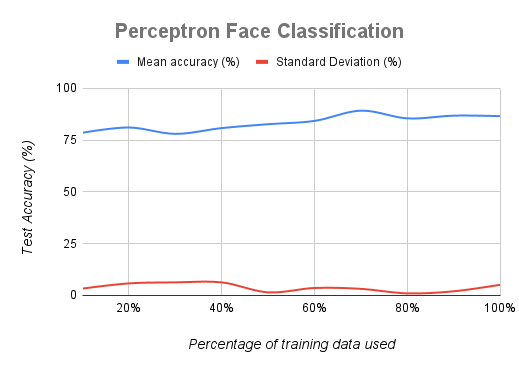
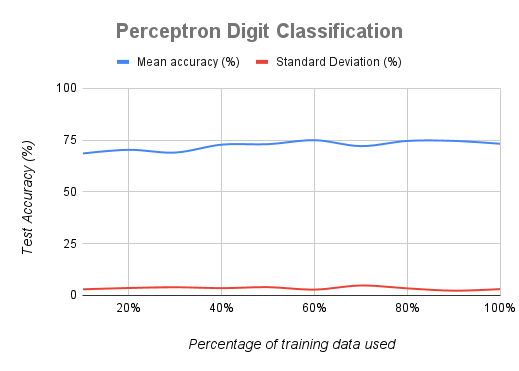
**Final Project Report**

Perceptron - *Face Classification*

* **Features:** The features used for the face classification portion of the Perceptron classifier were simple to calculate. The training images were split into (6 pixel) x (7 pixel) divisions, and the number of ‘#’ characters in each of these divisions were counted. We considered each of these counts to be a distinct feature, making a total of 100 features for each face image. We slightly tuned the parameters for the size of the divisions, but found the (6 pixel) x (7 pixel) divisions to be both great for accuracy and quick to calculate.
* **Training Time:** The initial training time for a random sample of 10% of the data set (45 images) resulted in a training time of 0.13 seconds. The runtime gradually grew as we sampled more data, and the final runtime for 100% of the data set (451 images) was 1.26 seconds. This is a very quick runtime considering that the test accuracy we obtained was relatively good.
* **Test Accuracy: **
  + The mean accuracy started surprisingly high given that it only saw 10% of the training data set (45 images). Generally, the accuracy marginally increased as more of the training data set was being used. The test accuracy at 100% use of the training data was 86.53%.
  + The standard deviation maintained a good level throughout all of the testing. The highest it ever got was 6.3% at 30% use of the training data (135 images).

Perceptron - *Digit Classification*

* **Features:** We used the same features for Perceptron - digit classification as we did for Perceptron - face classification. However, since the digit images have different dimensions than the face images, we changed the size of the divisions. After some manual tuning, we landed on using (4 pixel) x (4 pixel) divisions since it had the highest test accuracy, which gives us a total of 49 features. We could have used more features, but we would have to sacrifice low runtime in order to do so.
* **Training Time:** There were many more images in the digit training data set than there were in the face training data set, so the training time took longer for the digit classification. It took 0.49 seconds to train at 10% use of the training data (500 images), and 4.89 seconds at 100% use of the data (5000 images). Again, these times gradually increased as more and more of the training data was used. Overall, this a great runtime considering how many images it had to process.
* **Test Accuracy:**
* The mean test accuracy started at 67.08% at 10% use of data (500 images), and rose to 74.3% at 100% use of data (5000 images). Although these results aren’t quite as good as the results we got for the Perceptron - face classification, they’re still impressive since we used half as many features.
* The standard deviation looks very similar to the one we saw for the face classification. The maximum standard deviation was 6.247% at 70% use of the training data (2000 images).

Naive Bayes - *Face Classification*

* **Features:** We used the same features for the Naïve Bayes face classifier as we did for the perceptron face classifier (see above).
* **Training Time:** The initial training time for a random sample of 10% of the data set (45 images) resulted in a training time of 0.11 seconds. The runtime gradually grew as we sampled more data, and the final runtime for 100% of the data set (451 images) was 1.0 seconds. This is a very quick runtime considering that the test accuracy we obtained was relatively good.
* **Test Accuracy:**

**Chart, line chart

Description automatically generated**

* + The mean accuracy started surprisingly high given that it only saw 10% of the training data set (45 images). Generally, the accuracy marginally increased as more of the training data set was being used. The test accuracy at 100% use of the training data was 96.0%.
  + The standard deviation maintained a good level throughout all of the testing. The highest it ever got was 6.808% at 10% use of the training data (45 images).

Naive Bayes - *Digit Classification*

* **Features:** We used the same features for the Naïve Bayes digit classifier as we did for the perceptron digit classifier (see above).
* **Training Time:** There were many more images in the digit training data set than there were in the face training data set, so the training time took longer for the digit classification. It took 0.24 seconds to train at 10% use of the training data (500 images), and 2.33 seconds at 100% use of the data (5000 images). Again, these times gradually increased as more and more of the training data was used. Overall, this a great runtime considering how many images it had to process.
* **Test Accuracy:**

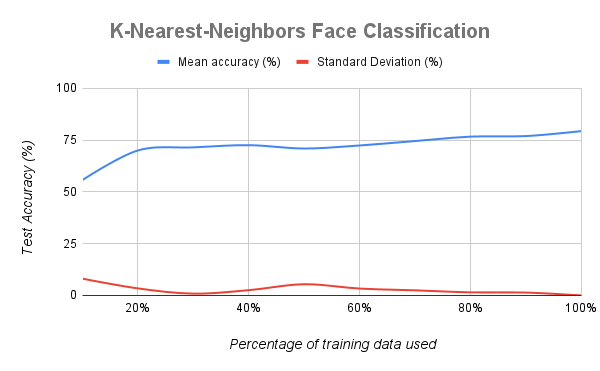
**Chart, line chart, scatter chart

Description automatically generated**

* The mean test accuracy started at 63.0% at 10% use of data (500 images), and rose to 74.3% at 100% use of data (5000 images). Although these results aren’t quite as good as the results we got for the Naïve Bayes - face classification, they’re still impressive since we used half as many features.
* The standard deviation looks very similar to the one we saw for the face classification. The maximum standard deviation was 2.127% at 20% use of the training data (2000 images).

K-Nearest-Neighbors - *Face Classification*

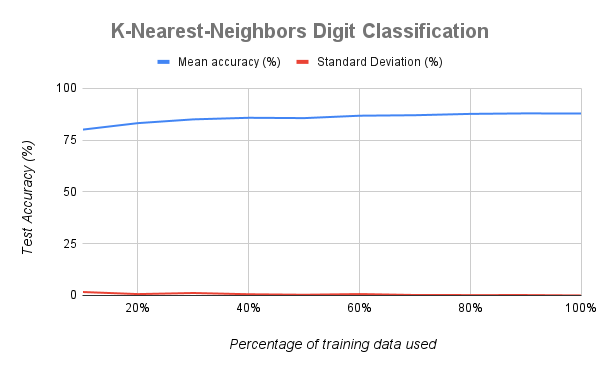
* **Features:** We used three features for the face classification portion of K-Nearest-Neighbors. The first feature is the total number of ‘#’ characters in each image, the second feature is the maximum number of ‘#’ characters in a single division, and the third feature is the location of the division with the maximum number of ‘#’ characters. The divisions are enumerated so each one has a unique number associated with it. The divisions are (5 pixels) x (2 pixels), and the K value we used was 29. We utilized hyperparameter tuning in order to find the parameters that maximize the test accuracy. The code for the tuning is commented out, and starts at line 36 in the “KNearestNeighbors.py” file.
* **Training Time:** It took 0.09 seconds to train at 10% use of the training data (45 images), and 0.32 seconds at 100% use of the data (451 images). These times are significantly better than the other classifiers we’ve seen. One of the reasons why this is the case is because we only used three features. In addition, K-Nearest-Neighbors is generally less computationally expensive than Perceptron or Naïve Bayes.
* **Test Accuracy:**



* + The mean test accuracy started at 59.68% at 10% use of data (45 images), and rose to 79.33% at 100% use of data (451 images). We also see quite a big jump in accuracy from using 10% of the training data to using 20% of the training data. From that point, there’s a steady, marginal increase in accuracy.
  + The standard deviation seems to decrease as the use of training data increases. The maximum standard deviation was 6.085% at 10% use of the training data (451 images).

K-Nearest-Neighbors - *Digit Classification*

* **Features:** Unfortunately, the three features that we used for the face classification portion of K-Nearest-Neighbors did not work well for the digit classification portion. We obtained a test accuracy of around 48% at 100% use of the training data (5000 images) when we used only those three features. Therefore, we added more features as a means to increase the test accuracy. Ultimately, we found that using the number of ‘#’ characters in each division as features worked well. Using hyperparameter tuning, it was found that dividing the images into (2 pixel) x (2 pixel) squares resulted in the highest accuracy when accompanied by a K value of 3.
* **Training Time:** It took 4.26 seconds to train at 10% use of the training data (500 images), and 45.77 seconds at 100% use of the data (5000 images). This is by far the longest runtime of all the classifiers. It makes sense that the runtime is so extreme because the quantity of images being processed is so vast, and each image has 196 features.
* **Test Accuracy:**



* + The mean test accuracy started at 79.14% at 10% use of data (500 images), and rose to 87.8% at 100% use of data (5000 images). The learning curve sees decent growth early on, then starts leveling off after using 40% of the training data.
  + The standard deviation is very low for the entirety of the training process. The maximum standard deviation was 0.503% at 10% use of the training data (500 images).