CS440 Project 2 Report

Douglas Gromek, Michael Comatas, Xiaoyu Sun

August 10, 2020

Naive Bayes

Naive Bayes Rule:

$$P(C|D) = \frac{P(D|C)P(C)}{P(D)}$$

$$P(D) = \sum_{c} P(D, C)$$

We know that $P(D)$ is a constant for all classes, so we could consider the equation above as:

$$P(C|D) \propto P(D|C)P(C)$$

where C stands for Class, D stands for Data. Thus, we only need to compute $P(D|C)$ and $P(C)$ in the implementation.

Digit - Implementation

Training We first initialize 10 of 2D arrays with 0. Each array references as our models for 10 digits. We know each image consists of symbols, by reading the training images, we save the image in the 2D array we have already. For example:

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Table 1: Initial model for a digit

Say we have two train images labeled 1, and the two images are saved below, as we mentioned, we would read the image, and if there is a symbol in a cell, we would save it as a number.

0	1	1	0
0	1	1	0
0	1	1	0
0	1	1	0

Table 2: training image 1 with label 1

0	1	1	0
0	1	1	0
0	1	1	0
0	1	1	0

Table 3: training image 2 with label 1

After feed the model for digit 1 with the 2 training images above, the model for digit 1 would looks like this:

0	2	2	0
0	2	2	0
0	2	2	0
0	2	2	0

Table 4: training image with label 1

After more and more images are feed to the model, we will have the model looks like this:

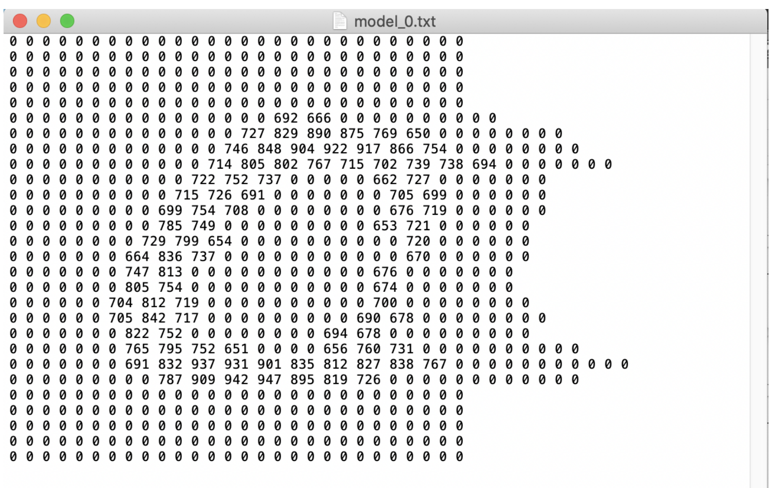
model_0.txt
```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 692 666 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 727 829 890 875 769 650 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 746 848 904 922 917 866 754 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 714 805 802 767 715 702 739 738 694 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 722 752 737 0 0 0 0 0 662 727 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 715 726 691 0 0 0 0 0 0 0 705 699 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 699 754 708 0 0 0 0 0 0 0 0 676 719 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 785 749 0 0 0 0 0 0 0 0 0 653 721 0 0 0 0 0 0
0 0 0 0 0 0 0 0 729 799 654 0 0 0 0 0 0 0 0 0 0 720 0 0 0 0 0 0
0 0 0 0 0 0 0 664 836 737 0 0 0 0 0 0 0 0 0 0 0 670 0 0 0 0 0 0
0 0 0 0 0 0 0 747 813 0 0 0 0 0 0 0 0 0 0 0 676 0 0 0 0 0 0 0
0 0 0 0 0 0 0 805 754 0 0 0 0 0 0 0 0 0 0 0 674 0 0 0 0 0 0 0
0 0 0 0 0 0 704 812 719 0 0 0 0 0 0 0 0 0 0 700 0 0 0 0 0 0 0 0
0 0 0 0 0 0 705 842 717 0 0 0 0 0 0 0 0 0 690 678 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 822 752 0 0 0 0 0 0 0 0 694 678 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 765 795 752 651 0 0 0 0 656 760 731 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 691 832 937 931 901 835 812 827 838 767 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 787 909 942 947 895 819 726 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 1: model for digit 1 after 100% training data

We also filter out all the numbers that smaller than 650, such that the numbers left represent the densest area that digit 1 has appeared.

Prediction In order to make a prediction on one image, we test the image with all 10 models, then we pick the largest $P(D|C)$ among all the possibilities as the prediction.

Steps:

In order to compute $P(C)$:

$$P(C = x) = \frac{how\ many\ digit\ x\ in\ the\ data\ pool}{the\ total\ amount\ of\ digits}$$

such that we can get the frequency of the digit x.

Then we need to calculate $P(D|C)$:

$$P(D|C) = \Pi_{i=1}^{m} P(f_i|C)$$

For example: for $P(D|C = 1)$, we loop through both the model for 1 and the test image, compare the common area between them,

$$P(f_n|C = 1) = \frac{common\ cells\ in\ row\ n}{total\ cells\ in\ row\ n}$$

Digit - Statistic

BAYES DIGIT				
Training Data%	Time (seconds)	Correct	Incorrect	Correct rate
10%	0.8687984943	108	892	0.108
20%	0.9240980148	108	892	0.108
30%	1.370334625	192	808	0.192
40%	1.936785936	384	616	0.384
50%	2.643389225	556	444	0.556
60%	2.898879766	586	414	0.586
70%	3.197668076	632	368	0.632
80%	4.594192743	647	353	0.647
90%	4.093582153	655	345	0.655
100%	4.685601711	640	360	0.64

Figure 2: Testing Data Collection for Digit

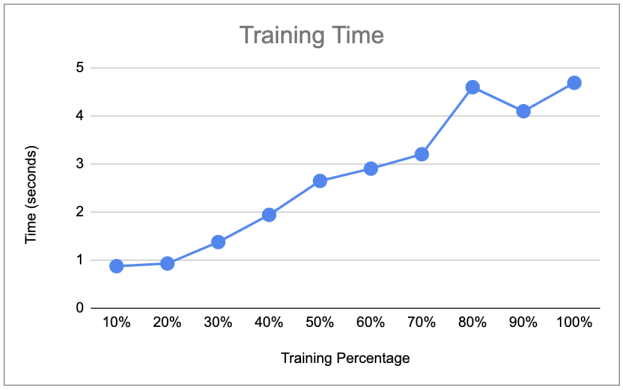

Figure 3: Training Time vs. Training Data Percentage

From figure 3 we can tell that the more data we use, the longer the time it takes.

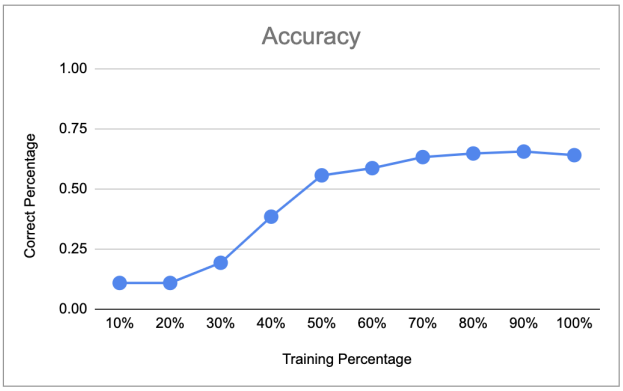

Figure 4: Correct percentage vs. Training Data Percentage

From figure 4 we can tell that the more training data we use, the prediction we made becomes more accurate. However, when the amount of data up to a certain number, the growth pace of the accuracy slows down. It might because of the limitation of our algorithm design, the maximum of the accuracy the feature can make is around 65%, such that the accuracy stops changing significantly.

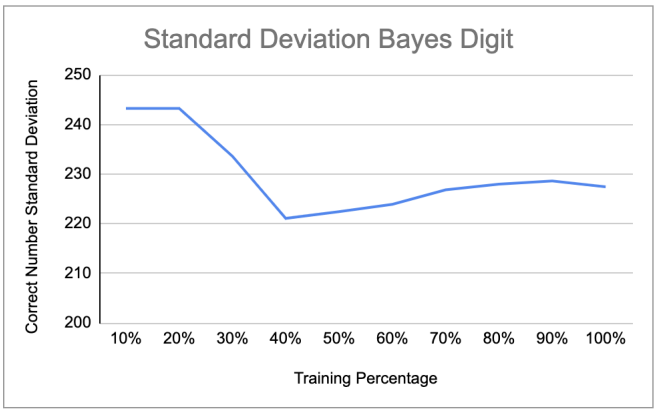

Figure 5: Standard Deviation for Bayes digit

From figure 5 we noticed the standard deviation tends to be inverse proportional to the amount of training data provided. After a certain amount of training data provided the standard deviation began to converge, for example around 40% or 50%. With less training data provided the standard deviation tended to be larger because there is a larger difference in accuracy. Then as the accuracy begins to be more precise the standard deviation converges. When the standard deviation would spike it was most likely because of overfitting, basically when the model "memorized" the figure rather than learning and "generalizing" to it.

Face - Implementation

Training First we initialize 4 lists:

- 1. Hold the max row length for every non-face image
- 2. Hold the max row length for every face image
- 3. Hold the max column length for every non-face image
- 4. Hold the max column length for every face image

We loop through all the training images, then populate above four lists. Then save it locally as a .txt file.

Prediction In order to make a prediction on one image, we get the max row and max column length values for that image. We then we pick the largest $P(D|C)$ as the prediction.

Steps: Take face image as an example

In order to compute $P(C)$:

$$P(C_{face}) = \frac{total\ amount\ of\ faces}{the\ total\ amount\ of\ images}$$

such that we can get the frequency of faces appear.

Then we need to calculate $P(D|C)$:

$$P(D|C) = \Pi_{i=1}^{m} P(f_i|C)$$

In this case, $P(D|C_{face}) = P(f_{row}|C_{face}) \times P(f_{column}|C_{face})$.

row:the times the max row length of current test image appears in the list2 above.
column:the times the max column length of current test image appears in the list4 above.

$$P(f_{row}|C_{face}) = \frac{row}{the\ length\ of\ list\ 2\ above}$$

$$P(f_{column}|C_{face}) = \frac{column}{the\ length\ of\ list\ 4\ above}$$

Face - Statistic

BAYES FACE				
Training Data%	**Time (seconds)**	**Correct**	**Incorrect**	**Correct rate**
10%	0.7915506363	78	72	0.52
20%	0.9069709778	84	66	0.56
30%	1.133635521	86	64	0.5733333333
40%	1.18377471	93	57	0.62
50%	1.296403646	101	49	0.6733333333
60%	1.459578753	111	39	0.74
70%	1.680142879	108	42	0.72
80%	1.756377459	106	44	0.7066666667
90%	1.950663567	104	46	0.6933333333
100%	2.018542051	106	44	0.7066666667

Figure 6: Testing Data Collection for Face

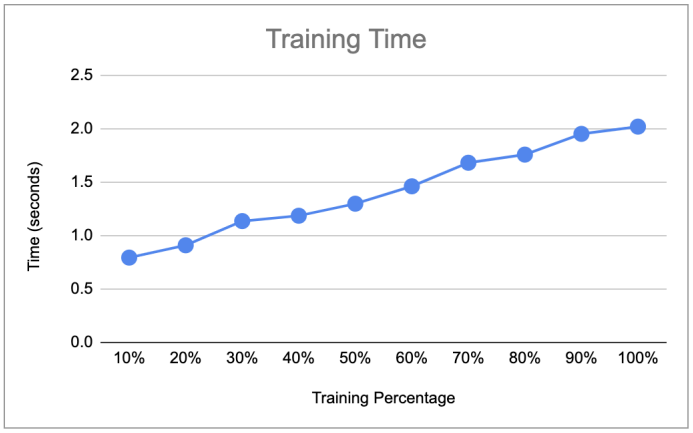

Figure 7: Training Time vs. Training Data Percentage

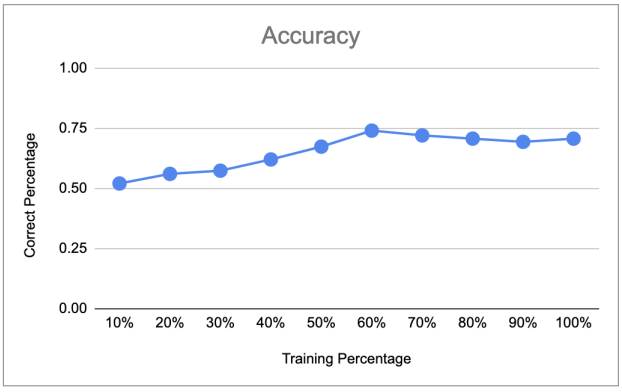

Figure 8: Correct percentage vs. Training Data Percentage

From figure 7 we can tell that the more data we use, the longer the time it takes.

From figure 8 we can tell that the more training data we use, the prediction we made becomes more accurate. However, when the amount of data up to a certain number, the growth pace of the accuracy slows down. Same as the reason for digit classification, it might because of the limitation of our algorithm design, the maximum of the accuracy the feature can make is around 70%, such that the accuracy stops changing significantly.

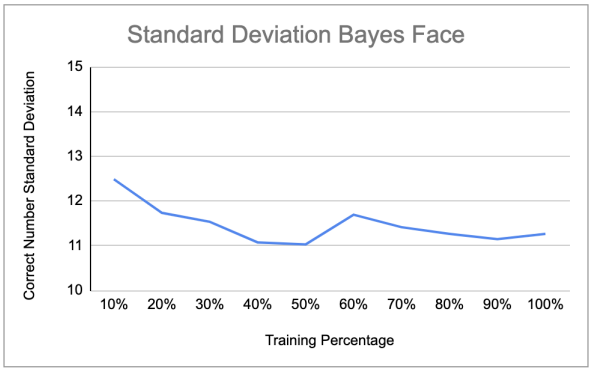

Figure 9: Standard Deviation for Bayes face

Perceptron

Digit - Implementation

Training:

1. We initialize the weight model by using random function to assign weights from -1 to 1.

2. For every training images, we loop through every row and every column of it, to get the following data:

- How many "+" appears in row x
- How many "#" appears in row x
- How many "+" appears in column y
- How many "#" appears in column y

We have a list of features that contains above data. Each of them is a feature, if the image's size is $x \times y$, such that as total, we have $2 \times x + 2 \times y$ features.

3. We use the equation below to calculate f value:

$$f(x_i, w) = w_0 + w_1\phi(x_i) + w_2\phi(x_i) + ... + +w_l\phi(x_i)$$

For example, in this case, $\phi(x_i)$ with w_2 would be feature 2 we have, which is how many "#" appears in row 0.

4. In order to adjust the weight model, we first calculate the f-value for all the 10 digits weight model, then pick the largest one as our prediction. In perceptron, the process of training is also a process of predicting. Then we compare the predict result to the actual label:

- If the prediction is right and the f-value < 0, then we adjust the all the weights of the prediction digit by:

$$w_0 = w_0 + 1$$

$$w_j = w_j + \phi(x_i)$$

- If the prediction is wrong and the f-value ≥ 0, then we adjust the all the weights of the prediction digit by:

$$w_0 = w_0 - 1$$

$$w_j = w_j - \phi(x_i)$$

  Also adjust the all the weights of the actual label by:

$$w_0 = w_0 + 1$$

$$w_j = w_j + \phi(x_i)$$

Prediction:

This part has the same process as the training, the only diffrences:

- 1. We read the weigh model file that we save locally as the model.
- 2. After we get the prediction, we don't adjust the weight model.

We loop through the weights model for the 10 digits, and calculate the f-value by the following equation:

$$f(x_i, w) = w_0 + w_1\phi(x_i) + w_2\phi(x_i) + ... + +w_l\phi(x_i)$$

Then we pick the largest one among the 10 f-values as our prediction.

Digit - Statistic

PERCEPTRON DIGIT				
Training Data%	**Time (seconds)**	**Correct**	**Incorrect**	**Correct rate**
10%	5.961833	206	794	0.206
20%	11.754816	376	624	0.376
30%	18.57084512	505	495	0.505
40%	26.2485859	382	618	0.382
50%	32.073849	464	536	0.464
60%	38.1622018	514	486	0.514
70%	43.174445	516	484	0.516
80%	50.1033573	633	367	0.633
90%	53.91777396	497	503	0.497
100%	63.27301383	614	386	0.614

Figure 10: Testing Data Collection for Digit

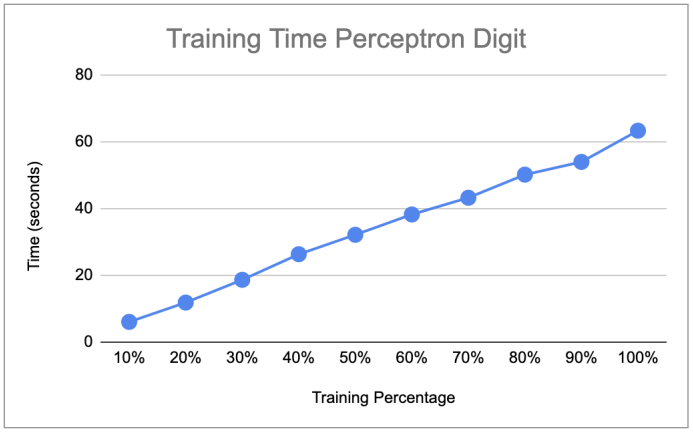

Figure 11: Training Time vs. Training Data Percentage

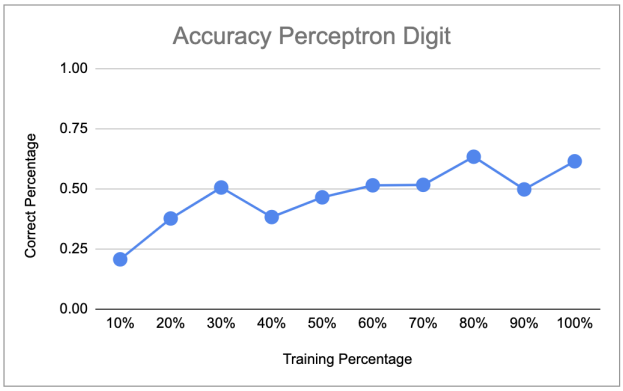

Figure 12: Correct percentage vs. Training Data Percentage

From figure 11 we can tell that the more data we use, the longer the time it takes.

From figure 12 we can tell that the more training data we use, the prediction we made becomes more accurate. However, when the amount of data up to a certain number, the growth pace of the accuracy slows down. Same as the reason before, it might because of the limitation of our algorithm design, the maximum of the accuracy the feature can make is around 60%, such that the accuracy stops changing significantly.

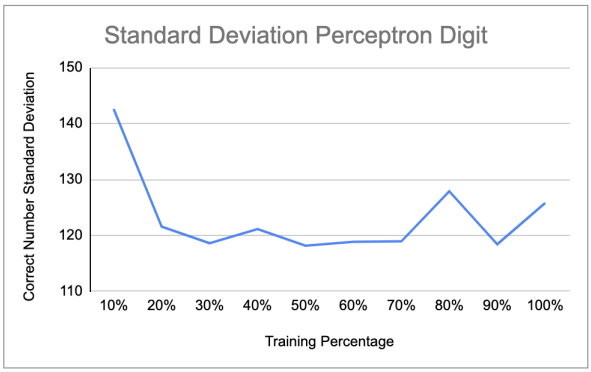

Figure 13: Standard Deviation for Perceptron digit

Face - Implementation

Training:

1. We initialize the weight model by using random function to assign weights from -1 to 1.

2. We loop through all the train images, then get the max lengths for row x among all the images, then store them somewhere. Thus the number of features we have equals the number of rows this image has.

3. We use the equation below to calculate f value:

$$f(x_i, w) = w_0 + w_1\phi(x_i) + w_2\phi(x_i) + ... + +w_l\phi(x_i)$$

For example, in this case, $\phi(x_i)$ with w_2 would be feature 2 we have, which is the max lengths for row 2 among all the images..

4. In order to adjust the weight model, we first calculate the f-value for face and non-face weight models, then pick the larger one as our prediction. In perceptron, the process of training is also a process of predicting. Then we compare the predict result to the actual label:

- If the prediction is right and the f-value < 0, then we adjust the all the weights of the prediction digit by:

$$w_0 = w_0 + 1$$

$$w_j = w_j + \phi(x_i)$$

- If the prediction is wrong and the f-value ≥ 0, then we adjust the all the weights of the prediction digit by:

$$w_0 = w_0 - 1$$

$$w_j = w_j - \phi(x_i)$$

  Also adjust the all the weights of the actual label by:

$$w_0 = w_0 + 1$$

$$w_j = w_j + \phi(x_i)$$

Prediction:

This part has the same process as the training, the only diffrences:

- 1. We read the weigh model file that we save locally as the model.
- 2. After we get the prediction, we don't adjust the weight model.

We loop through the weights model of face and non-face, and calculate the f-value by the following equation:

$$f(x_i, w) = w_0 + w_1\phi(x_i) + w_2\phi(x_i) + ... + +w_l\phi(x_i)$$

Then we pick the larger one as our prediction.

Face - Statistic

PERCEPTRON FACE				
Training Data%	Time (seconds)	Correct	Incorrect	Correct %
10%	7.316613197	77	73	0.5133333333
20%	14.36563396	77	73	0.5133333333
30%	21.35206699	91	59	0.6066666667
40%	29.38281894	73	77	0.4866666667
50%	20.95867419	73	77	0.4866666667
60%	42.02887559	71	79	0.4733333333
70%	34.41538191	70	80	0.4666666667
80%	18.15125227	99	51	0.66
90%	26.01145077	68	82	0.4533333333
100%	68.33838844	103	47	0.6866666667

Figure 14: Testing Data Collection for Face

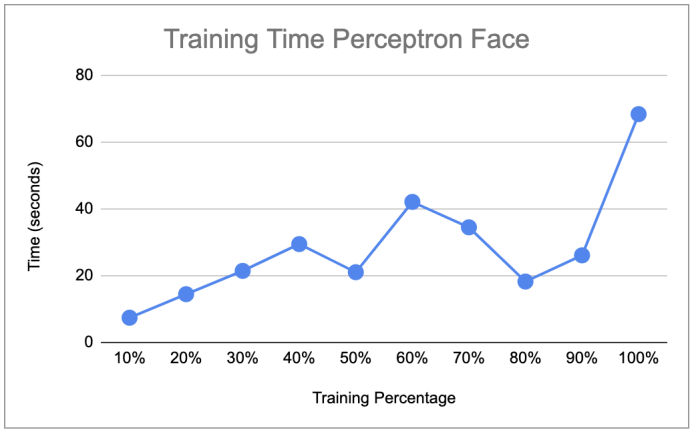

Figure 15: Training Time vs. Training Data Percentage

From figure 15 we can tell that the more data we use, the longer the time it takes.

From figure 16 we can tell that the more training data we use, the prediction we made becomes more accurate. However, from 40% to 70% the accuracy is lower than the two sides, the reason could that part of data is biased.

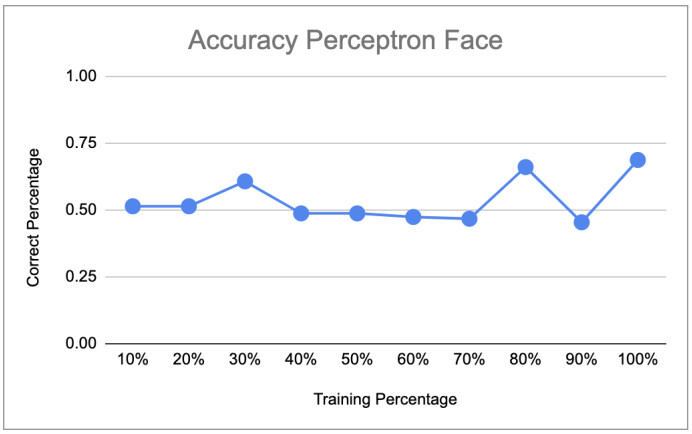

Figure 16: Correct percentage vs. Training Data Percentage

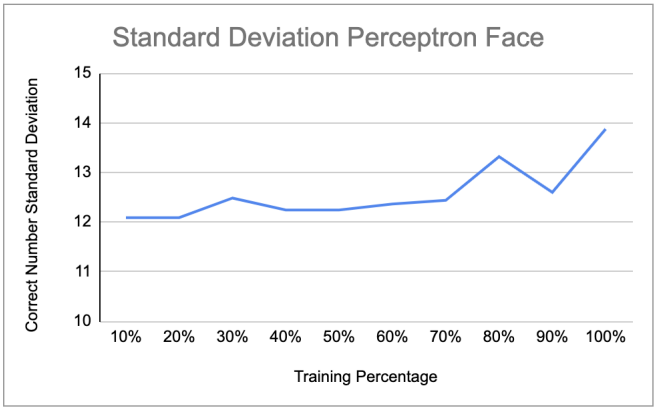

Figure 17: Standard Deviation for Perceptron face