AI Final Project Report

Nicholas Prezioso (njp107). Michael Serpico (mvs71), and Aziz Rahman (ar1174)

May 6, 2018

For this project we implemented 2 different machine learning algorithms. We implemented the Naive Bayes and Perceptron algorithms. These algorithms are used to take in some type of data as input and then use that data to help identify what those pictures are. For instance in our project, we looked at pictures of numbers to determine the actual number they represent, and then we looked at pictures of faces to determine whether or not the pictures correspond with faces. We will go through each of these algorithms, and we will explain how fast and accurate the algorithm performed and why based on the amount of testing data we used.

Figure 1.1: Naive Bayes with Digits

Data Used in Training	25%	50%	75%	100%
Percent Correct (Validation)	59%	70%	71%	73%
Percent Correct (Testing)	51%	63%	65%	64%
Time to train (seconds)	0.297	0.5	0.672	0.875

The data for Naive Bayes using digits can be seen in Figure 1.1. This figure shows that the algorithm increased in correctness as we used more data in testing. Our implementation however never surpassed 70% in correctness when it came to determining what number was being shown. The reasoning for this could be because of the features that we chose to use. Features are a way of getting data about a specific part of the picture for the algorithm. For example, we used whether or not a pixel was on as our feature. So if a pixel at 12,3 was on, the feature would be set to 1. We only implemented our algorithm with each pixel as the features. If we were to add or change our features to include more of them or use more useful features, it is possible our percent correct would be greater. Also, it can be seen in the figure that the time to train increased the more data we used.

Figure 1.2: Naive Bayes with Faces

Data Used in Training	25%	50%	75%	100%
Percent Correct (Validation)	65%	81%	99%	98%
Percent Correct (Testing)	53%	56%	81%	82%
Time to train (seconds)	1.391	2.5	3.594	4.672

Looking at the faces for Naive Bayes, (Figure 1.2) our data was marginally more accurate. We reasoned that the cause of this was the features that were used in learning. The features we used for faces were determining if a pixel was an edge. This most likely gives us more information about the picture compared to whether the pixel was on or off. We were able to get our data above 70%, which is considered a success. Also, because the feature was harder to check, the time required was higher for faces than it was for digits.

Figure 2.1: Perceptron with Digits

Data Used in Training	25%	50%	75%	100%
Percent Correct (Validation)	37%	60%	68%	64%
Percent Correct (Testing)	43%	60%	63%	62%
Time to train (seconds)	1.078	2.11	3.25	4.343

When using perceptron with digits, our accuracy was about the same as it was for using Naive Bayes (see Figure 2.1). The reason for this is again the fact that we could have implemented better features. However, the time takes slightly longer than Naive Bayes. The reason for this most likely is how the algorithm itself works in comparison to Naive Bayes. Perceptron adjusts the weights constantly in the algorithm, and that alone could take a some time.

Figure 2.2: Perceptron with Faces

Data Used in Training	25%	50%	75%	100%
Percent Correct (Validation)	75%	87%	93%	92%
Percent Correct (Testing)	65%	74%	71%	78%
Time to train (seconds)	1.375	2.719	3.859	5.438

Using perceptron with faces, things went similarly to what was described above. Looking at the data in Figure 2.2, the time it takes is slightly higher than perceptron with digits and higher than Naive Bayes. This is because of the feature that is used to train and because perceptron has to adjust weights constantly. Also the accuracy was above 70%, which means the algorithm is working as intended.

Takeaways/Other Points:

It is important to note that when running Naive Bayes, we let it find its own k value instead of manually putting in the k value when we ran it. Another thing to note is that we used the default number of iterations for perceptron. If we had increased the number of iterations, we would have gotten slightly more accurate numbers. It is also important to note that we did not randomly train our algorithm given our data set. It is trained in the order given by the Berkeley code. From using these two machine learning algorithms, we learned how important it is to tell the algorithm what to look for when learning. In our case, it was apparent that just using whether a pixel was on or not was not the best way to have the machine learn. We also learned just how important training can be for machine learning. With just the increase of a few percent in the amount of data we used to train our machine, the percentage of correct guesses increased substaintially.