RUTGERS
UNIVERSITY

CS440 Project

Image Classification

Da Huo

April 29, 2018

0. Image Processing and Feature Extraction

We used the code provided from Berkeley's website for imageing processing and feature extraction.

1. Naive Bayes Classifier

1.1 Features

In this algorithm, the features we used for Naive Bayes digits classfier is just a set of pixel features. For each pixel ϕ_j, it can take either value 0, means this pixel is white, or value 1, means this pixel is black/grey. And we reture a dictionary contains the coordinate of pixel as keys, and either 1 or 0 as values.

1.2 Smoothing

In this algorithm, we use Laplace smoothing, which is to add a constant k to every possible observation value,

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

1.3 Train and Tune

To train our program, we first need to find

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

To find this probability, we keep a dictionary with key is each legal label Y and values is number of occurance, and iterate over all data set and increment each label's value by 1 every time. After we get total count for all labels, we calculate the probability $P(Y = y)$ for each label and store it in a python dictionary.

To find

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

We keep the following dictionary structure:
featureCounter={label: dict{ featureKey: {lebalFeatureValue: number of this value for this feature in training data given label y}}

Then for each label with each feature with each specific value, we increment its count by 1.

To calcualte the conditional probability, for each legal label, feature, legal feature value, we take its count, plus k, and divide by its label count plus total count of legal feature values times k since we want to take $+k$ out of the sum.

1.4 Classify

To classify a datum, we compute the log probability of $P(y|f_1, ..., f_n)$ since the probability we get will be a too small number, so we take the log probability to make it easier to compare.

We compute the log probability for each one of the legal label following the equation below

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

Then the label with maximum probabiltiy will be our guess.

1.5 Experiment

1.5.1 Digits Classifying

We tested our program on 10%,20%,30%,40%,50%, 60%,70%,80%,90%,100% of the total training data size, we get the accuracy and standard deviation on 100% of the total test data size. We generated the following graphs (figure 1-3) for time for training, accuracy, and standard deviation.

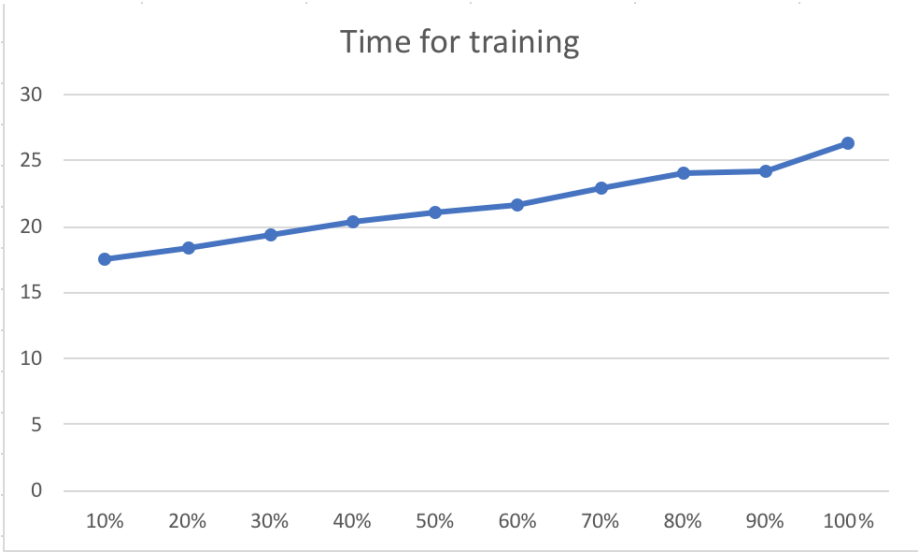

Figure 1: time spent for training in naiveBayes digits classfier

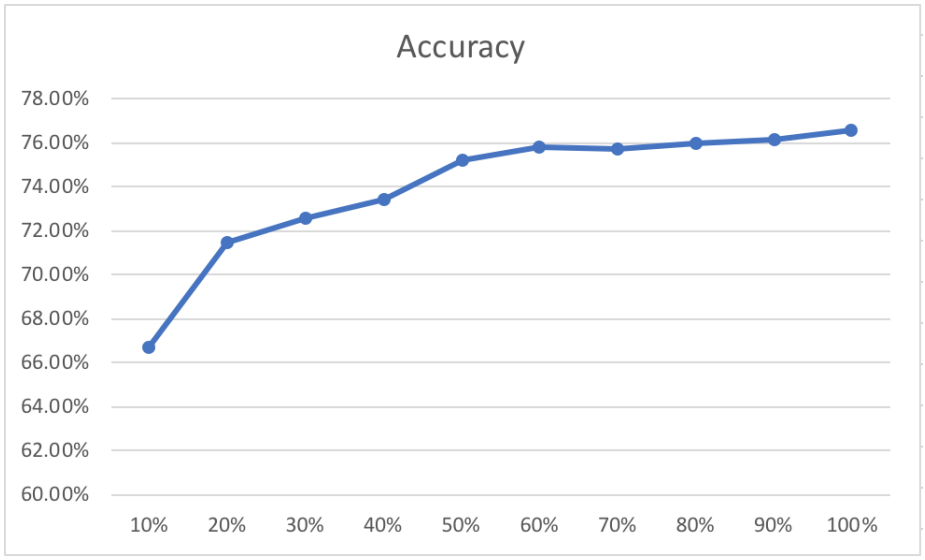

Figure 2: accuracy of classification in naiveBayes digits classfier

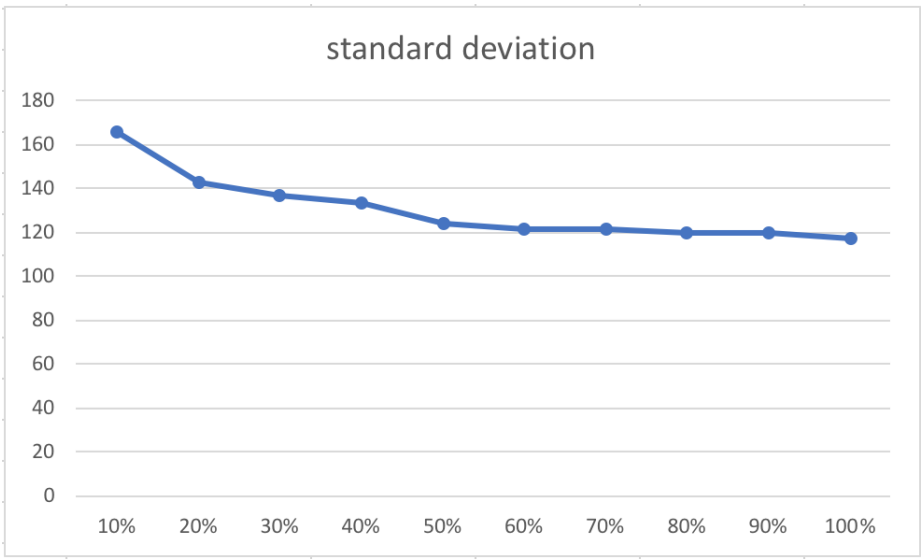

Figure 3: standard deviation of classification in naiveBayes digits classfier

The detailed information about each experiment is stored in results.txt file under project directory.

1.5.2 Faces Classifying

We tested our program on 10%,20%,30%,40%,50%, 60%,70%,80%,90%,100% of the total training data size, we get the accuracy and standard deviation on 100% of the total test data size. We generated the following graphs (figure 4-6) for time for training, accuracy, and standard deviation.

The detailed information about each experiment is stored in results.txt file under project directory.

2. Perceptron Classfier

2.1 Features

In this algorithm, the features we used for Naive Bayes digits classfier is just a set of pixel features. For each pixel ϕ_j, it can take either value 0, means this pixel is white, or value 1, means this pixel is black/grey. And we reture a dictionary contains the coordinate of pixel as keys, and either 1 or

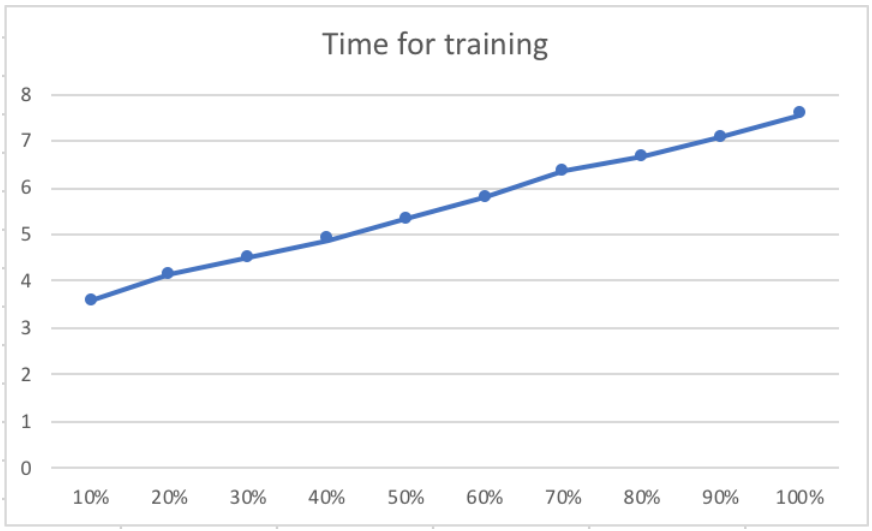

Figure 4: time spent for training in naiveBayes faces classfier

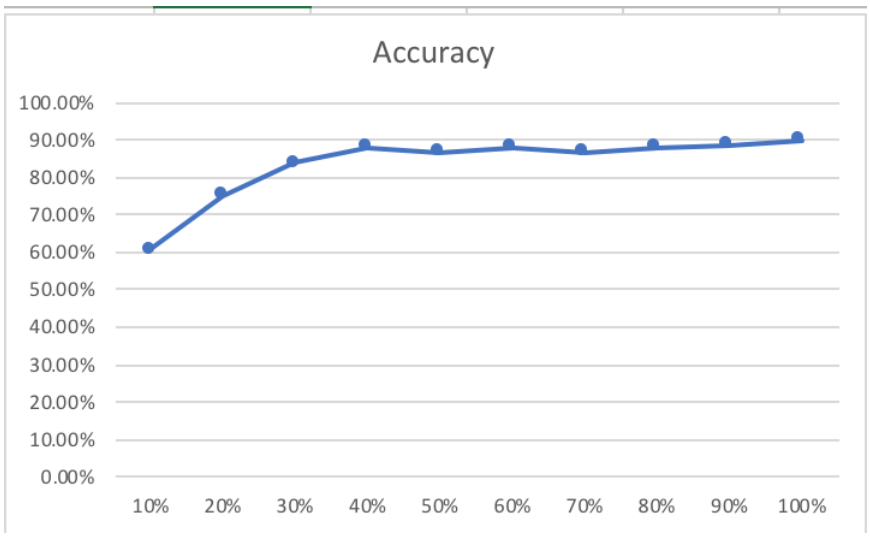

Figure 5: accuracy of classification in naiveBayes faces classfier

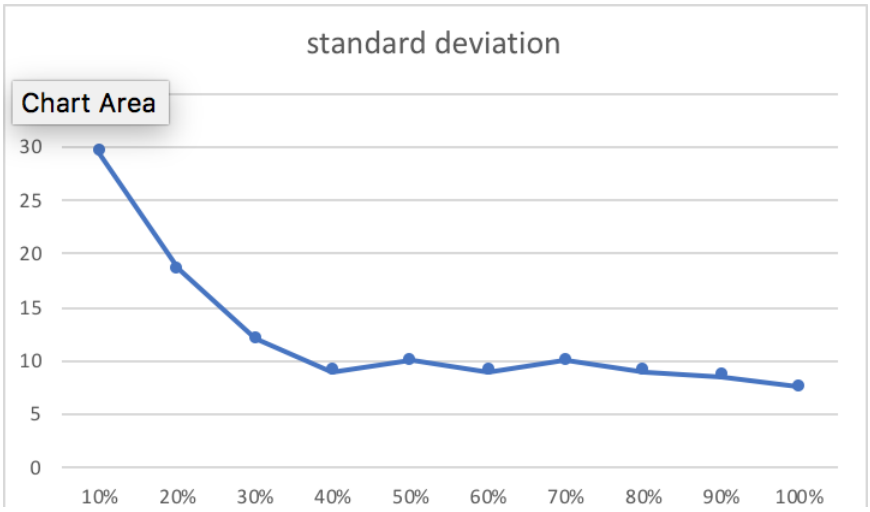

Figure 6: standard deviation of classification in naiveBayes faces classfier

0 as values.

2.2 Initializing Weights

In this algorithm, we initialize all weights to 0s at the begining and then modify them.

2.3 Training

To train our program, we first set a max iteration number. Once this number is reached, then we stop training our program.

We keep a list of weights corresponding to the features for each legal label. We use python dictionary to store these lists where the key is the label and value is list of weights.

For each iteration, we loop over all training data, for each datum, we compute a list of numbers corresponding to all legal labels following the equation:

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

we pick the label with highest $f(x_i, w)$ to be our guess. If $guess == label$, means we predicited correctly and no changes is needed. If $guess != label$, it means the weight list for correct label is too small and weight list for our

guess is too large, so we do the following modifications:

$$
\begin{aligned}
&for i = 1, 2, ..., j : weights[label][i] += \phi_i(datum) \\
&w_0 += 1 \\
&for i = 1, 2, ..., j : weights[guess][i] -= \phi_i(datum) \\
&w_0 -= 1
\end{aligned}
$$

We repeat this process until nothing is changed in one iteration or the max iteration is reached.

2.4 Classifying

To classify a given datum, we compute a list of numbers corresponding to all legal labels following the equation:

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

And we pick the label with highest $f(x_i, w)$ to be our guess.

2.5 Experiment

2.5.1 Digits Classifying

We tested our program on 10%,20%,30%,40%,50%, 60%,70%,80%,90%,100% of the total training data size, we get the accuracy and standard deviation on 100% of the total test data size. We generated the following graphs (figure 7-8) for time for training, accuracy, and standard deviation.

The detailed information about each experiment is stored in results.txt file under project directory.

2.5.2 Faces Classifying

We tested our program on 10%,20%,30%,40%,50%, 60%,70%,80%,90%,100% of the total training data size, we get the accuracy and standard deviation on 100% of the total test data size. We generated the following graphs (figure 10-12) for time for training, accuracy, and standard deviation.

The detailed information about each experiment is stored in results.txt file under project directory.

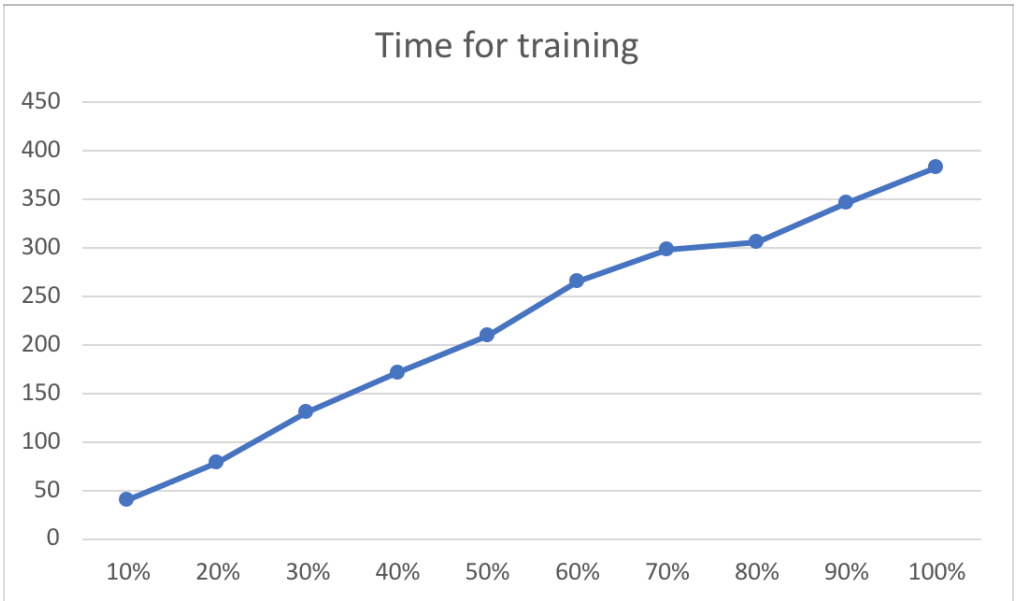

Figure 7: time spent for training in perceptron digits classfier

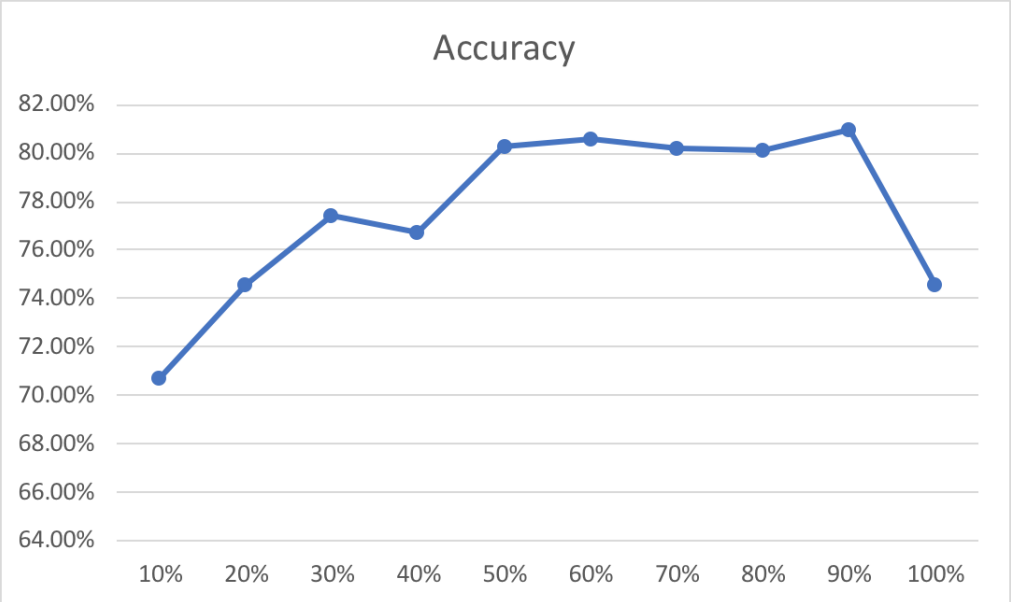

Figure 8: accuracy of classification in perceptron digits classfier

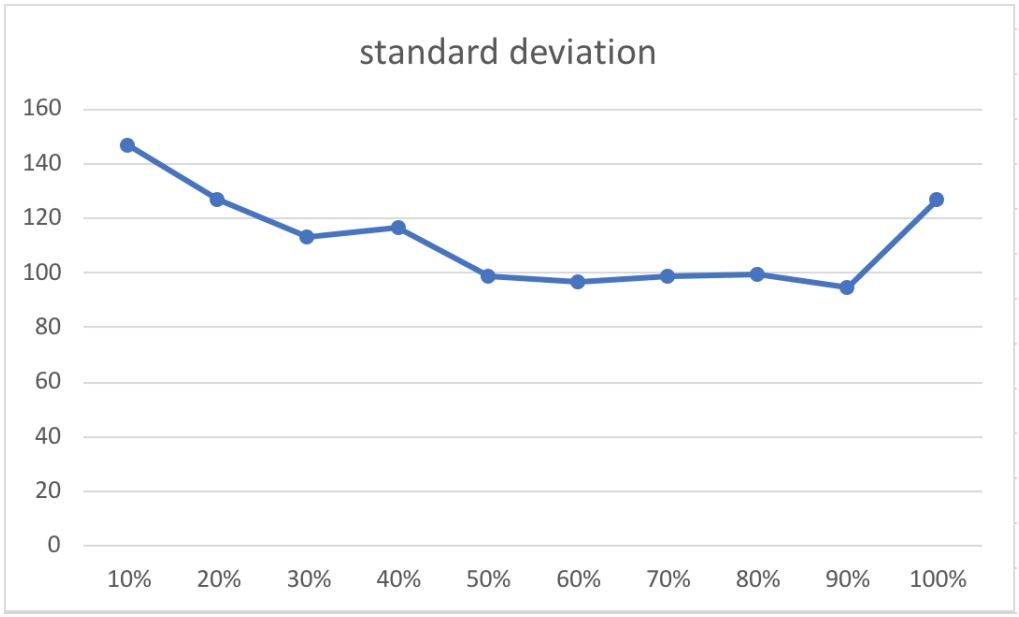

Figure 9: standard deviation of classification in perceptron digits classfier

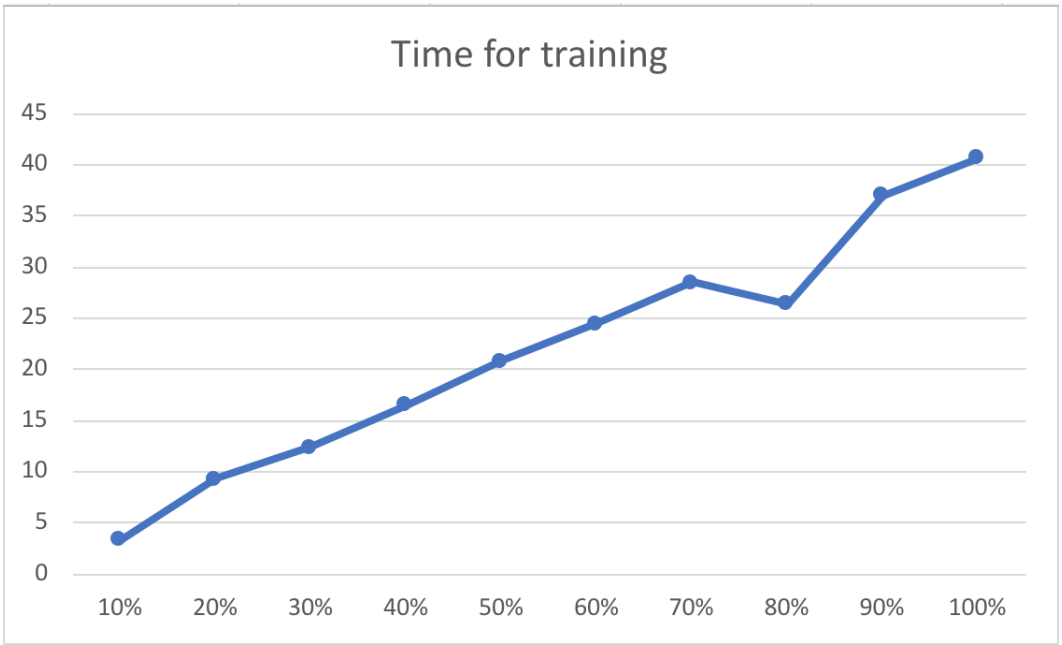

Figure 10: time spent for training in perceptron faces classfier

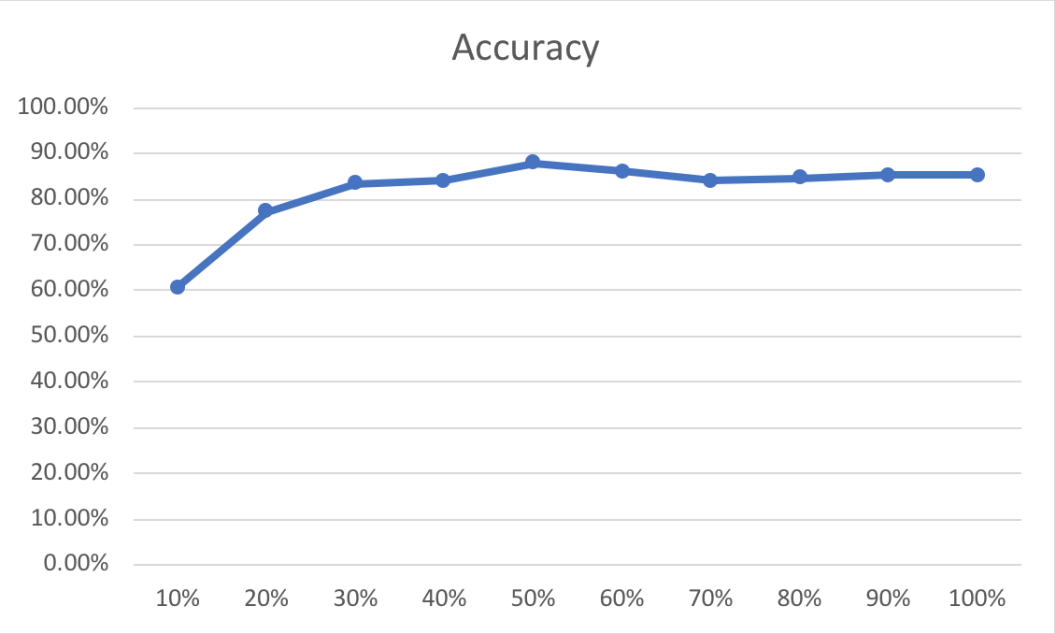

Figure 11: accuracy of classification in perceptron faces classfier

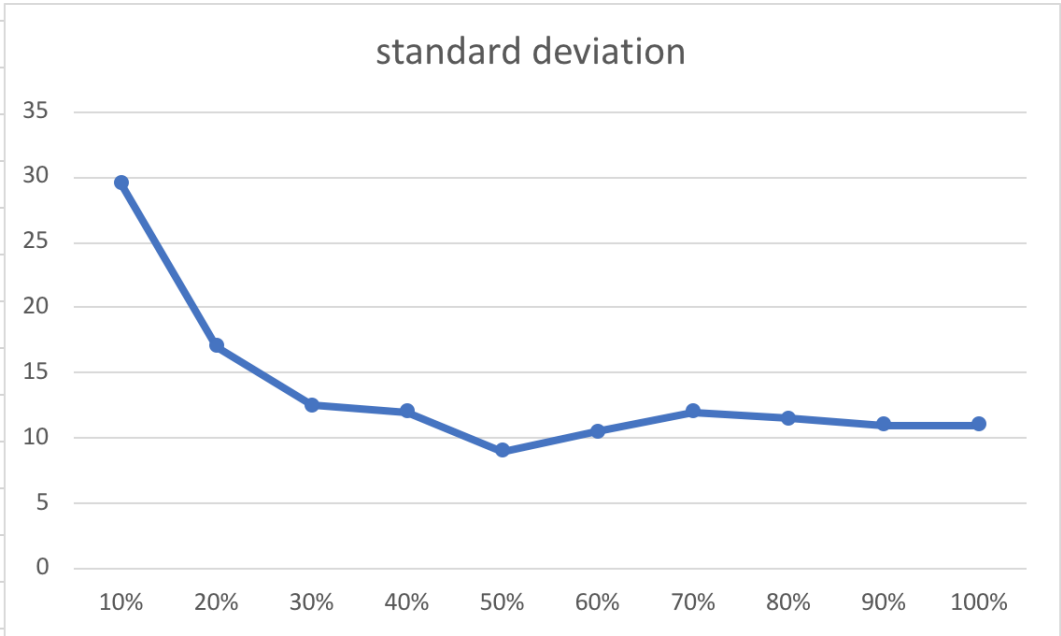

Figure 12: standard deviation of classification in perceptron faces classfier

3. Discussion

In all the experiment we performed, our program can perform with an accuracy $> 70\%$ if enought training is provided.

In terms of time spent on training process, it is proportional to the amount of traning data provided, means the more training data is provided, the longer it takes to finish the training process.

In terms of accuracy, it is proportional to the amount of training data provided at the begining, but after a certain amount of training data is provided, for example 40%, it converges to a certain accuracy and will not increase too much if more training data is provided. Sometime the accuracy may drop a bit when more training data is given because of overfitting.

In terms of standard deviation, it is inverse proportional to the amount of training data provided at the begining, but after a certain amount of training data is provided, for example 40%, it converges to a certain number and will not change too much if more training data is provided. Sometime the standard deviation may rise a bit when more training data is given because of overfitting.

4. Lessons Learned

The lesson I learned is that more training data does not nessesarily means a higher accuracy. The accuracy will converge to a certain amount depending on the featuer chosen and algorithm used. So when it comes to train our program, we need to do some experiment to find the amount of training data we have to use in order to reach the converging point. So we don't waste too much time on training process with the same accuracy.