Image Classification

Vasishta Kalinadhabhotla, Jing Yang, and Parth Mehrotra

May 6, 2018

1 Introduction

The intention of this project was to compare and contrast two classification algorithms: the Naive Bayes classifier, and the Perceptron classifier. Both algorithms were implemented primarily in Python using only standard libraries included with Python. The experiments were conducted for two different image data sets: digits and faces. Each experiment using the classifiers was tested by training 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, and 100% of the training data incrementally with a sample size of 5. The results were then averaged, and the standard deviation of the results were calculated.

2 Naive Bayes Classifier

Naive bayes classifier is based on the Bayes theorem. It works by calculating the probability for each possible label using conditional probability of all features for certain label. In the training process, we count the number of occurrences of each value, and divide it by the total number of all values. This will be used as conditional probability for this value for the feature (in this assignment a feature is a pixel). In classifying phase, we calculate the joint distribution of each label value. The label with highest probability with given features is the predicted value.
When we measure the performance of the classifier against the digits dataset, we immediately began to see a very high degree of accuracy with only 20% of the training data (figure 4.1.1). For a more complex dataset like the faces dataset we require at least 40% of the training data to start performing at

acceptable levels (figure 4.2.1). This might be due to the nature of how complex the faces data set is in comparison to the digits one.

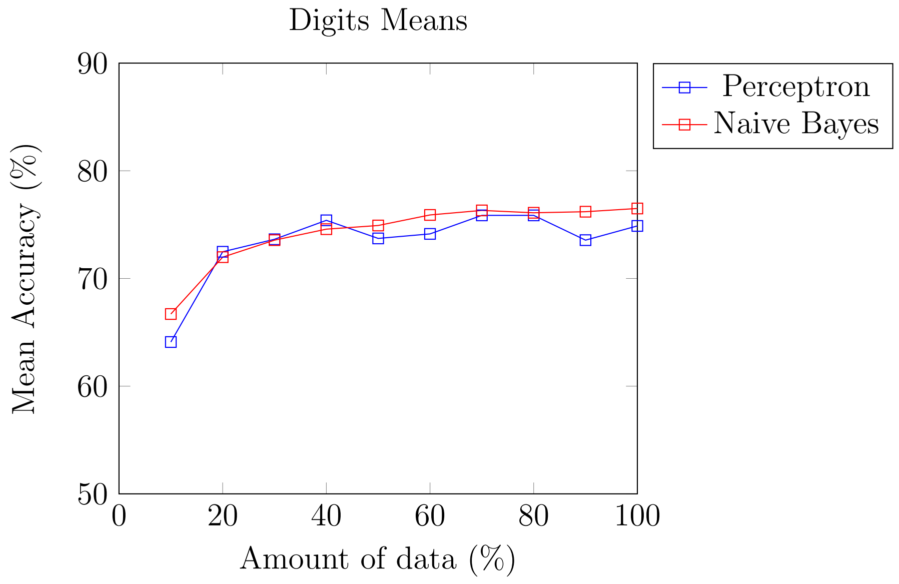
3 Perceptron Classifier

The perceptron classifier was designed as essentially a fully connected neural network with an input layer and an output layer. Each pixel signified an input where # and + characters were quantified with a value of 1.0, and whitespace characters were quantified with a value of 0.0. Every one of these inputs were connected to each perceptron, and there was one perceptron for each potential output of the classifier ([0, 1, 2, 3, 4, 5, 6, 7, 8, 9] for digit images, and [0, 1] for face images). During the training phase, when the classifier loaded an image, it quantified the pixels and fed them to the input layer. The inputs were then multiplied by the weights associated with the connections between an input node, and an output node. These values were then totaled at the output layer, and the final output of the classifier was decided to be the output node with the greatest total sum. If this output agreed with the associated label, then the classifier simply went onto the next image. If there was a discrepancy, the weights connecting all inputs to the output associated with the correct answer was updated according to the values given as input.
This design allows for the classifier to increase the weights corresponding to the inputs that coerce a correct output proportional to the value of the input. On the other hand, it also negatively impacts the connections between inputs that correspond to the incorrect output during training. One thing we have noticed is that since the final output depends on the relationship between outputs, the values that are given to the input themselves do not matter, rather it is the relationship between them that matters. Therefore it can be said that increasing training data of the same type does not benefit this classifier after a certain point, but the detail contained within the data would provide a greater benefit. Greater detail means that the relationship between the values in the pixels varies more greatly than the provided data set that only varies between 3 characters, which only translate (in this model) to two values (1.0 for # or +, and 0.0 for whitespace). This accounts for the plateau that occurs and can be seen in figures 4.1.1 and 4.2.1 below. The average accuracy for the perceptron classifier with 100

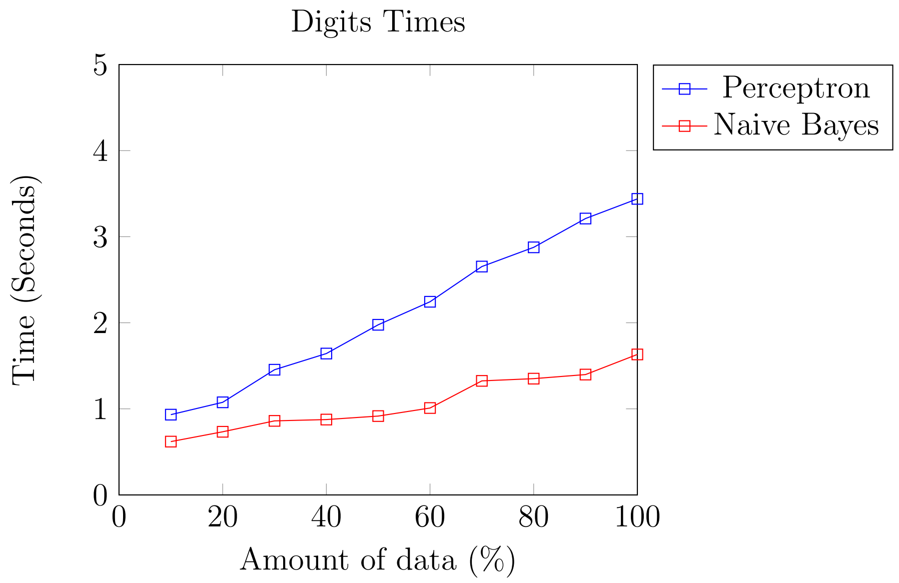
4 Plots

4.1 Digit Plots

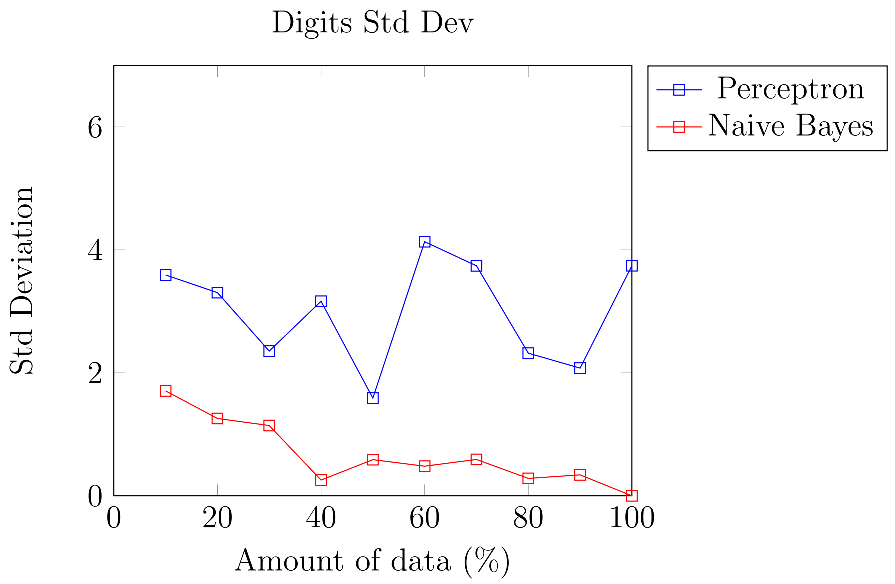
4.1.1 Digit Mean Accuracy



4.1.2 Digit Times

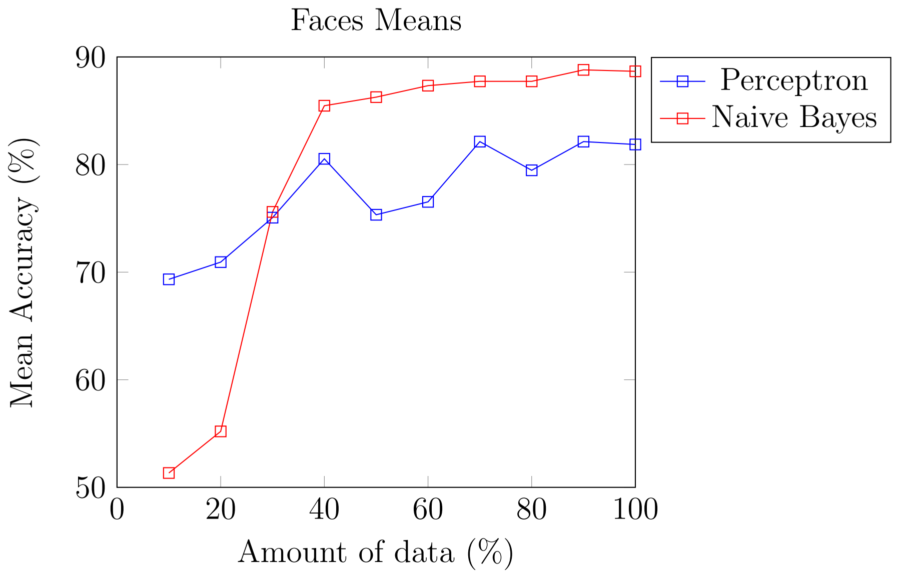


4.1.3 Digit Std Dev

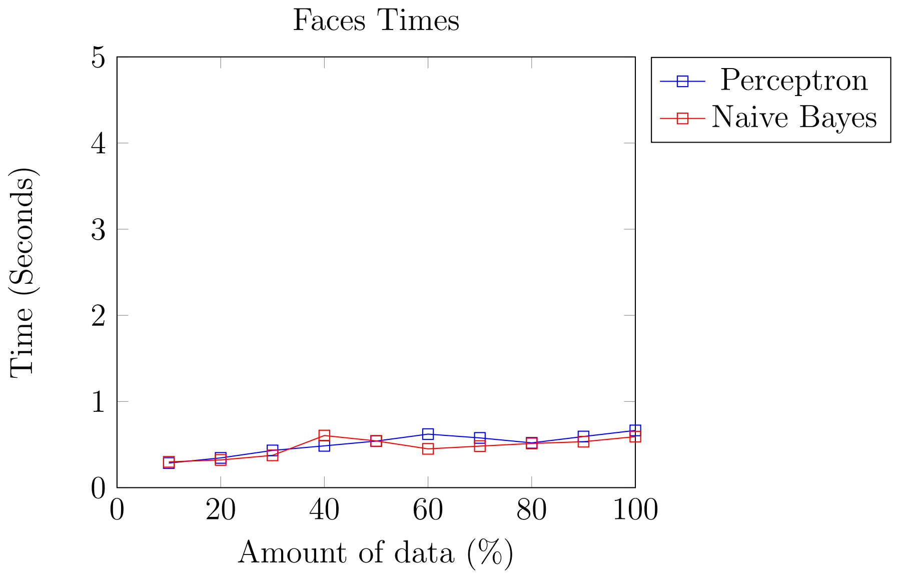


4.2 Faces

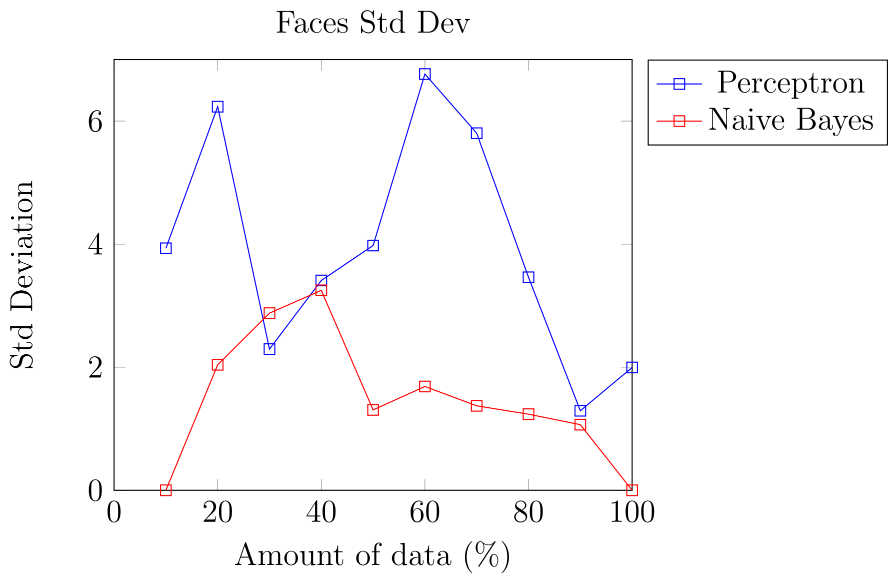
4.2.1 Faces Mean Accuracy



4.2.2 Faces Times



4.2.3 Faces Std Dev



5 Discussion and Conclusions

One of the key differences that can be seen between these two algorithms is the rate of increase in the accuracy of the classifier as more training data is added. The naive bayes classifier benefited with more training data to a higher degree than the perceptron classifier. The perceptron classifier, as explained in section 3, started to plateau and sometimes even do worse as the training data was increased to above 40% of the data available.
Another observation that can be seen is that the runtime is affected by the number of outputs that the classifier has to deal with. When classifying digits, the runtime for both classifiers was significantly higher, as there were 10 possible outputs, but was significantly faster when classifying faces, when there were only 2 possible outputs. On top of that, the classifiers had greater accuracy when there were less outputs. This is reasonable, since even if the classifier guessed the outputs at random, the probability of being correct increases as the number of choices decreases.