Machine Learning regarding Naive Bayes and Perceptron

Zain Ul-Abdin

August 11, 2020

1 Introduction

In this report we will examine the code and results of Naive Bayes and Perceptron as classifiers for hand-written single digits and human faces.

2 Naive Bayes

Naive Bayes utilizes probabilistic reasoning to calculate the probabilities of each feature for each class given training data, and uses this information to make a prediction in test data. Given that this relies heavily on Bayes Rule, this means that this algorithm assumes that *every* feature is independent of each other, hence being Naive.

2.1 Implementation

The algorithm was implemented in Python 3.6 as of the following specifications:

- Feature Extraction: Features are extracted by concatenating each image into one long string, with each char being a feature.
- Training: In order to train, Naive Bayes needs to use the training data to calculate the probability of each feature for each label. We must first find the probability of each label occurring in the training data, simply by counting the occurrence of each label and dividing it by the total test images. The algorithm then finds the probability of each feature occurring in a given label by using the occurrences of that feature in all instances of each label and the total instances of that label, adding a constant integer to both values to avoid a probability of 0 and to incorporate Laplace Smoothing.
- Predictions: Predictions are made by using the following formula which takes the maximum probability for all labels:

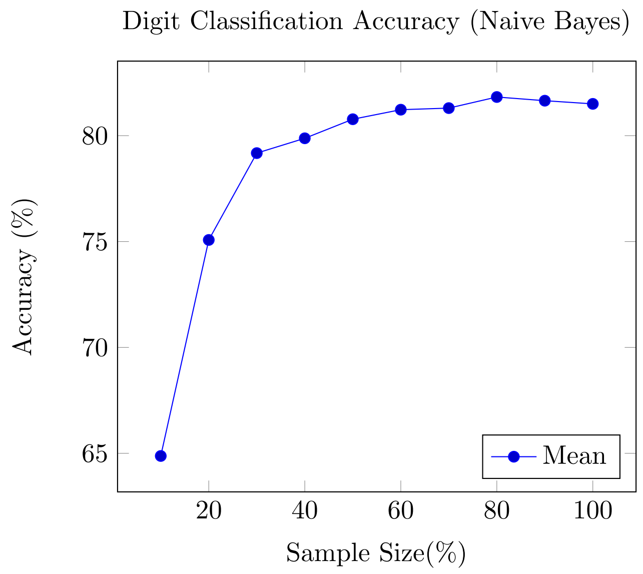
  $$\max_y \left(\log P(y) + \sum_i \log P(f_i \mid y) \right)$$

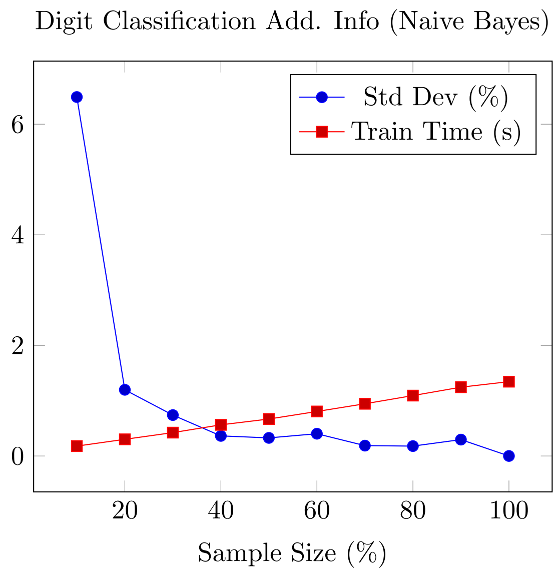
  Note: logarithms are used because multiplying by many probabilities may result in underflow.

2.2 Results

2.2.1 Digit Classification

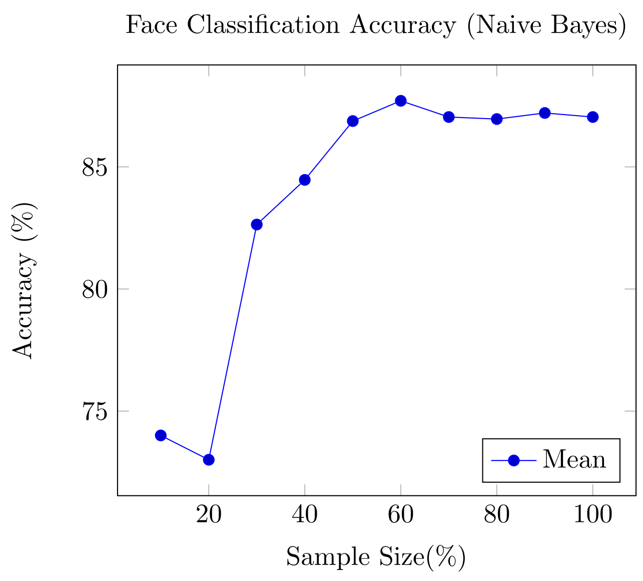
Sample Percent(%)	Mean Accuracy	Standard Deviation	Train Time (s)
10	64.875	6.4913	0.1785
20	75.075	1.1966	0.3012
30	79.175	0.7395	0.4224
40	79.875	0.3631	0.563
50	80.775	0.3269	0.6682
60	81.225	0.4023	0.8047
70	81.3	0.1871	0.9439
80	81.824	0.1785	1.092
90	81.65	0.2958	1.2439
100	81.5	0	1.3438

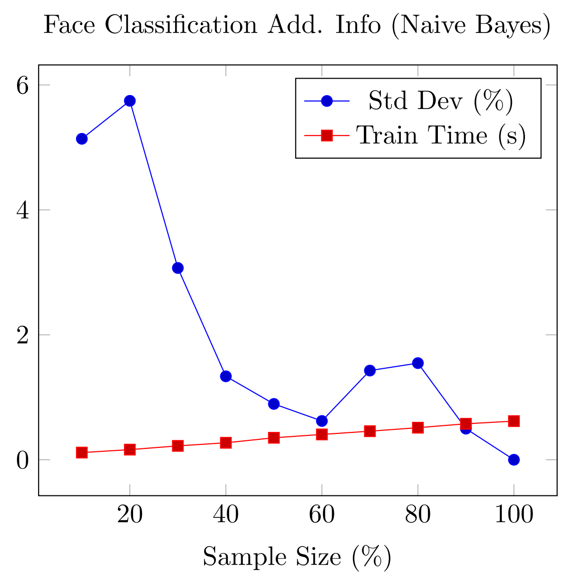




2.2.2 Face Classification

Sample Percent(%)	Mean Accuracy	Standard Deviation	Train Time (s)
10	74.003	5.138	0.1152
20	73.006	5.746	0.1625
30	82.64	3.07	0.2224
40	84.468	1.336	0.2718
50	86.877	0.894	0.352
60	87.707	0.621	0.4056
70	87.043	1.428	0.4575
80	86.96	1.547	0.5136
90	87.209	0.498	0.576
100	87.043	0	0.6189





2.3 Result Analysis

The mean accuracy is a curve that continues to increase, however plateaus at a certain point. The standard deviation also was a curve that continued to decrease, yet eventually plateaued. This means that as more training data is given, the algorithm becomes more determined in its guesses. The training time increased at about a linear rate.

3 Perceptron

Perceptron can be defined as a single-layer neural network, which models all modern deep neural nets. The algorithm attempts to learn a linear decision function f comprising of enumerated features and weights assigned to each of them, for any label. This function, for the training example set $X_{train} = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, is defined as:

$$f(x_i, w) = w_0 + w_1\phi_1(x_i) + w_2\phi_2(x_i) + w_3\phi_3(x_i) + ... + w_l\phi_l(x_i)$$

This learning process, therefore, mainly consists of analyzing one example at a time and adjusting weight vectors accordingly.

3.1 Implementation

The algorithm was implemented in Python 3.6 as of the following specifications:

- Feature Extraction: Features are extracted using a list on integers, with each char in the image being enumerated into an integer according to user-given specifications.
- Training: In order to train, Perceptron needs an initial value for the weight vectors. These are initialized to a random number between -1 and 1. It then iterates through all training images in a random order, up to some ceiling or until no weights are updated. Within these iterations, for each label, perceptron will calculate a score using the following equation:

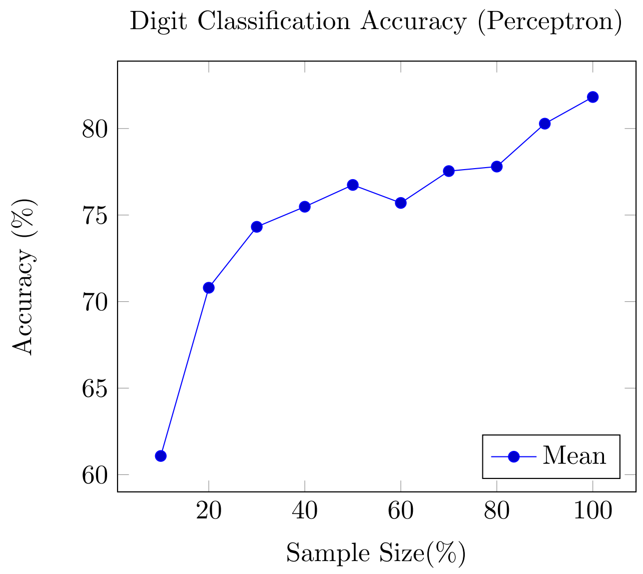
  $$score(f, y) = \sum_i f_i w_i^y$$

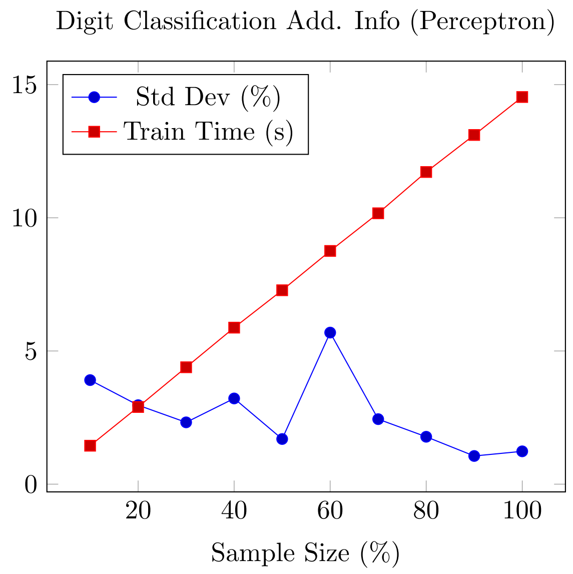
  It will then make a prediction,y_{pred} , based on the label with the highest score. If the prediction is correct and equals the real label, y_{real}, this means that the weights are doing their job correctly, and the algorithm does nothing. However, if the the prediction is wrong, this means that $w^{y_{pred}}$ is too large, and should be decremented. As such the algorithm does $w^{y_{pred}} = w^{y_{pred}} - f$. Keep in mind that both $w^{y_{pred}}$ and f are vectors represented as lists in Python. Here we have an interesting choice. We may also say that $w^{y_{real}}$ was too low. After all, if $w^{y_{real}}$ was larger, so would be $score(f, y_{real})$, and the algorithm may have chosen y_{real} as the prediction. However incrementing w_{real}^{y} may actually revert decremental changes made in previous iterations. To account for this formality, I define some threshold, ϵ, to where $w_i^{y_{real}}$ only gets decremented if $|w_i^{y_{pred}}\phi - w_i^{y_{real}}\phi| > \epsilon$, or in other words, if the change in the specific score element is large, in which case the algorithm does $w^{y_{real}} = w^{y_{real}} + f$.
- Predictions: Predictions are made in the same way that training predictions were made, by taking the *score* sum for each label, and choosing the label with the highest *score*.

3.2 Results

3.2.1 Digit Classification

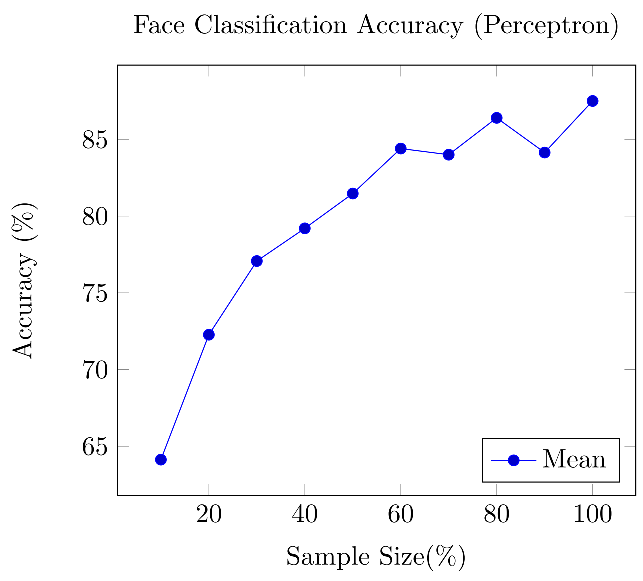
Sample Percent(%)	Mean Accuracy	Standard Deviation	Train Time (s)
10	61.08	3.9061	1.4427
20	70.8	2.9645	2.8963
30	74.32	2.3198	4.3883
40	75.48	3.2171	5.8759
50	76.74	1.6942	7.2794
60	75.7	5.6896	8.755
70	77.54	2.4402	10.1691
80	77.8	1.7788	11.7185
90	80.28	1.0572	13.1089
100	81.82	1.2319	14.5342

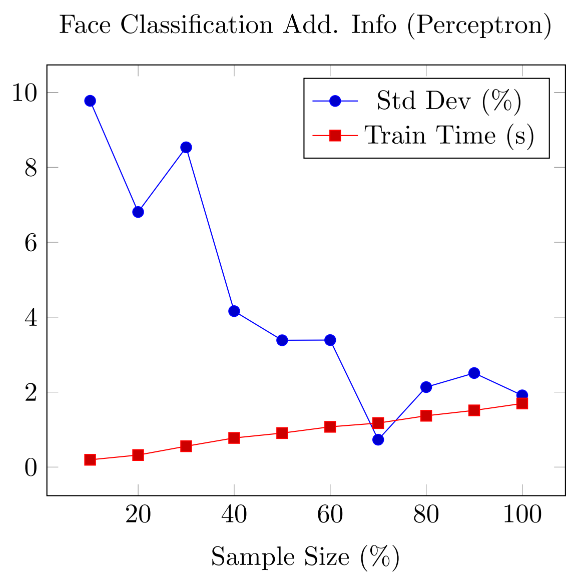




3.2.2 Face Classification

Sample Percent(%)	Mean Accuracy	Standard Deviation	Train Time (s)
10	64.13	9.7743	0.1941
20	72.27	6.8065	0.3202
30	77.07	8.5333	0.5555
40	79.2	4.1612	0.7767
50	81.47	3.3836	0.9066
60	84.4	3.3889	1.0738
70	84	0.7303	1.1739
80	86.4	2.1333	1.3689
90	84.14	2.5087	1.5122
100	87.5	1.9137	1.6958





3.3 Result Analysis

The mean accuracy is less of a curve, with a less-defined ceiling or plateau. The standard deviation seems to be quite erratic, while still following a general trend of decreasing with more training samples. The training time increased at a very linear rate.

4 Discussion of Results

Naive Bayes has a higher accuracy in the territory of very small sample sizes. However Perceptron seems to gain more benefits from very large training data sets, as the accuracy curve has a less severe plateau than that of Naive Bayes. The training time for Perceptron was much higher than that of Naive Bayes. This is may be because Bayes only needs to calculate the probabilities of each feature after going through the entire training data set once. In contrast, for Perceptron, the entire training data set can be iterated over multiple times, with each sample resulting in many sums of multiplications, which can quickly add up.

5 Closing Remarks

Both Naive Bayes and Perceptron are very powerful machine learning algorithms in regards to classification problems. While their abilities are shown in this project, there are certainly improvements which can be made. Not much time went into experimenting with the Laplace smoothing constant in Naive Bayes or the threshold constant, ϵ, in Perceptron to attempt to increase accuracy. Perhaps we can programatically find an optimal value for these constants in the training stage and use those to do a final training step.