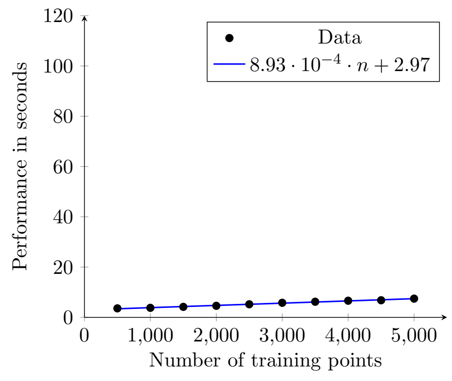
Introduction to Artificial Intelligence: CS440

Final Project

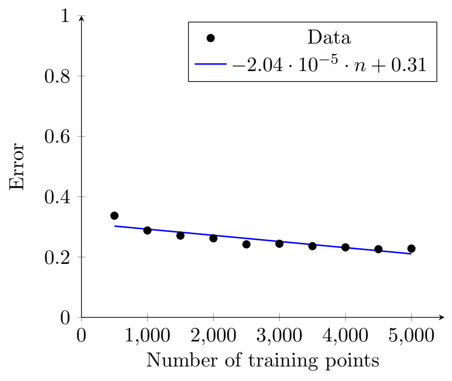
Fin Herbig - fh222 - 186000103 Zachary Ford - zjf6 - 184005631
Jaidev Singh Saberwal - js2592 - 188002130

Part 1: Performance Comparison:

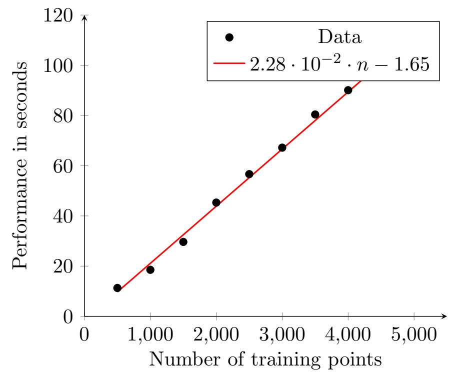
To compare the performance of each implementation, we recorded the performance data and from that, generated functions based on the number of data points used for training. These functions can be seen on the graphs below based on the data collected on the digits dataset.



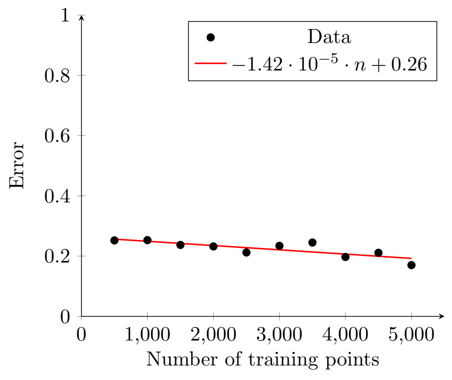
(a) Naive Bayes Classifier Performance



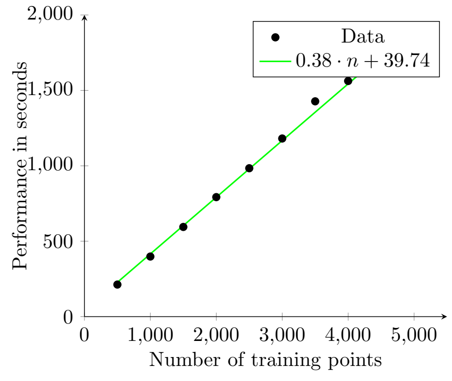
(b) Naive Bayes Classifier Prediction Error



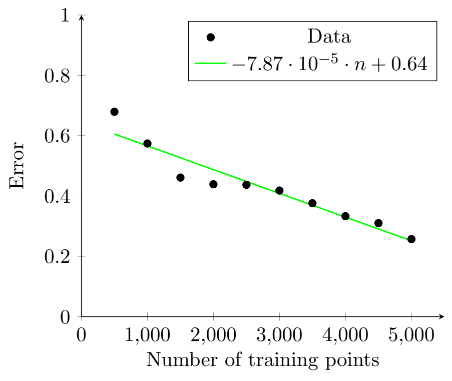
(c) Perceptron Performance



(d) Perceptron Prediction Error



(e) Neural Network Performance



(f) Neural Network Prediction Error

These graphs show how for every algorithm, the runtime is proportional to the amount of training data points and the error is inversely proportional to the number of training data points. We collected data on the faces dataset as well which, while generally having a lower prediction error, had fewer data points. A larger dataset is more useful for the purposes of collecting data as we are able to get a more accurate representation. Otherwise, the faces dataset showed similar results to the numbers dataset. One thing to note is that the scale for performance in the graph depicting the neural networks performance shown in graph (e) is scaled differently due to it taking far longer than the other two algorithms. Comparing just the performance data, the Bayes classifier has the best performance of any of the algorithms and it scales the best as more training data is introduced. On the other hand, the time training takes for the neural network to run is much greater than the other algorithms and it increases the most as more training data is added. However, the neural network is also able to "learn" much faster than the other algorithms as shown in graph (f). The other two algorithms are similar when it comes to learning as well where it seems as though the native Bayes classifier is only very slightly better than the perceptron. From the data we have collected, it appears as though the native Bayes classifier is the best for the tasks we assigned to it due to it being both the best performing and had a good learning rate. However, we expect that as the amount of training data is increased, the perceptron and neural network will have less prediction error at the cost of significantly worse performance.

Part 2: Naive Bayes Classifier

The Naive Bayes Classifier algorithm is based on Bayes theorem where each feature (random variable) is independent of one another. However, for finding $p(y|x)$, we can use a likelihood ratio instead of calculating $p(x)$ in the denominator since it is effectively a constant. This means we only need to find $p(x|y)$ and $p(y)$ and our likelihood ratio will look like:

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)} \propto$$

$$\frac{p(x|y = true)p(y = true)}{p(x|y = false)p(y = false)}$$

To find $p(y)$ and $p(x|y)$ we are able to estimate it from the training data. For $p(y = true)$, we find how often y is true and divide that number by the number of training examples. To find $p(y = false)$ we do the same where y is false. For $p(x|y = true)$ we can estimate it by dividing the number of times a feature takes on a specific value for a label being true by the number of times that label is true. We then multiply those results over every possible value the feature can take to get $p(x|y = true)$. For $p(x|y = false)$, we do the same again but with the label being false. After we find both these estimates, we can plug them into our likelihood function to get our result.

While implementing the naive Bayes classifier algorithm, we learned that even though it is based purely on probability, It was able to classify some faces/digits it has not seen enough when each data point has no relation to each other in the data set. It was assumed the the exact position of the center of the images mattered however, through testing, it did not.

Part 3: Perceptron

Instead of using probabilities, the perception algorithm uses weights corresponding to labels instead. Given a set of corresponding features and weights, we calculate a score for each label and take the label with the largest score as the "true" label for the set of given features. We then compare this label to the actual given label, if they are the same we move on to the next set of features. If they are not the same, we update the weights by adding or subtracting the feature value of the given feature. When the algorithm runs through every data point and the weights are not changed, the algorithm is complete.

While implementing the perceptron algorithm we learned how rather than probabilities, the weight in a connection can be used to classify something. Actually implementing the algorithm for modifying the weights was very insightful as well as we were able to see directly how features increase or reduce the weights. This information was especially helpful with implementing the neural network algorithm due to a perceptron being a single layer neural network.

Part 4: Neural Network

A neural network is similar to a perceptron but where a perceptron has a single layer, a neural network consists of one or more layers. These layers are processing steps which allow for more complex functions to be expressed. In our implementation, we used a sigmoid function to calculate the values of nodes in the following layers based on the previous node values and the weights between those nodes. The value at each node represents the nodes feature value. The process for modifying the weights between nodes done through back propagation. This is done by finding the δ error for each weight and modifying the weight based on the δ error. The δ error for each weight is found by taking the derivative of both the total network error and the sigmoid function. All the weight changes are found by using the previous layers errors to make small changes depending on how important each node was for each specific training set

While implementing a full neural network, we learned how important getting the back propagation correct is. Every little change to the weights on each node greatly impacts what the final out put is. It was a huge challenge to make a versatile algorithm that can working with a range of input and output sizes. Any incorrect index or changed variable can result in the complete failure of the system. We also found that the training time for the Neural Networks are much longer than the others, while they were measured in either milliseconds or seconds, the Neural Nets training time was measured in the multiple minutes. This caused the time to create the AI to take much longer than expected as rather then finding out your code did not work it seconds, sometimes it takes up to ten minutes to find out your code failed. One thing that helped up succeed in this algorithm was to repeat the training data through after it com pleated it the first time, this improved the accuracy's even with the smaller data sets

Part 5: Conclusion

This project gave us great insight into some different classification algorithms. It was especially interesting to see how a neural network expands from a perceptron. From the data that we have collected we found the best algorithm do use for our dataset was the bayes classifier. However, as stated before, it is expected that both the neural net and perceptron will perform better with an increased training dataset. This is one of the limitation of the project, had we used a larger dataset, our results would have been more representative of the actual algorithms implemented. With these extra data points we may have been able to confirm whether or not the perceptron and neural network would have actually performed better than the naive

Bayes classifier. Another limitation was the machine on which we tested our algorithms. Our data was collected on the iLab machines, however, it would have been interesting to see how the performance scales on different hardware.