CS440 - Project Image Classification

Professor Abdeslam Boularias

By:
Devvrat Patel (dap299)
Shubham Mittal (sm1841)

December 9, 2019

Part 1 : Image Input and Features

- We used the code provided Berkeley's site to get all the inputs and feature extractions.

Part 2 : Algorithms

- Naive Bayes Algorithm
  - Features - The features we used for Naive Bayes classifier were the set of pixel features. Each pixel could either be 0 (white), or 1 (black/grey).
  - Training and Tuning - To train the program we calculated.

$$P(Y = y) = \frac{\text{number of data with label } Y = y \text{in training set}}{\text{total number of training data}}$$

To get this probability, we had a dictionary with key label Y and values is the total number of occurrences. We iterate over all the data set and increment the label's value by 1 every time. We get the total count for all the labels this way and calculate the probability P(Y=y) for each label and store in dictionary. Then we find the conditional probability,

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

for each legal label, feature, and the legal feature value, we take the count, plus k, and divide by its label count plus total number of legal feature values times k as we need to take the k out of the sum.

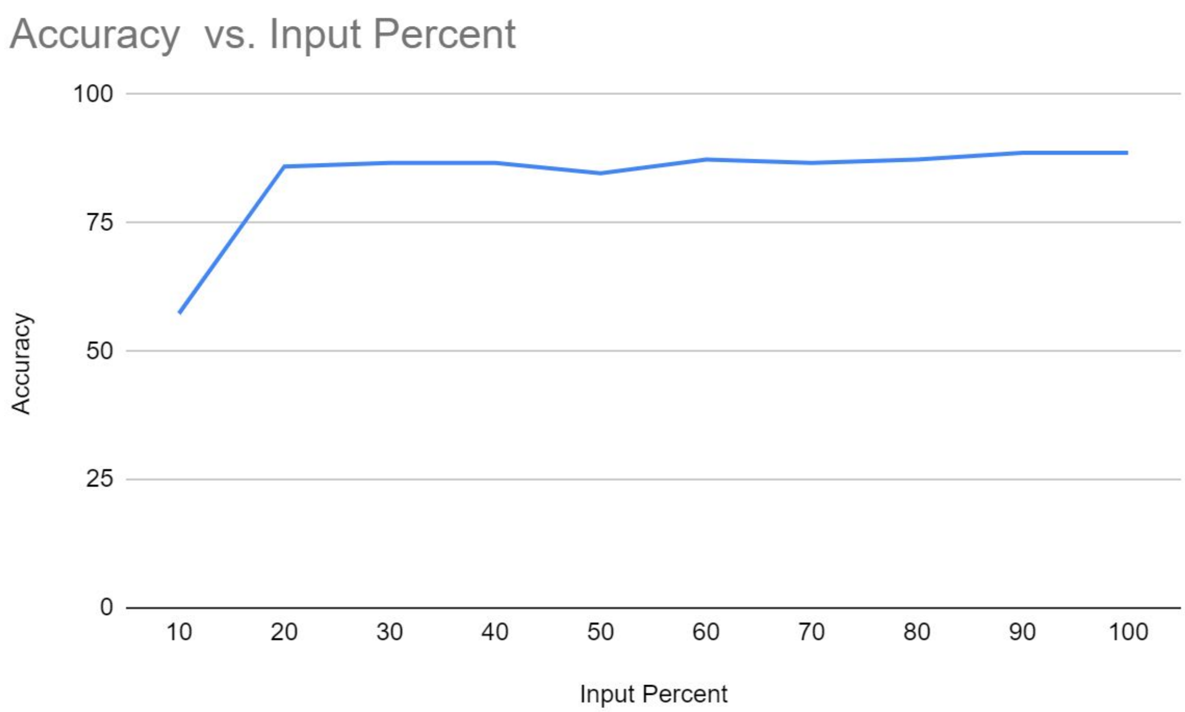
  - Classify - For classifying, we take the log of the probability as the probability will be very small.

$$\log P(y) + \sum_{i=1}^{m} \log P(f_i | y)$$

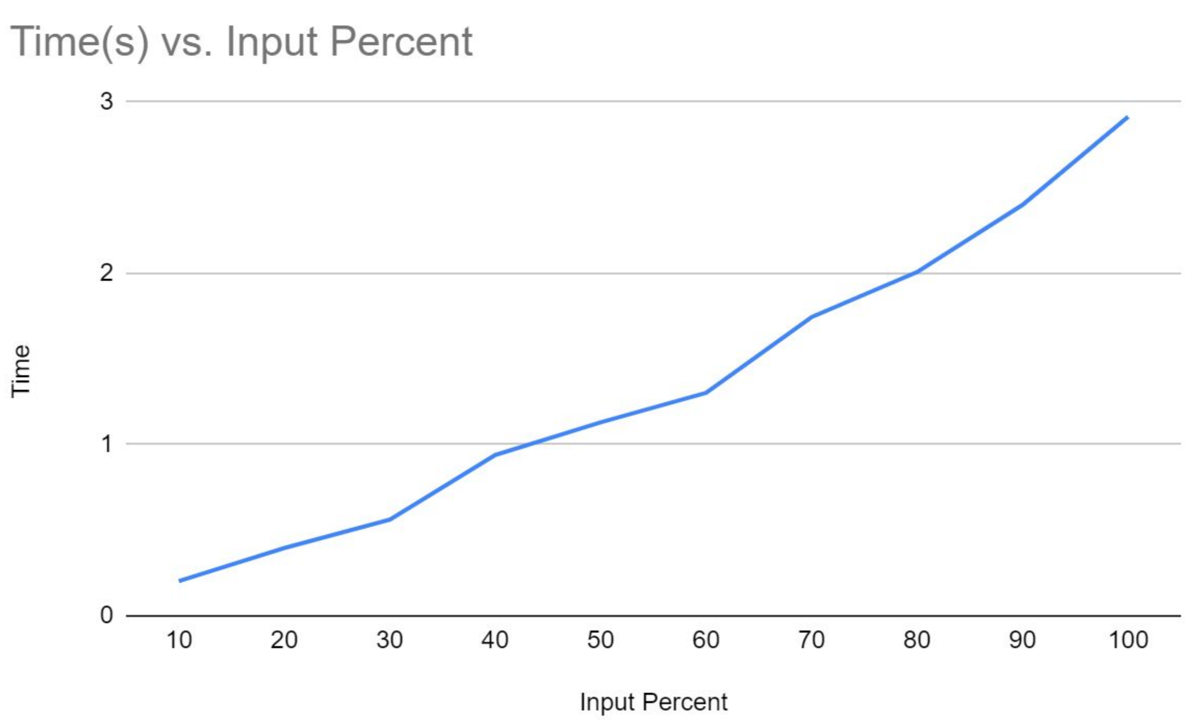
We compute this probability for every legal label using the equation above. The label with the maximum probability will be our guess.

  - The following are the results for both the digits and faces on 10%, 20%, 30% … 100% of the data using Naive Bayes Algorithm.

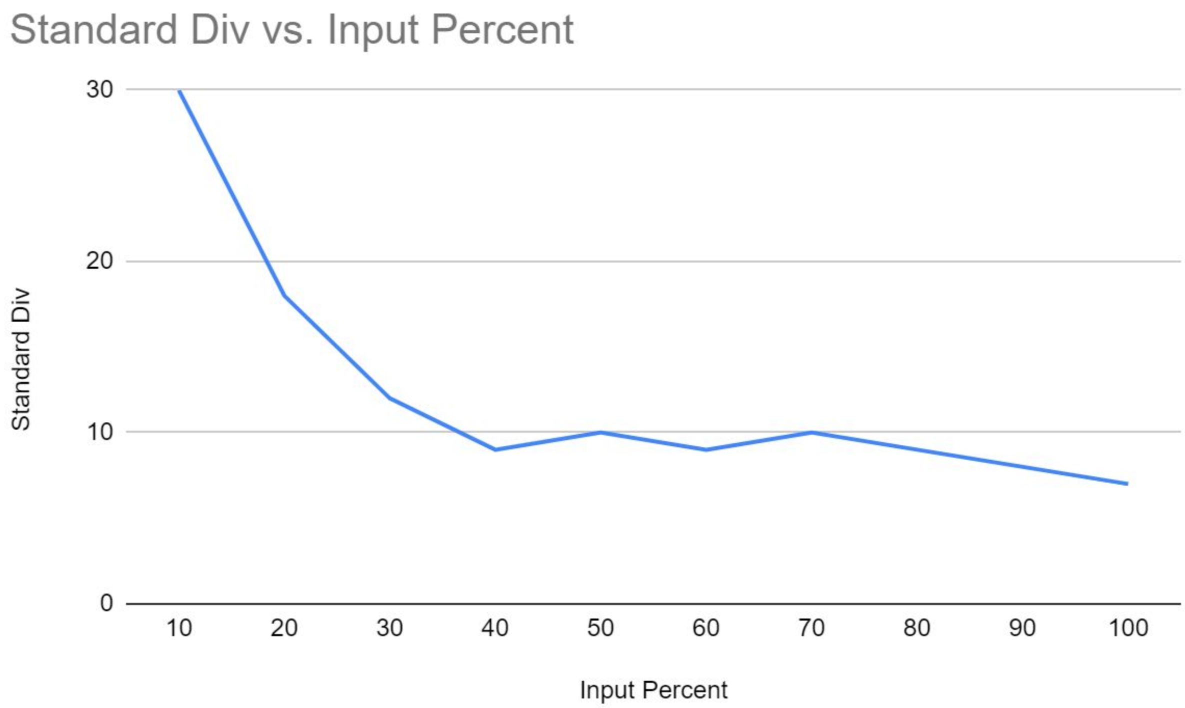
Naive Bayes Faces Accuracy



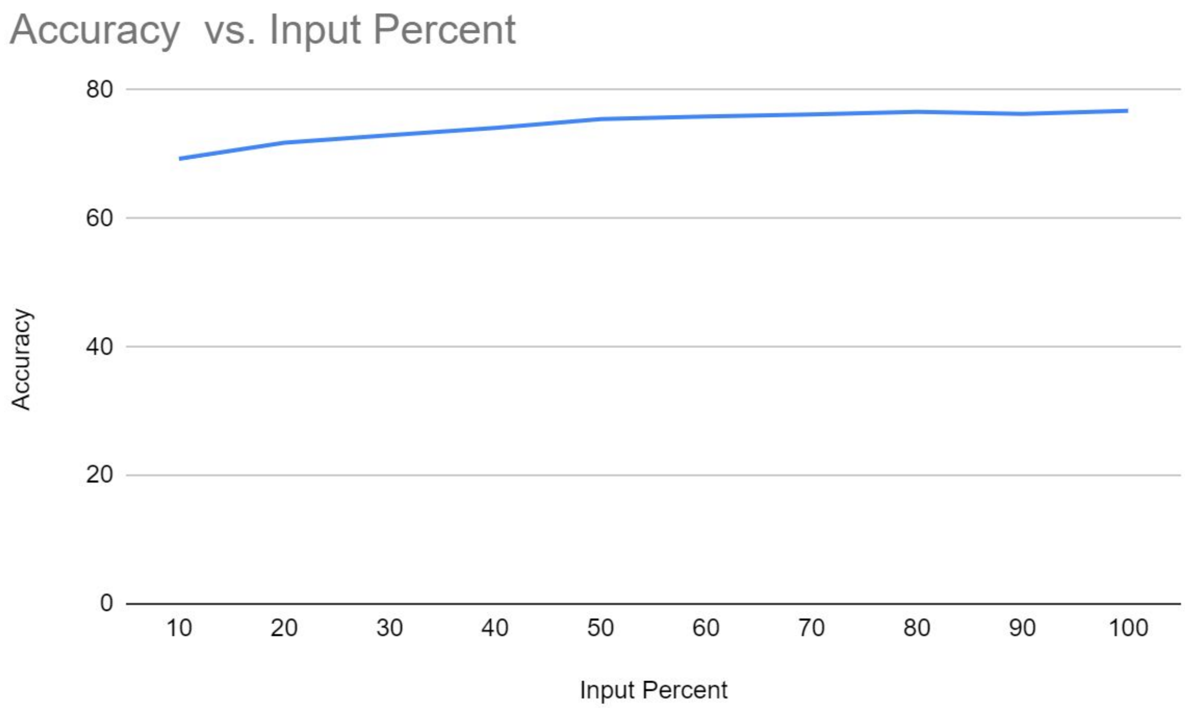
Naive Bayes Faces Timing



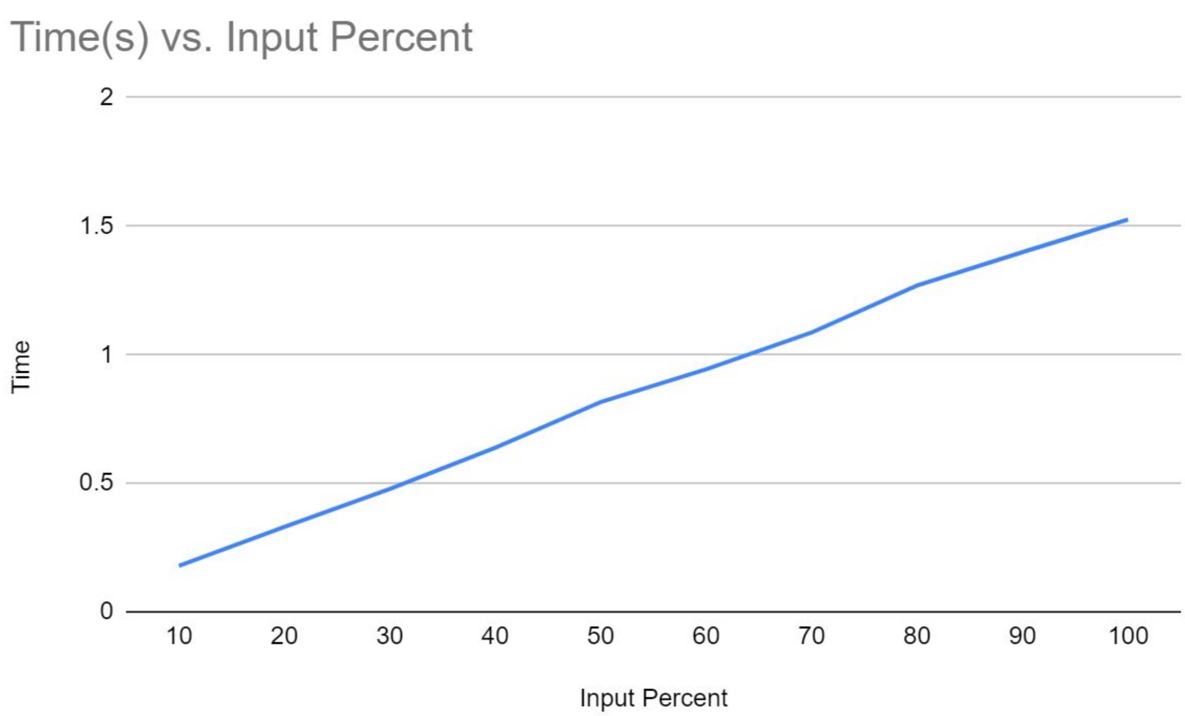
Naive Bayes Faces Stanard Div



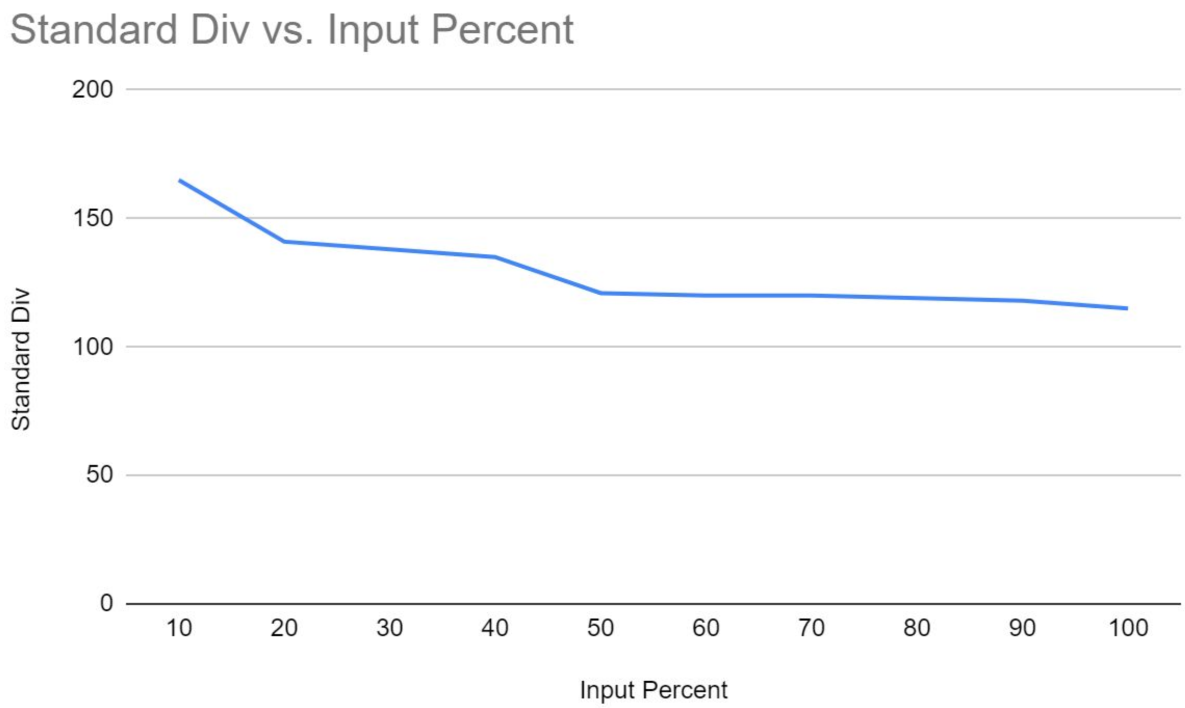
Naive Bayes Digits Accuracy



Naive Bayes Digits Timing



Naive Bayes Digits Standard Div



- Perceptron
  - Features - In this algorithm, we used the same features as Naive Bayes.
  - Weights - We initialize the weights to be 0 in the beginning and then later modify them.
  - Training - We have set the max iteration for perceptron (3) in the beginning. Once we reach the max iteration, we stop training our program. We also keep a list of weights to the features for each legal label. We store these values in the python dictionary to store the lists where the key is the label and the value si the list of weights. Every iteration, we loop loop over the training data, for each datum, and compute a list of numbers corresponding to all the legal labels using the equation below.

    $$f(x_i, w) = w_0 + w_1\phi_1 + ... + w_j\phi_j$$

    We pick the label with the highest f(xi,w) value as our guess. If our guess is correct then we predicted the right answer and no changes are required. If our guess is different then we need to update our weight list for the correct label and incorrect label(s) using the following equations.

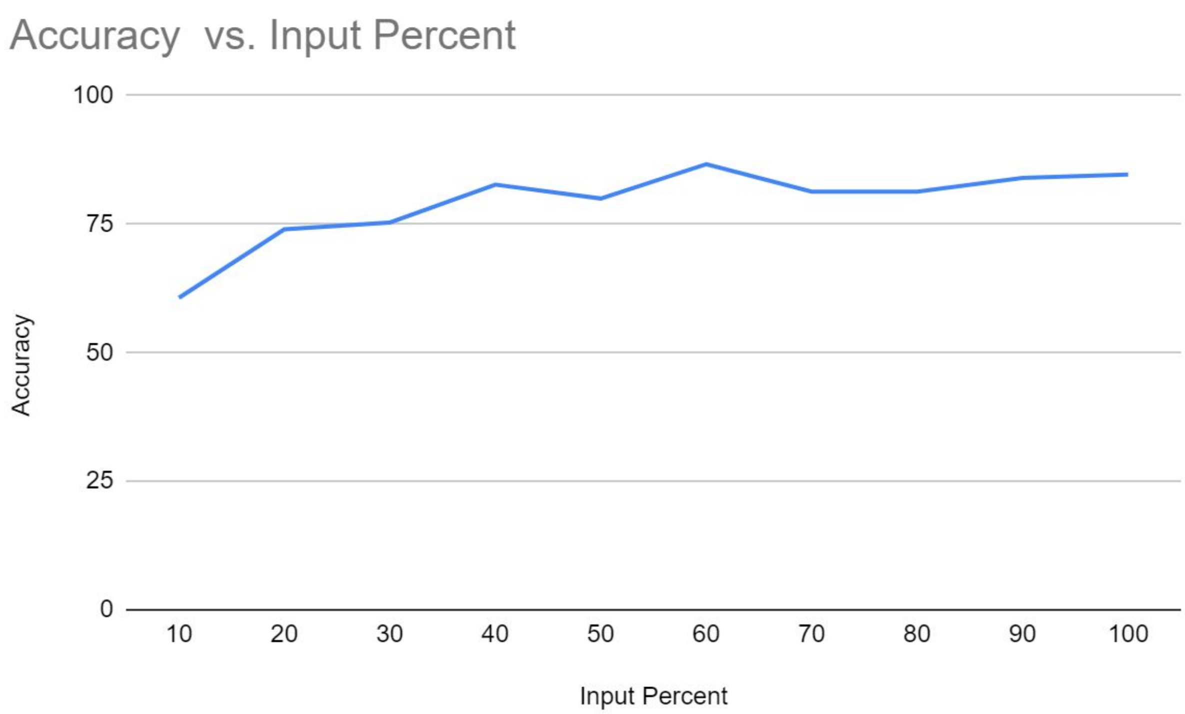
    $$fori = 1, 2, ..., j : weights[label][i] + = \phi_i(datum)$$
    $$w_0 + = 1$$
    $$fori = 1, 2, ..., j : weights[guess][i] - = \phi_i(datum)$$
    $$w_0 - = 1$$

  - Classify - For classifying, we calculate a list of numbers corresponding to all the legal labels using the following equation,

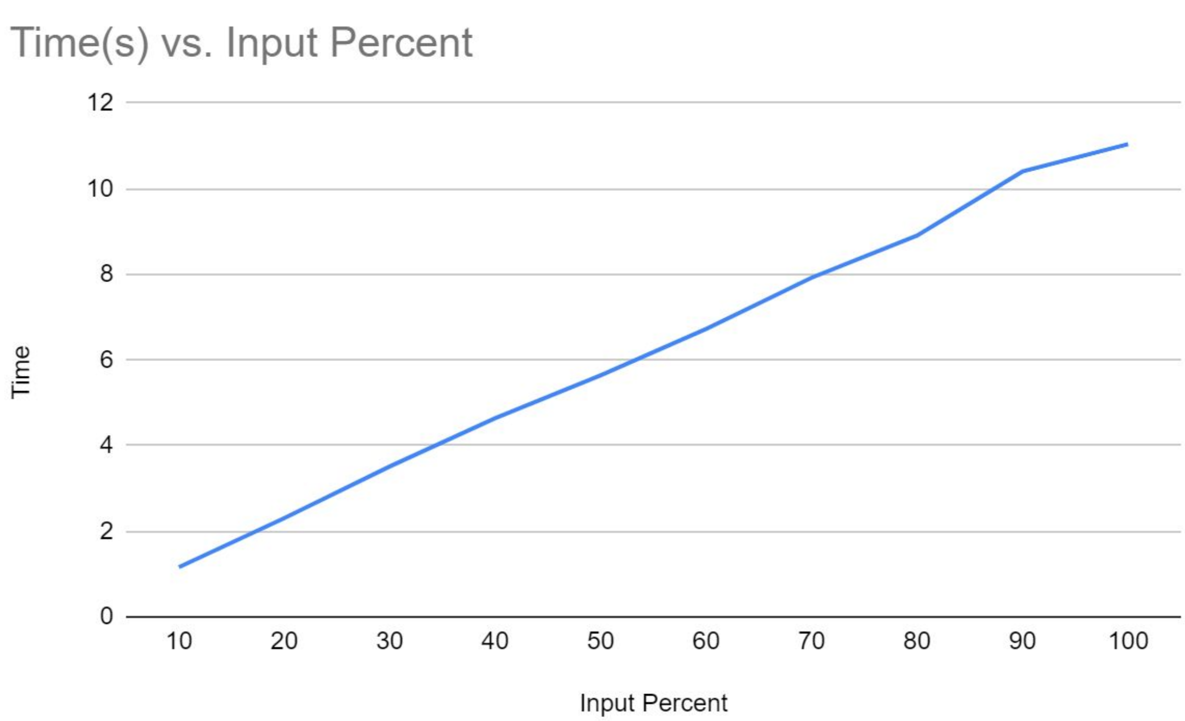
    $$f(x_i, w) = w_0 + w_1\phi_1 + ... + w_j\phi_j$$

    We pick the label with the highest f(x,w) as our guess.
  - The following are the results for both the digits and faces on 10%, 20%, 30% … 100% of the data using the Perceptron algorithm.

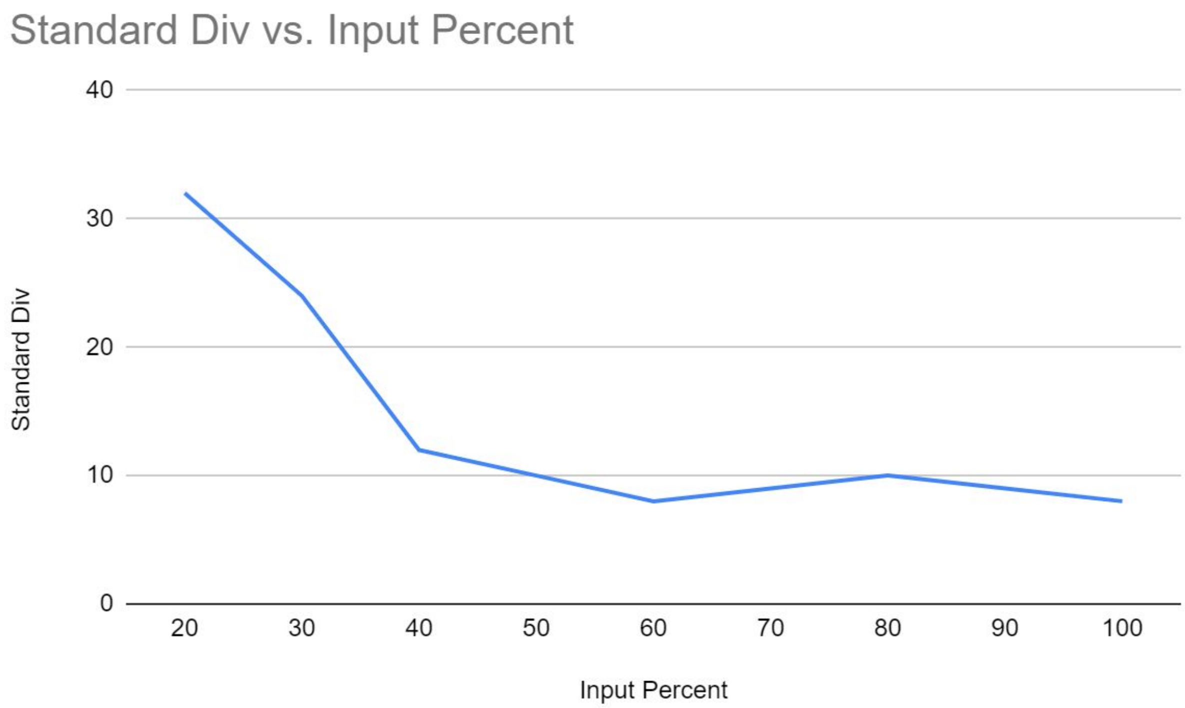
Perceptron Faces Accuracy



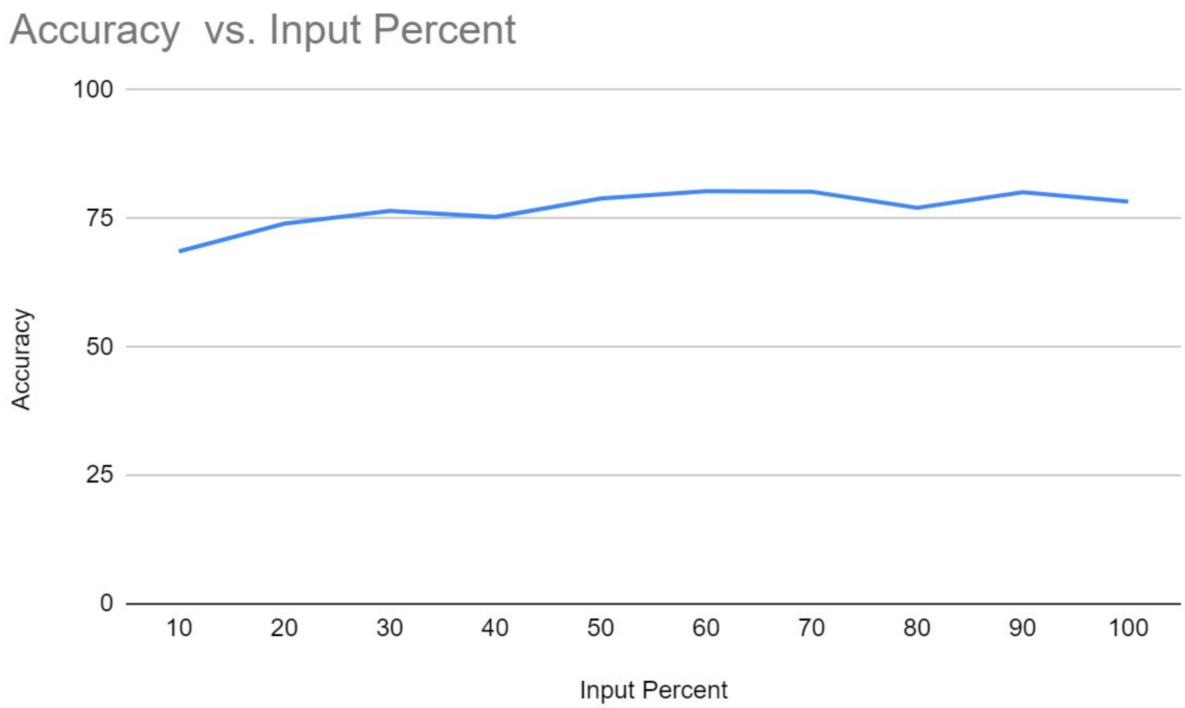
Perceptron Faces Timing



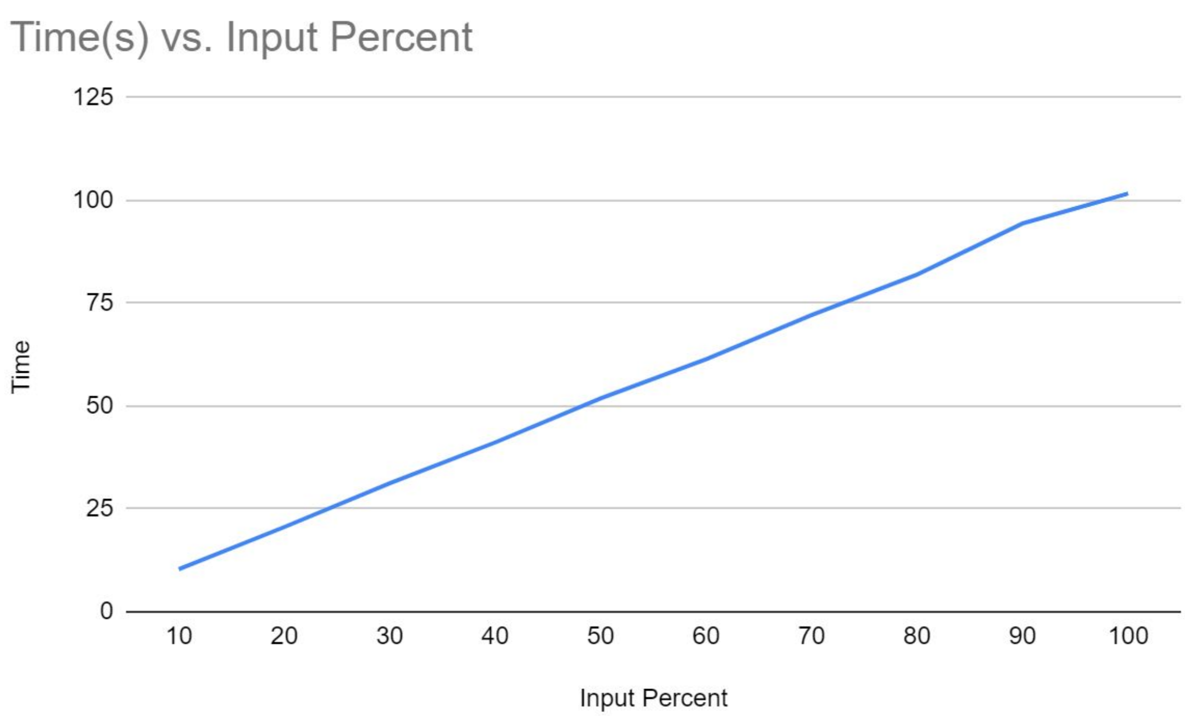
Perceptron Faces Standard Div



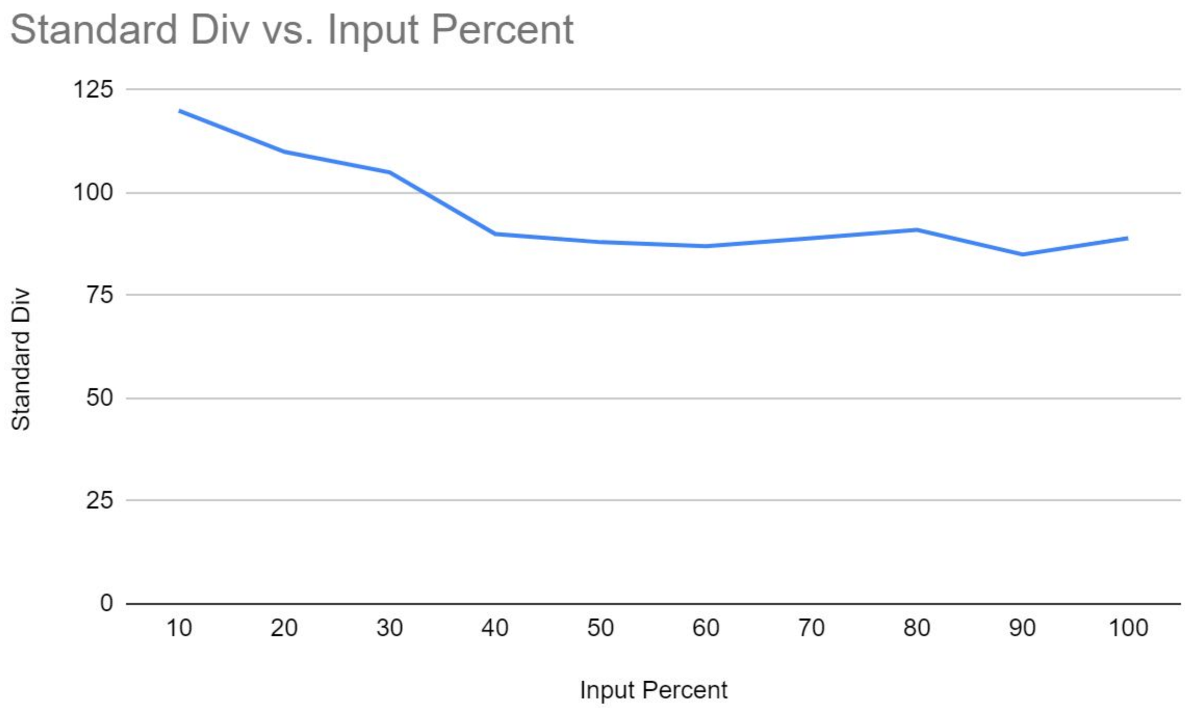
Perceptron Digits Accuracy



Perceptron Digits Timing



Perceptron Digits Standard Div

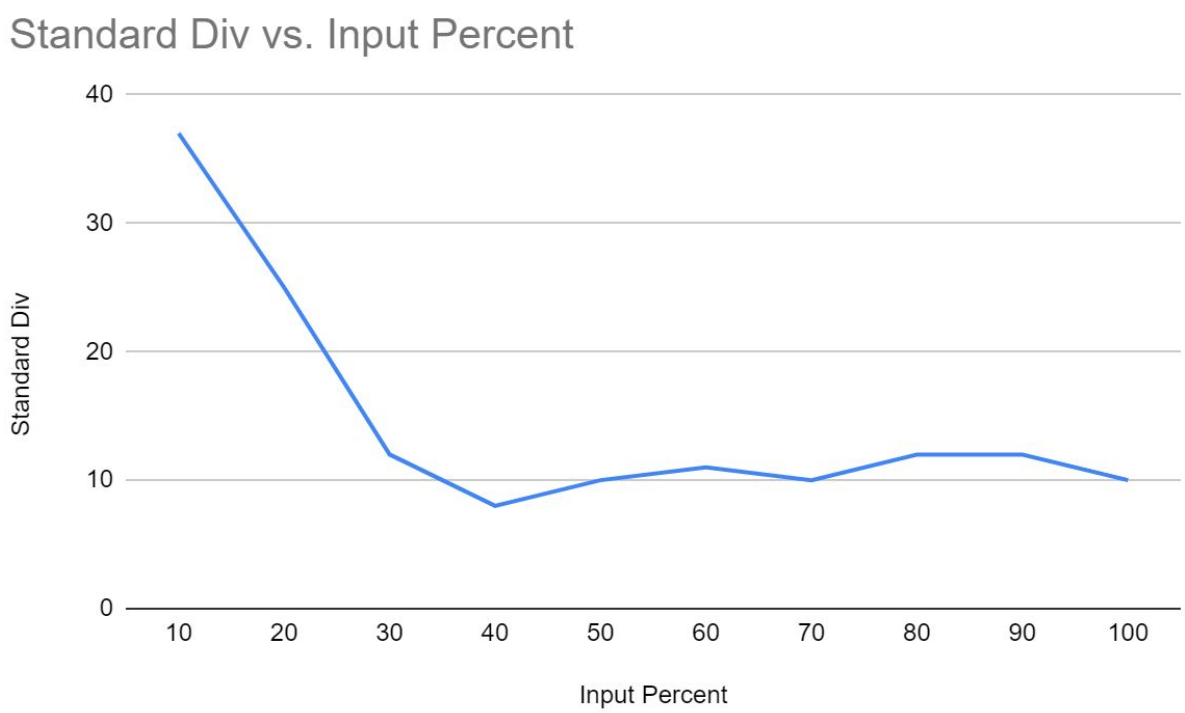


- Neural Network (Write about how it works and shit)

Neural Network Faces Accuracy
(ADD GRAPH)

Neural Network Faces Timing
(ADD GRAPH)

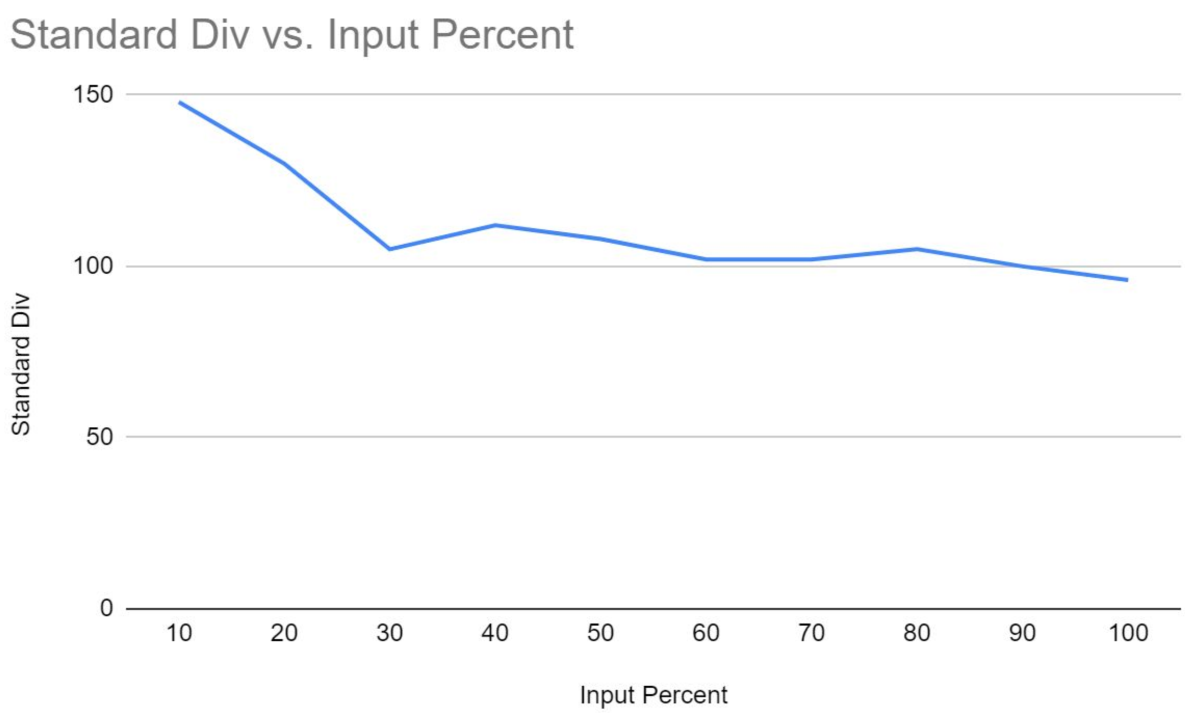
Neural Network Faces Standard Div



Neural Network Digits Accuracy
(ADD GRAPH)

Neural Network Digits Timing
(ADD GRAPH)

Neural Network Digits Standard Div



Part 3 : Discussion

- We got accuracy of over 75% for all the algorithms we performed with enough amount of training provided.
- The time spent in training was directly proportional to the amount of training data. The more the data is, the more it takes to finish the training. Perceptron algorithm took the longest in terms of timings.
- The accuracy was not proportional to the data the number of iterations in some cases. It usually started with a low accuracy for all the algorithms, goes up in the middle, converges and gets some ups and downs. There were times where the accuracy went down after using the entire data as well.
- The standard deviation was inversely proportional to the amount of total training data that is used. For example if we use about 50% of the data, then it will converge to a certain point and will not change much after that even if more data is provided as it is overfitting. There were still some instances where it changed but the instances were rare.

Part 4 : Lessons Learned

- Some of the lessons we learned is that the amount of training data does not necessarily increase the amount of accuracy. It is the quality of the training data that matters, and we don't know how good the data is until we test it. Also, we the amount of time spent in training a model also does not guarantee higher accuracy. Therefore, we do not need to increase the number of iterations and waste time for no reason.

Work Cited

Code Skeleton, Image Input, and features
http://inst.eecs.berkeley.edu/~cs188/sp11/projects/classification/classification.html

Perceptron Algorithm
https://www.youtube.com/watch?v=R2XgpDQro9k

Naive Bayes Algorithm
https://www.youtube.com/watch?v=YVwT--QiZGA