Sam Chamberlin
CS 472 – Machine Learning
Dr. Martinez
September 22, 2020

Perceptron Lab

1. **Perceptron Learning Algorithm**

   A perceptron class was written with features including user selection of shuffling data, custom initial weights, learning rate, and number of epochs. The perceptron class can randomly split training and testing sets and then fit, predict, and score data sets. Weights are updated stochastically, after each training instance. If number of epochs aren't defined by the user, then the class implements its own stopping criteria based on accuracy improvement. Python code is included in the attached Perceptron Lab (code) submission.

2. **Test Files**

   Two ARFF files were created with one being linearly separable and the other not in order to test the perceptron class. The two dataset files, "lin_sep.arff" and "not_lin_sep.arff" are also included in the attached Perceptron Lab (code) submission.

3. **Training Observations**

   The perceptron was trained on both sets using a number of different learning rates from 0.01 to 1. It was found that for the linearly separable data, the learning rate only had a minimal effect on the final weights, resulting in a slightly different sloped decision line, however, the model completed with the same accuracy (100%) and in the same number of epochs for every varying learning rate case tested. It was interesting to note that changing the number of epochs run with a user defined epoch count had no effect at all on the model's results. Similarly, for the non-linearly separable data, it was found that the learning rate had no effect other than increasing each of the weights. The accuracy for this data is 75%.

4. **Plots of Instances**

   Plots of both data sets are included below with the decision line which separates the active (on) points as well as the points that are inactive (off) based on their labels.

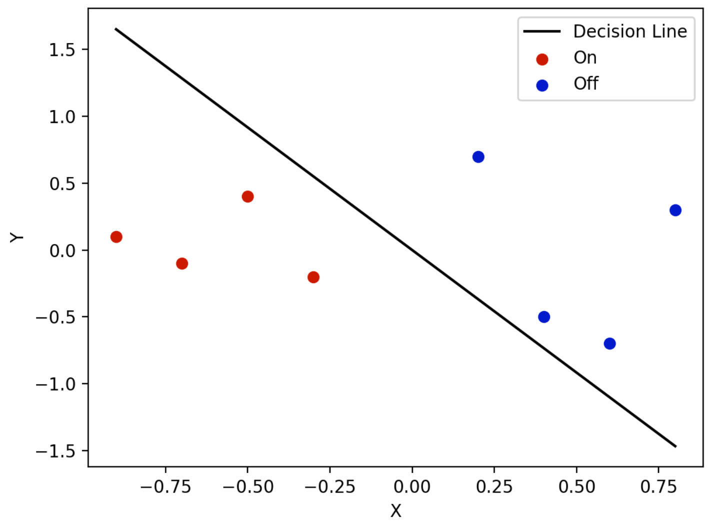

Figure 1: Linearly separable data plot

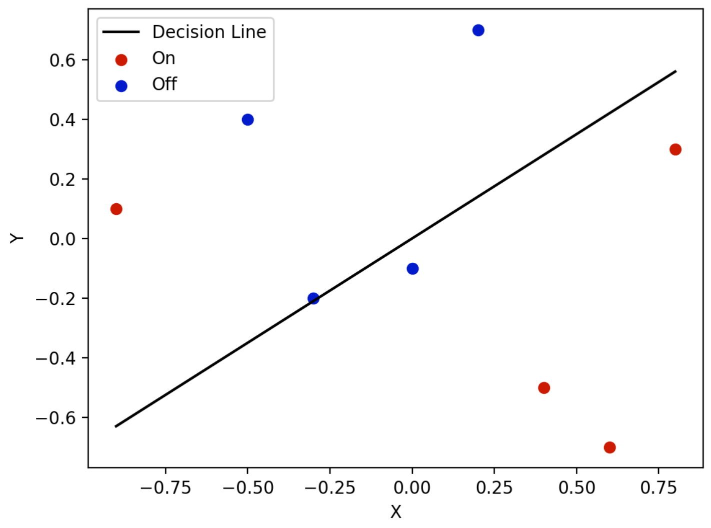

Figure 2: Non-linearly separable data plot

5. Predicting Voter Affiliation

The perceptron model was used to learn a set of voting data to determine if a voter was Republican or Democratic based on 16 features that were observed. The model was run in five separate trials. Data was shuffled in between epochs and split randomly each run with 70% of the data used for training and 30% of the data for testing.

Table 1: Results of the five trails on the voter data set

	Final Training Set Accuracy	Final Test Set Accuracy	# of Epochs
Trial 1	0.97	0.95	6
Trial 2	0.90	0.88	6
Trial 3	0.94	0.94	7
Trial 4	0.96	0.97	6
Trial 5	0.92	0.93	6
Average	0.94	0.93	6

Looking at the weights of each feature, it is interesting to note how much they change relative to each run. A few patterns emerged after observing the weights meticulously, run after run, however. The features that seem to have the greatest effect on a voter's party affiliation are the "Physician Fee Freeze" and "Anti-Satellite Test Ban" attributes. On the other spectrum, the features that were consistently reporting low weights, and thus had little influence on a voter's party affiliation, were the "Superfund Right to Sue" and "Duty Free Exports". Again, it was surprising how the weight values for a majority of the attributes were so volatile which made it difficult to determine the features that truly have the greatest and least influence.

Below an observation of average misclassification rate versus epoch number is included. In order to get these results, the accuracies for both the training set and test set were collected at each epoch and used to determine how many people in the voter data set were misclassified by the model. As expected, a downward sloping trend that levels out as it approaches 0% is present.

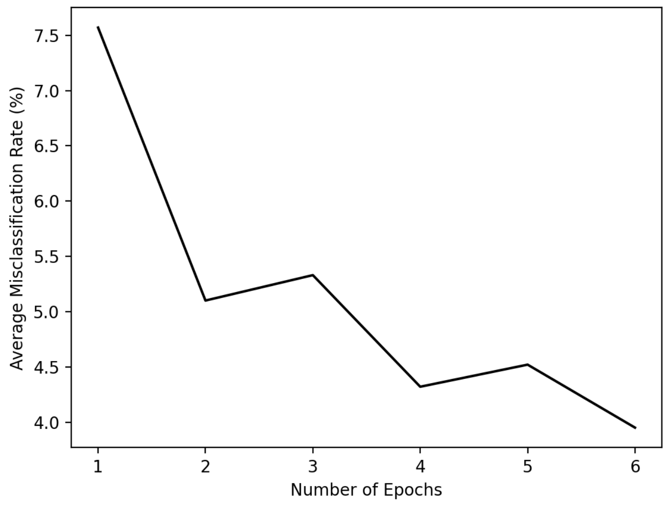

Figure 3: Average misclassification of voters at every epoch interval

6. Comparison to the Scikit-Learn Toolkit Perceptron Algorithm

In order to compare the performance of this model, the perceptron algorithm provided by Scikit-Learn was used to learn the voter data as well. The final test set accuracy was 96%. This value, while on the higher end of the typical test accuracy provided by the hand-written perceptron, is still right in the range already observed. One aspect of the Scikit-Learn perceptron algorithm is its clean structure and the vast amount of hyperparameters available. From what I can tell, the two algorithms work the same, however, the Scikit-Learn perceptron algorithm is incredibly consistent. The accuracy reported didn't change until I altered the hyperparameters, such as shuffling. But even then, once shuffling was activated, the accuracy reported was unchanging.

The Scikit-Learn algorithm was also tested on "not_lin_sep.arff". It was surprising to see that the refined algorithm reported 62.5% each time. The best possible decision line, however, is able to get 75% accuracy as shown by the hand-written perceptron algorithm.