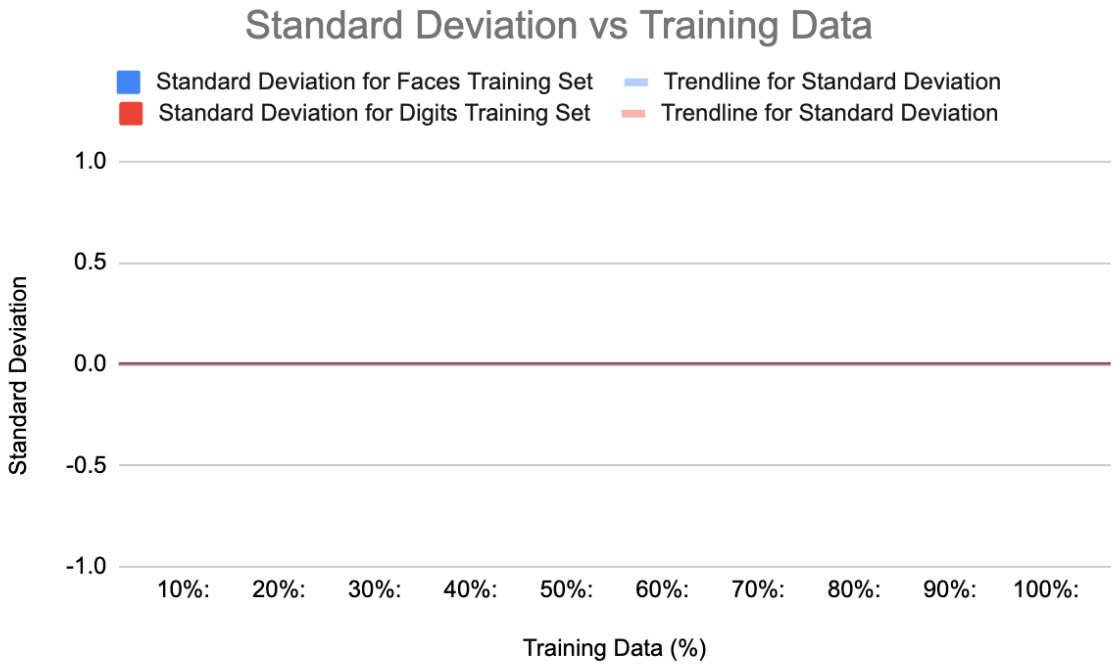
Log- Joint Equation: $log\ P(y)\ +\ log\ P(x|y)$
$log\ P(y)$ represents log-prior probability of label “y”
$log\ P(x|y)$ represents the log probability of the feature vector “x” given label “y”

Time Performance: The time performance is based on training the model on maximum of 3 iterations.



Accuracy Performance:





Conclusion:
As we increased the amount of training data used for our model, the time it took to run the program increased (as shown in the graph). The mean accuracy and standard deviation stayed consistent through each 5 iterations showing that our algorithm for naive bayes is not as sensitive to randomization as the perceptron and mira classifiers.

Part C: Mira Classifier

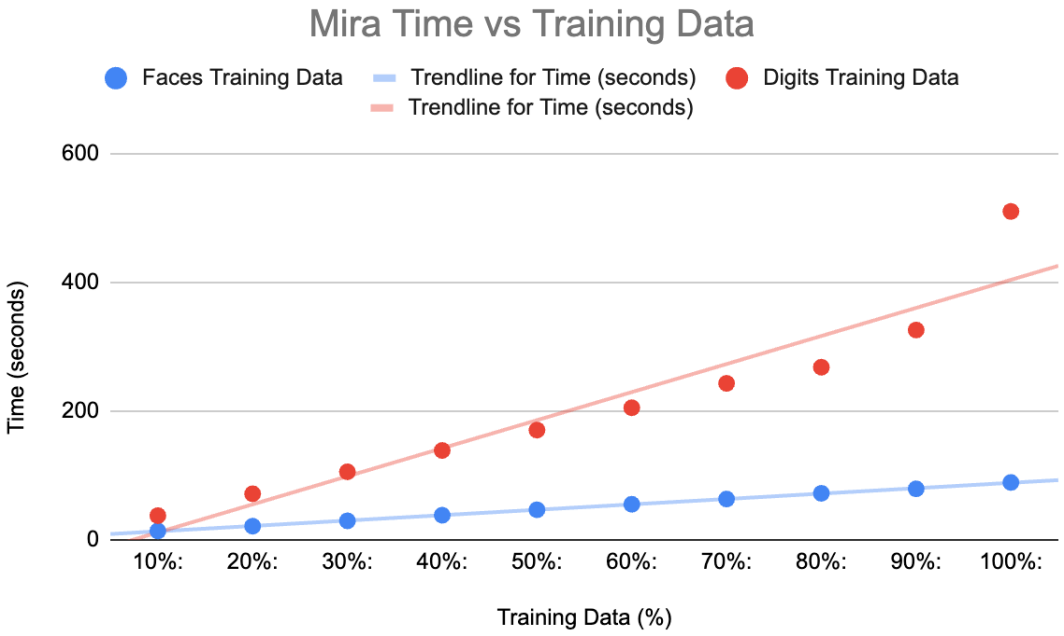
In our implementation of Mira, we first initialize the class with the legal labels, type, c variable for regularization, max iterations, and initializeWeightsToZero() method to set all weights to zero.

Training:
Stores the feature instances of the trainingData into self.features. Depending on the value of automaticTuning the algorithm either uses the C grid values or uses self.C. For the sake of the project, we use self.C

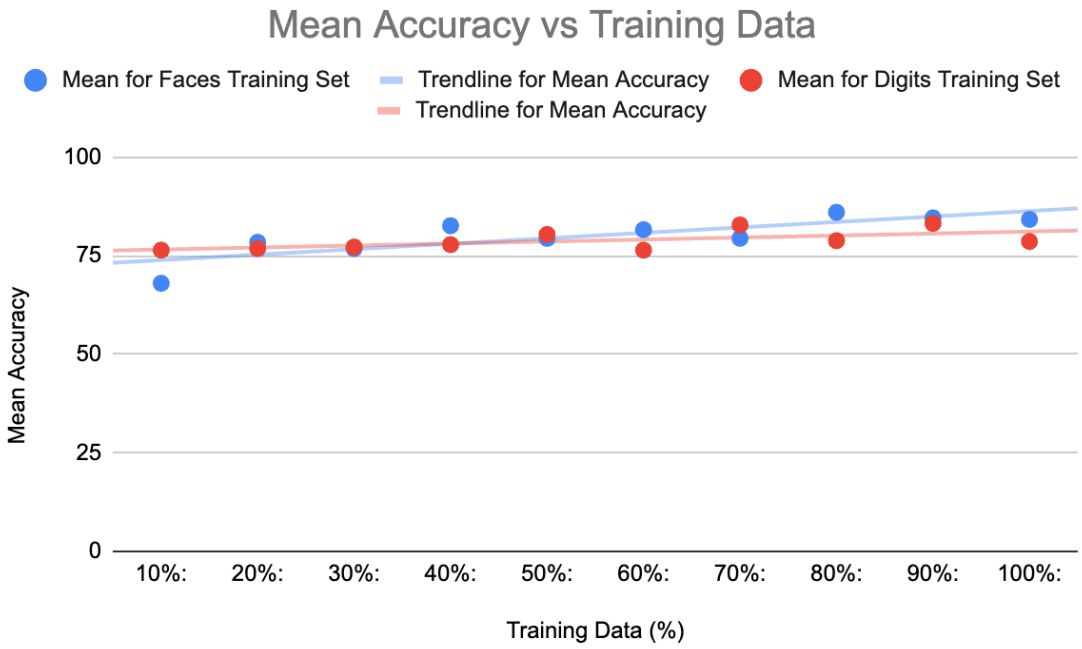
Train and Tune:
The train and tune method selects the weight that gives the best accuracy on the validation data set. The method iterates over the training data to update the weights based on a calculation score for each label using current weights and instances. If the label with the maximum score is not the true label of the instance, the weight is updated with a scaling factor that outputs the best weights found during the for loop.

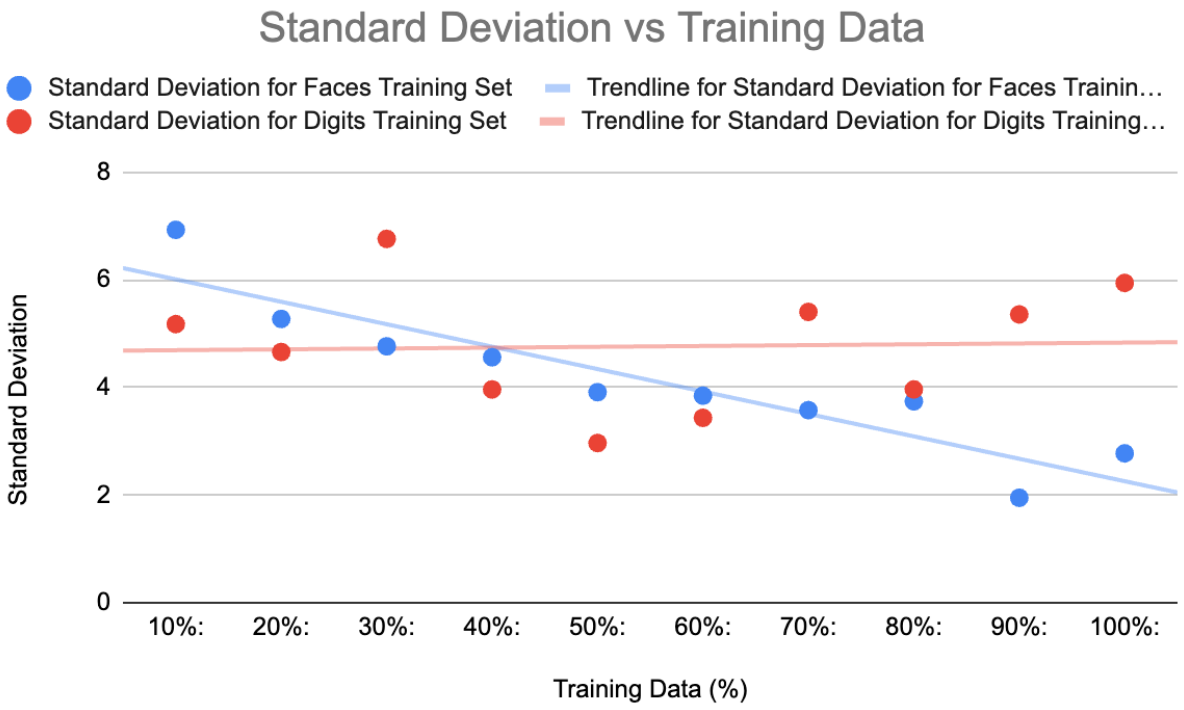
Classify:
The classify method computes the score of each label in the list of instances passed through by finding the dot product of the weights and the feature vector. The goal is to return the label with the highest score as the predicted label for that instance.

Time Performance: The time performance is based on training the model on maximum of 3 iterations



Accuracy Performance:





Conclusion:
As we increased the amount of training data used for our model, the time it took to run the program increased (as shown in the graph). The standard deviation for faces decreased as the training data percentage increased whereas the standard deviation for digits stayed relatively consistent. This could be a result of varying complexity in both data sets as faces may be more complex in appearance compared to digits.

Part D: Results and Learned Lessons

1. Naive Bayes:
    a. Naive Bayes is a probabilistic model that makes assumptions about the independence of features given the class.
    b. Relatively simple to implement and can be trained efficiently on large datasets
2. Perceptron:
    a. Perceptron is a linear classifier that uses a threshold function to classify examples.
3. MIRA:
    a. MIRA is a linear classifier that uses a margin-based objective function to update the weight vector.

b. Lless sensitive to the choice of the learning rate and can converge faster than perceptron.

Overall, the choice of classifier depends on the problem the model is attempting to solve and what type of data is being used to train,validate, and test the model. Naive Bayes is a good choice for text classification and other problems with high-dimensional data, while perceptron and MIRA are better for linear classification problems with non-linearly separable data.