CS 440: Intro to AI **Rutgers: Fall 2021**

Final Project - Face and Digit Classification

December 14, 2021

Names: Ryan Coslove (rmc326), Shane Ngai (sn718), Bryan Sun (bs893)

Part: . Part 1 - Image Input

Skeleton code was used from the Berkeley's wesbite at https://inst.eecs.berkeley.edu// cs188/sp11/projects /classification/classification.html for inputs and feature extractions.

Part: . Part 2 - Algorithms

Naive Bayes

For Naive Bayes digits classifier we used a set of pixel features. For each pixel ϕ_j it can take either value of a white pixel as 0 or a black/grey pixel as 1. We created a dictionary, key labeled Y, containing the coordinate of the pixels as keys containing 0s and 1s.

For training and Tuning, we used the following equation for probability:

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

To get the probability, we used the dictionary Y and values for the total number of occurrences.We iterated through the data set and incremented the label's value by 1 each time to get the total count of all labels. We could then calculate $P(Y = y)$ for each label and store them in the dictionary. Conditional probability is the following Laplace smoothing equation with constant k:

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

Each legal label, feature, and legal feature value, we took the count plus k, then divided by its label count plus the total count of legal feature values times k because we want to eliminate $+k$ from the sum.

For classification, we computed the log probability of $P(y|f_1, ..., f_n)$ because the probability we would receive would be too small. Making the probability a log makes the numbers easier to compare. We computed log probability for each legal label using the following equation:

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

The label with the maximum probability became our guess.

For digits classifying, we tested our program through increments of 10% (10%, 20%, 30%,..., 90%, 100%) of the total training data size. Standard deviation and accuracy was found on 100% of the total testing data size. The following graphs were created to show the time for training (in seconds), accuracy (in percentage), and standard deviation of the data sizes tested.

Naive Bayes Digits Classification:

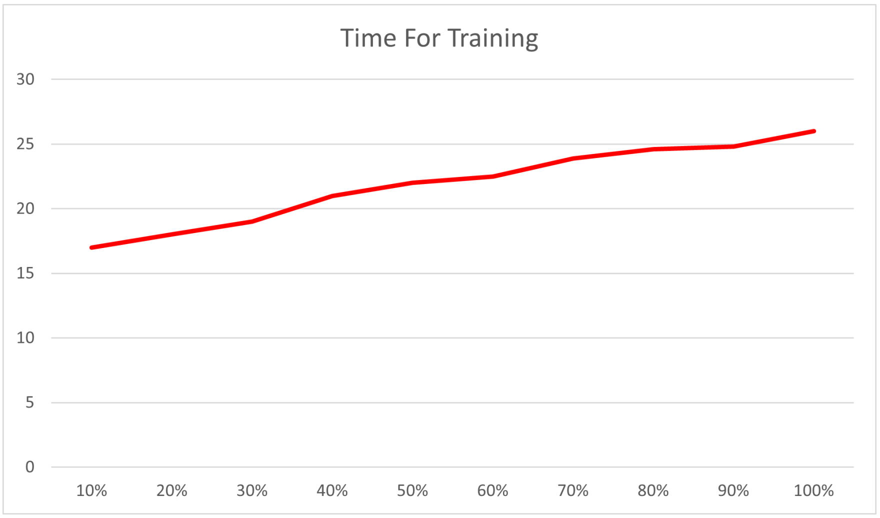

Figure 1: Naive Bayes Digits Classifier Training

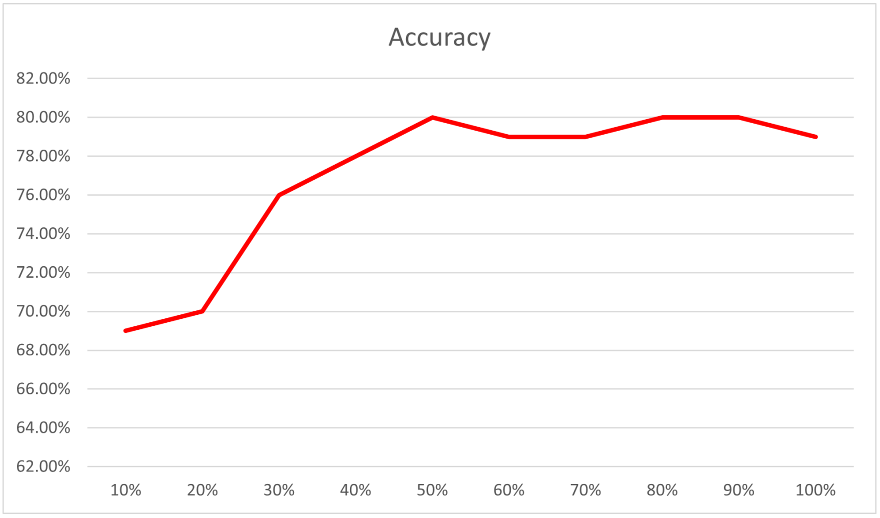

Figure 2: Naive Bayes Accuracy of Digits Classification

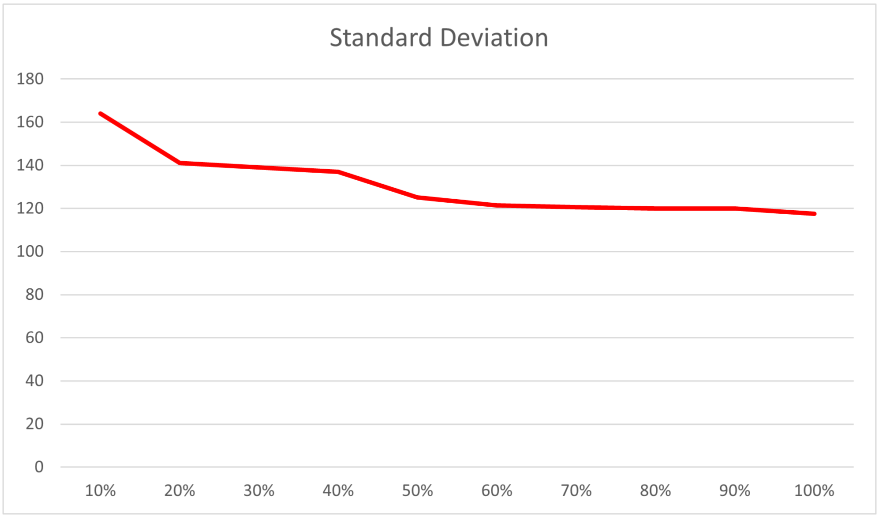

Figure 3: Naive Bayes Standard Deviation of Digits Classification

Naive Bayes Faces

For Faces classifying we tested our program in increments of 10% (10%, 20%, 30%,..., 90%, 100%), of the total training data size. Standard deviation and accuracy was found on 100% of the total testing data size. The following graphs were created to show the time for training (in seconds), accuracy (in percentage), and standard deviation of the data sizes tested.

Naive Bayes Faces Classification:

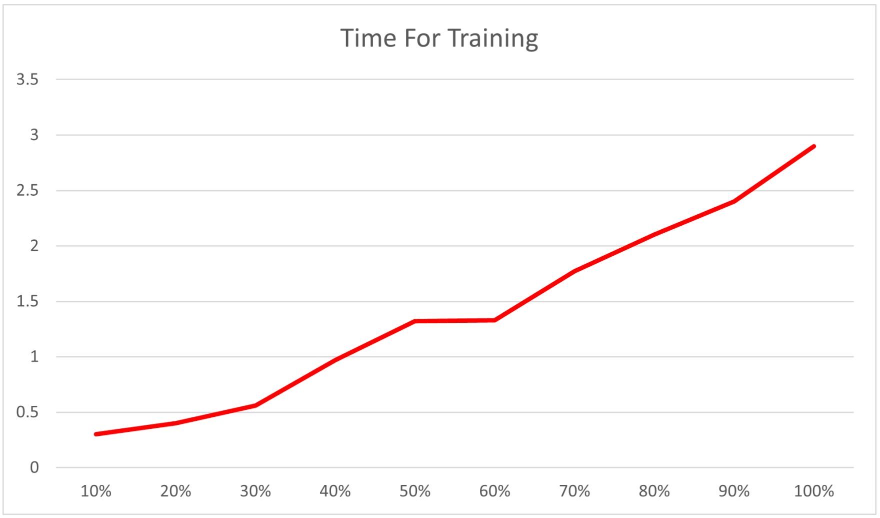

Figure 4: Naive Bayes Faces Classifier Training

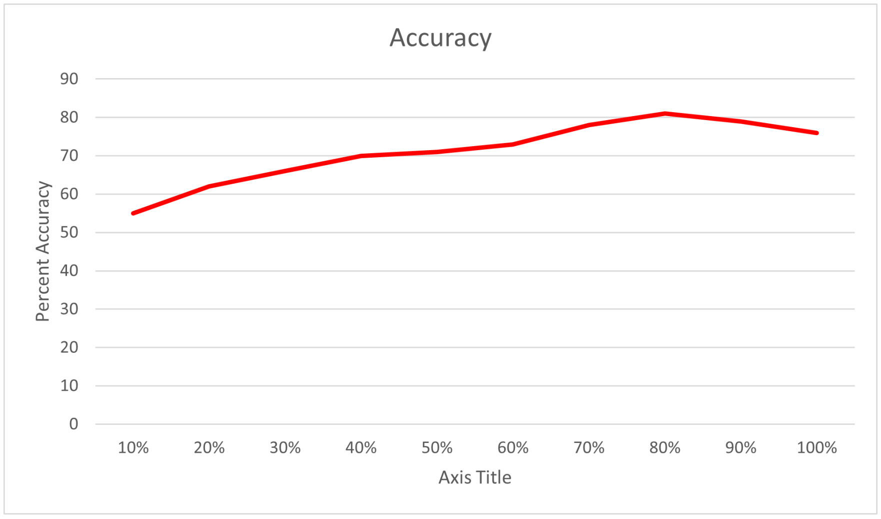

Figure 5: Naive Bayes Accuracy of Faces Classification

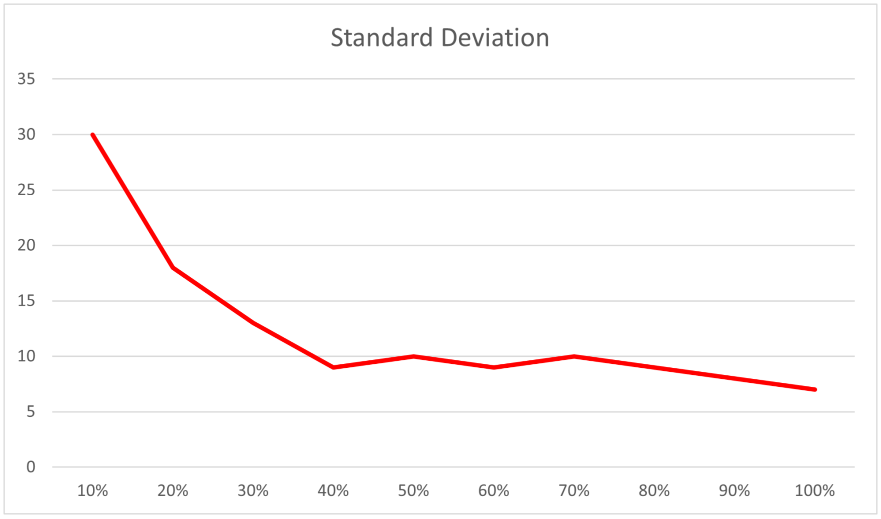

Figure 6: Naive Bayes Standard Deviation of Faces Classification

Perceptron

Perceptron Digits Classification

We approached this algorithm similar to how we did Naive Bayes as just a set of pixel features. For each pixel ϕ_j it can take either value of a white pixel as 0 or a black/grey pixel as 1. We created a dictionary,key labeled Y, containing the coordinate of the pixels as keys containing 0s and 1s. In the algorithm, we initialized all weight to 0s from the beginning then modified them.

To train the program, we set a max iteration number. When the number was reached the program terminated the training sequence. We kept a list of weights to features in the dictionary. The key is the legal label and the value is the list of weights. We iterated over all training data and computed a list of numbers of all legal labels using this equation:

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

We picked the label with the highest $f(x_i, w)$ as our guess. If $guess == label$, then we correctly predicted the label correctly and no change was necessary. If $guess! = label$, then the weight list for the correct label was too small and the weight list for our guess was too large. This was our proceeding step:

for $i = 1, 2, ..., j : weights[label][i]+ = \phi_i(datum)$

$w_o+ = 1$

for $i = 1, 2, ..., j : weights[label][i]- = \phi_i(datum)$

$w_o- = 1$

This process was repeated until nothing changed in one iteration or the max iteration was reached.

For classification of a datum, we computed a list of numbers corresponding to all legal labels using the same equation as before:

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

and again picked the label with the highest $f(x_i, w)$ as our guess.

For digits classifying, we tested our program through increments of 10% (10%, 20%, 30%,..., 90%, 100%) of the total training data size. Standard deviation and accuracy was found on 100% of the total testing data size. The following graphs were created to show the time for training (in seconds), accuracy (in percentage) and standard deviation of the data sizes tested.

Perceptron Digits Classification:

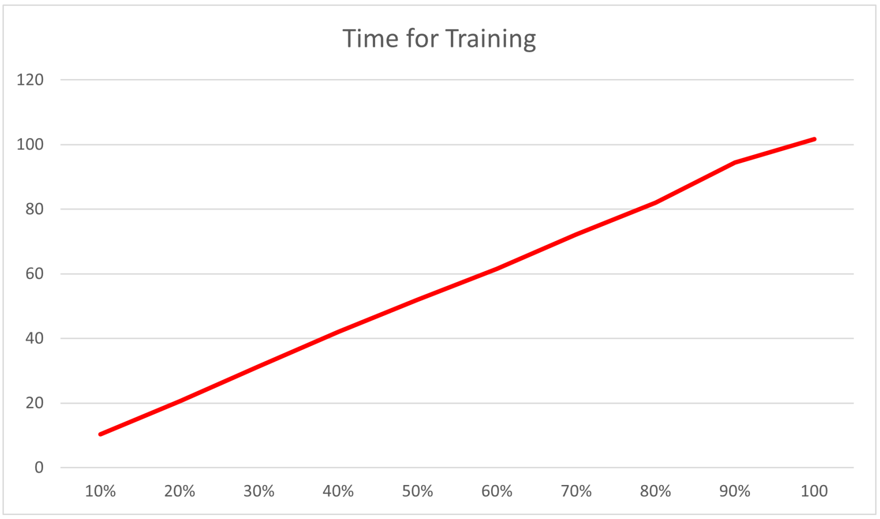

Figure 7: Perceptron Digits Classifier Training Time

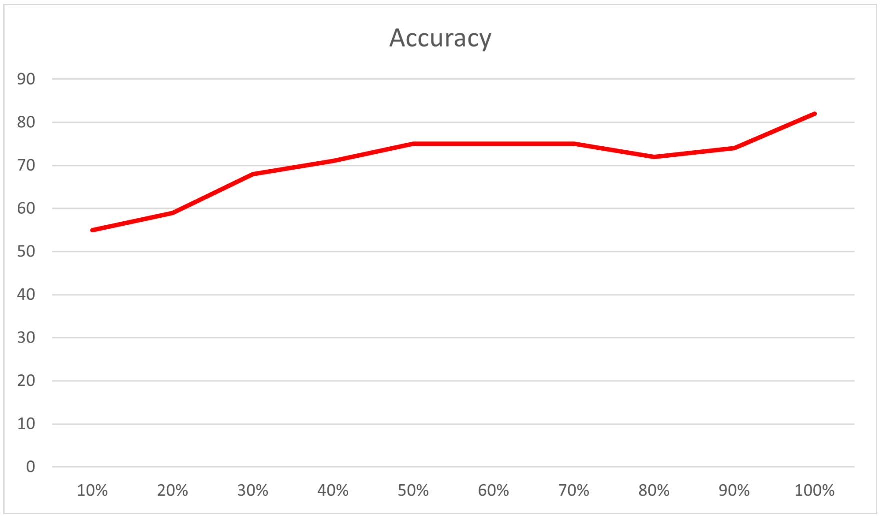

Figure 8: Perceptron Accuracy of Digits Classification

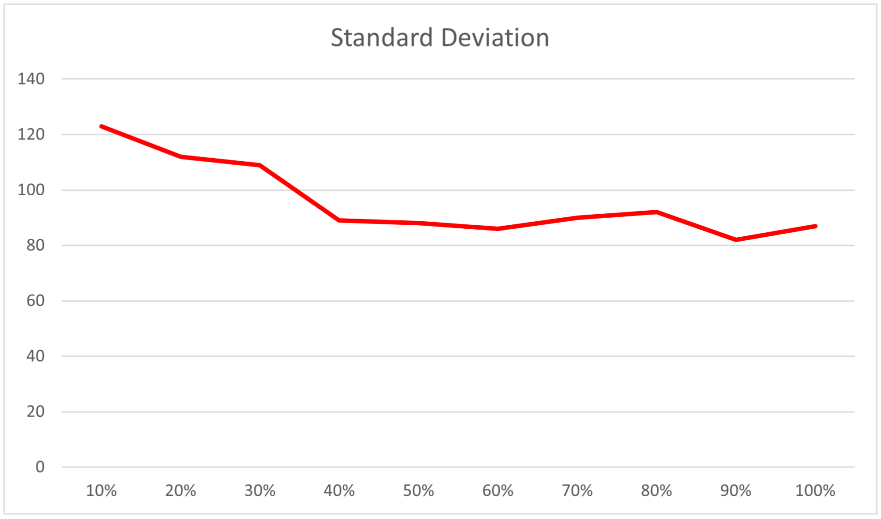

Figure 9: Percepton Standard Deviation of Digits Classification

Perceptron Faces

For Faces classifying we tested our program in increments of 10% (10%, 20%, 30%,..., 90%, 100%), of the total training data size. Standard deviation and accuracy was found on 100% of the total testing data size. The following graphs were created to show the time for training (in seconds), accuracy (in percentage) and standard deviation of the data sizes tested.

Perceptron Faces Classification:

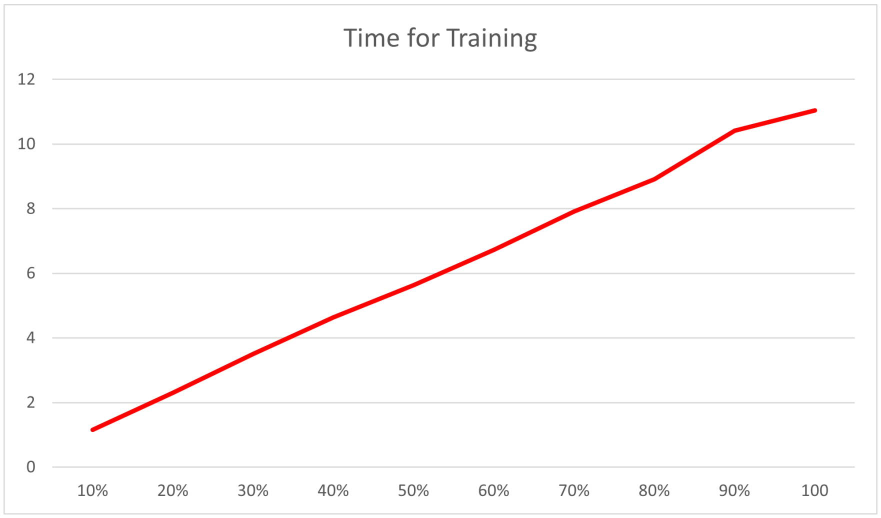

Figure 10: Perceptron Faces Classifier Training

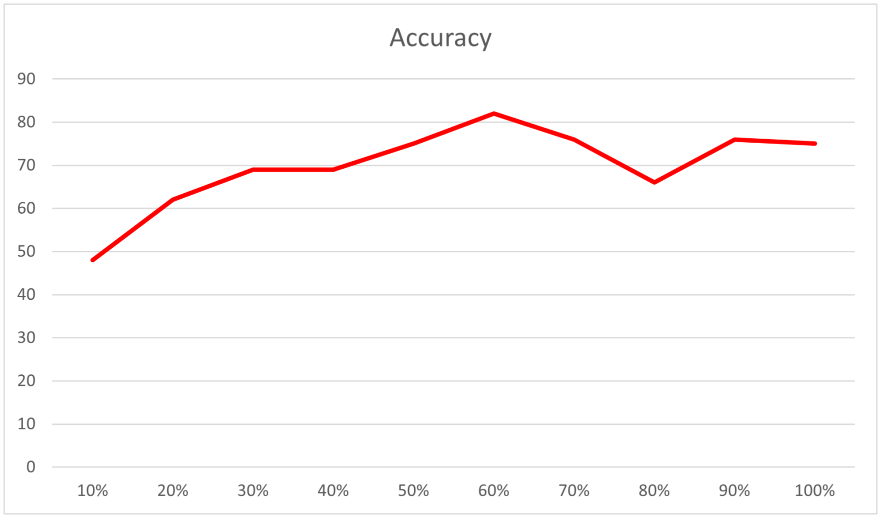

Figure 11: Perceptron Accuracy of Faces Classification

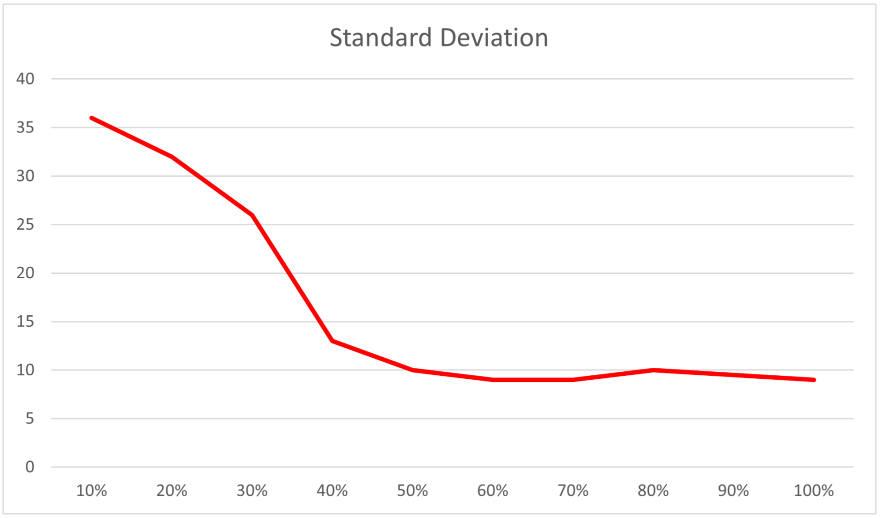

Figure 12: Perceptron Standard Deviation of Faces Classification

K-Nearest Neighbor

We approached this algorithm by taking a sample, calculating its distance to all training samples, then classifying to the appropriate label the majority of K closest samples are. We implemented KNN for its use of cosine distance. Cosine is the angle between length vectors n in n-dimensional space and we used m and n as our distance vector variables. The following equation was used to determine the distance:

$dist(m, n) = 1 - \cos(\theta)$

$dist(m, n) = 1 - \frac{m*n}{\|m\|\|n\|}$

KNN uses a geometric series to determine the distance of the function. This distance is the length of the line segment connecting two points and was calculated with the following equation:

$dist(m, n) = \sqrt{\sum_{i=1}^{z}(m_i - n_i)^2}$

For digits classifying, we tested our program through increments of 10% (10%, 20%, 30%,..., 90%, 100%) of the total training data size. Standard deviation and accuracy was found on 100% of the total testing data size. The following graphs were created to show the time for training (in seconds), accuracy (in percentage), and standard deviation of the data sizes tested.

K Nearest Neighbor Digits Classification:

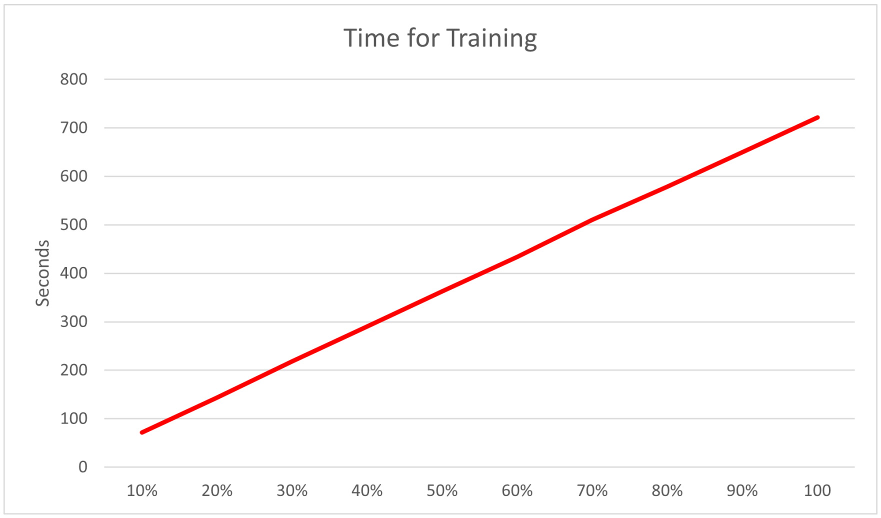

Figure 13: K Nearest Neighbor Digits Classifier Training Time

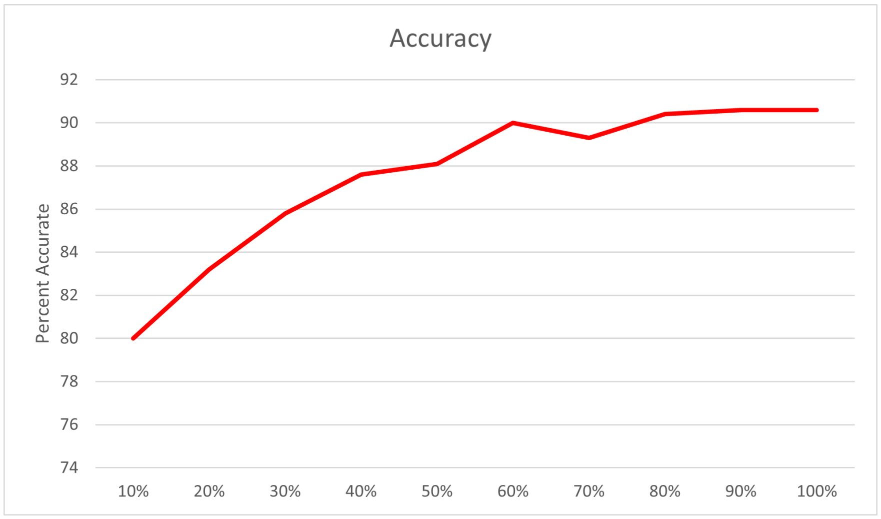

Figure 14: K Nearest Neighbor Accuracy of Digits Classification

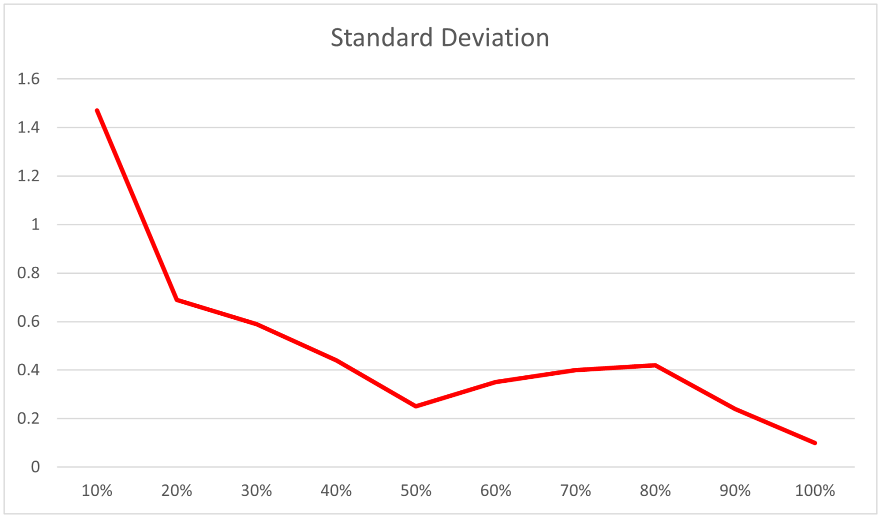

Figure 15: K Nearest Neighbor Standard Deviation of Digits Classification

K Nearest Neighbor Classification of Faces

For Faces classifying we tested our program in increments of 10% (10%, 20%, 30%,..., 90%, 100%), of the total training data size. Standard deviation and accuracy was found on 100% of the total testing data size. The following graphs were created to show the time for training (in seconds), accuracy (in percentage), and standard deviation of the data sizes tested.

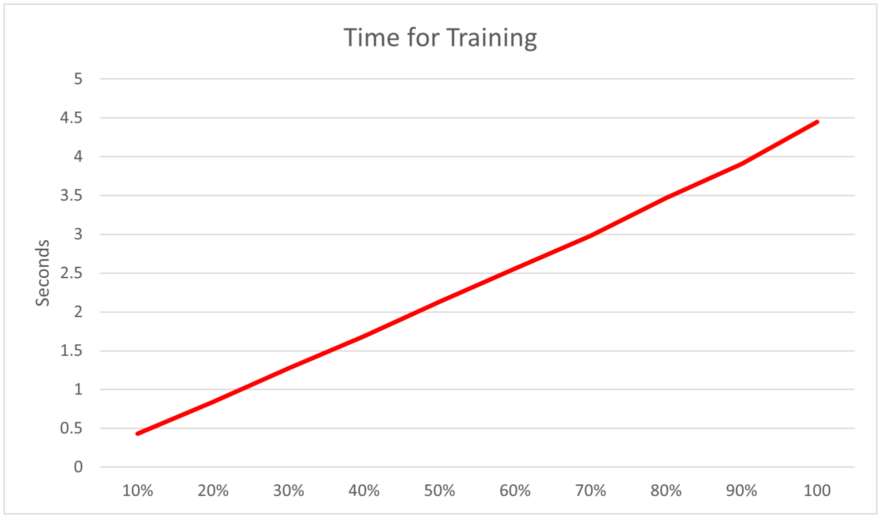

Figure 16: K Nearest Neighbor Faces Classifier Training Time

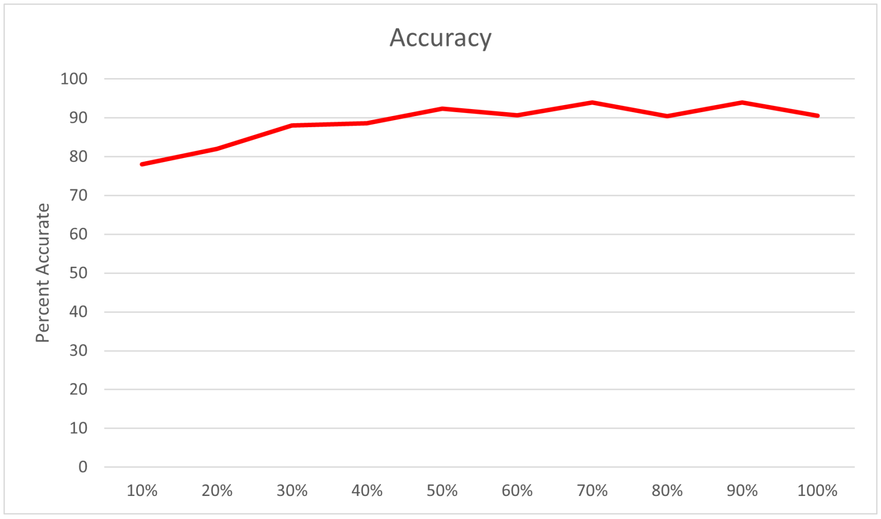

Figure 17: K Nearest Neighbor Accuracy of Faces Classification

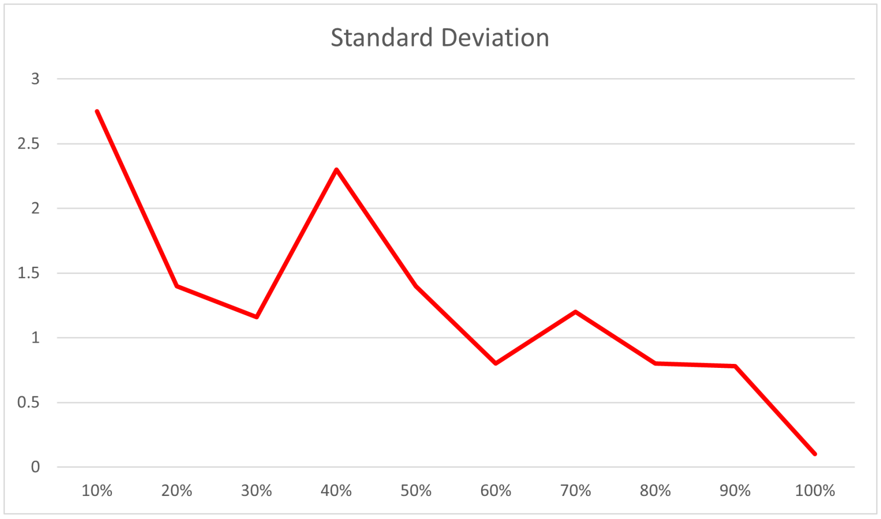

Figure 18: K Nearest Neighbor Standard Deviation of Faces Classification

---

Part: . Part 3 - Discussion

For all experiments performed, we crossed the 70% accuracy barrier/goal for this project, given enough training. Time spent on the training process showed it was proportional to the amount of data provided. The more date provided, the longer it took to finish the training process. The opposite is also true (less time for less data).

For the accuracy of the programs, it was also proportional to the amount of data provided. This was mostly true for the beginning though. When the training data reached 40-50%, it often maintained a certain accuracy level and did not drastically increase in performance after that point. Sometimes when the data is over fitting, the accuracy slightly decreased.

For the standard deviation of the programs, it was proven to be inverse proportional for data provided in the beginning vs towards the end. Again, often around 40-50% margin, the standard deviation did not drastically change after reaching that point as more data is provided. Sometimes the standard deviation increased when more data was provided and began over fitting.

Part: . Part 4 - Lessons We Learned

The lessons we learned is that performance of an algorithm or program given more data does not guarantee more accuracy or better overall performance. The accuracy of the algorithms/programs would often settle near a certain point of accuracy, that point being dependent of the algorithm. When training a program, it is more efficient to perform an experiment or write an algorithm that finds which training data is optimal to finding that settling/converging point. That would better allow us to reduce run time of the training sequence and still meet the accuracy we expect.

Resources

Skeleton code and source files/project references

`https://inst.eecs.berkeley.edu//~cs188/sp11/projects/classification/classification.html`

Laplace smoothing

`https://towardsdatascience.com/laplace-smoothing-in-na%C3%AFve-bayes-algorithm-9c237a8bdece`

Perceptron youtube video

`https://youtu.be/-KLnurhX-Pg`