Perceptron

Sean Wade

OVERVIEW

The perceptron algorithm is a simple iterative approach to linearly separating data. This create a binary decision boundary for classification problems.

1. Implement Perceptron

To start this project I had to first create my machine learning package in python. The package is entitled mltools and it includes core functionality that I will use for the rest of the semester. The most important piece for this project was my DataManager class. This class parses an .ARFF file and stores it as member variables. The main data is in a numpy array for manipulation and the labels are in a list. Along with storing the data it has important data processing functions like normalizing the data, getting the mean, and breaking into a test-train split. Mltools also includes a abstract model class that serves to predict, fit, and analyze the data. I built my perceptron with this framework.

2. Create ARFF Files

In order to have a simple test for my perceptron, I created two synthetic datasets. These datasets had two binary classes with four instances of each. The first was linearly separable, with all the ones in the first quadrant and the zeros in the third. The second wasn't separable with randomly picked points. The data is attached in the data folder of this submission.

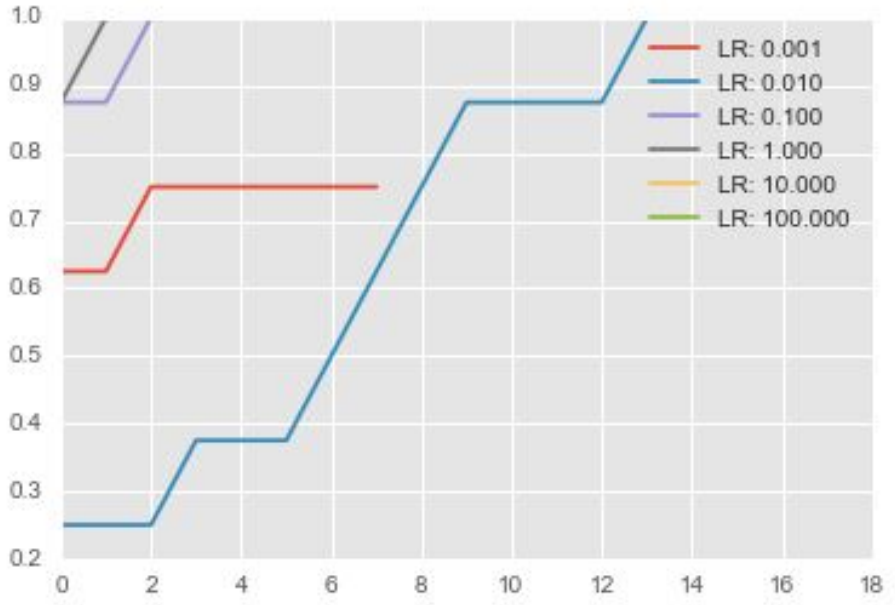
3. Training Hyperparameters

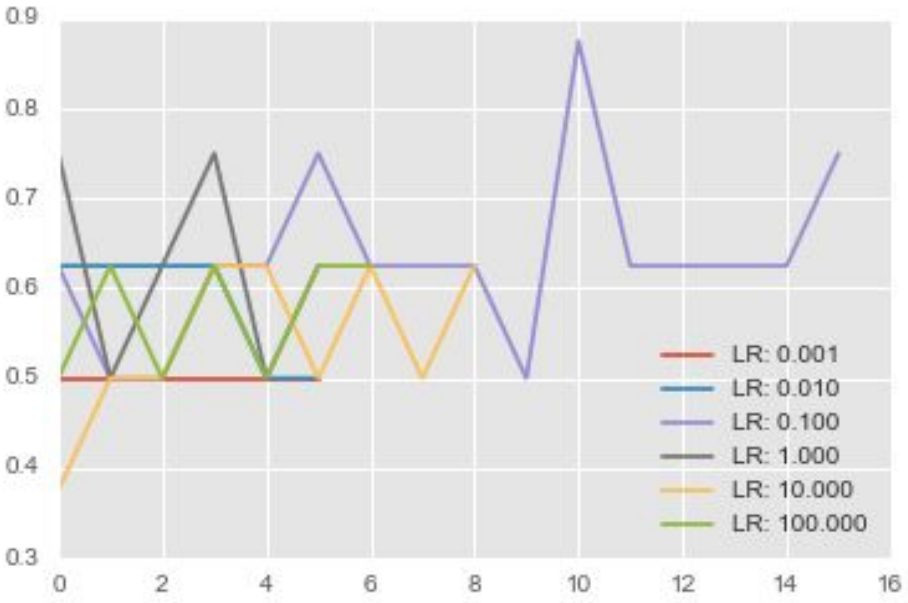
My perceptron has two stopping criteria. The first is when the algorithm runs the designated number of epochs. The second is when the accuracy doesn't improve for 5 consecutive epochs.

During the model selection I tested different learning rates. These were the list [.001, .01, .1, 1, 10, 100]. Since the data is so small it was hard to detect a main pattern. Many times a lucky initialization of the weights would get a correct prediction right away on the separable data set. I ran the list of learning rates 50 times and the biggest thing I noticed was that with the separable dataset a small learning rate like .001 would always be the last to finish or trigger the

early stopping (this can be seen in the figure below). Even though the large rate would bounce around the minimum, the data was easy to fit and it wouldn't be a problem.

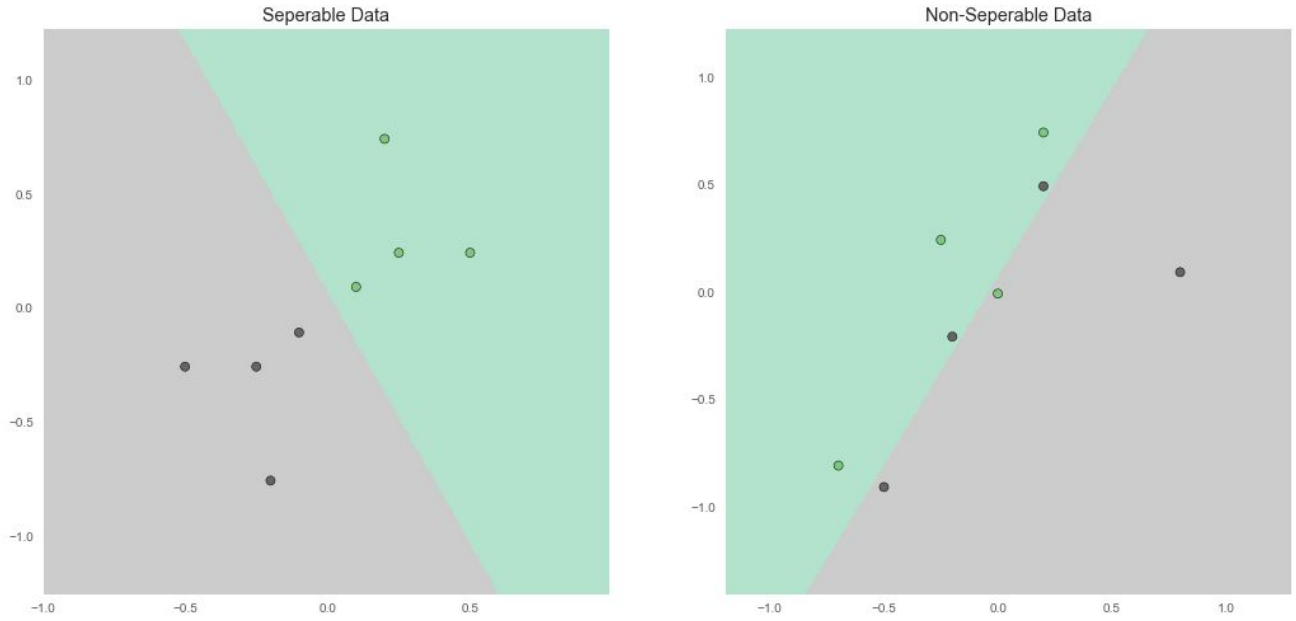
The non separable data was much different. All of the learning rates would terminate early after oscillating accuracy. The higher the rate the more volatile these spikes were.





4. Visualize Results

Using a learning rate of .1 we can see the decision boundary that the perceptron discovers. Within 5 epochs the algorithm was able to come up with a very clear decision boundary for the separable data. For the non-separable data the loss would not converge because it is impossible for the model to learn this data with its current capacity. It is interesting to note that the boundary get close to correctly classifying the other two black points in the green.



5. Voting Classification

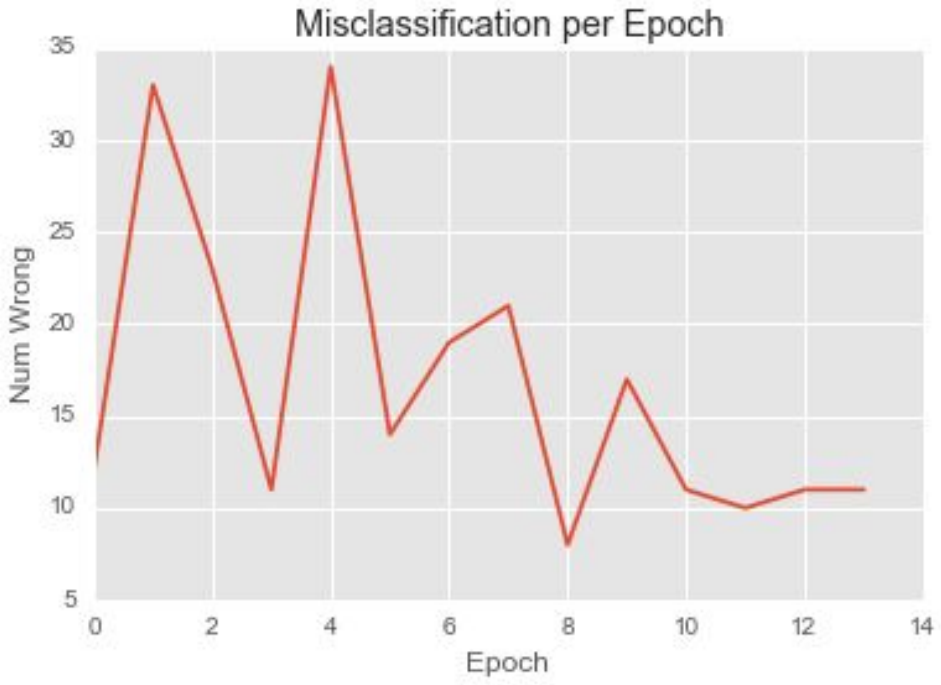
The goal of the voting classification is to predict if a voter is democrat of republican. We are given voting info on topics like immigration, education, and religion in schools.

For 5 trials, with a 7:3 split, the average accuracy for my perceptron was 97%. This was achieved in an average of 12 epochs.

5 Trial Results:

1. 95.96% Epochs: 6
2. 96.27% Epochs: 14
3. 96.89% Epochs: 16
4. 98.44% Epochs: 15
5. 97.51% Epochs: 10

By looking at the weights we are able to make some interesting observations. The two largest magnitude numbers were the weights for the categories SYNFUELS-CORPORATION-CUTBACK and PHYSICIAN-FEE-FREEZE. The smallest were WATER-PROJECT-COST-SHARING and EDUCATION-SPENDING. I was surprised that the best indicators of party were such obscure topics. This leads me to believe that when people don't know how to vote they typically vote with their party.



6. My Experiment

For my experiment I made a multiclass perceptron classifier. I used this to classify flowers in the iris dataset. This classifier makes N independent perceptrons for N classes. To train it I made the labels one for the respective perceptron class and zero for all the other classes. After training each perceptron would perform a binary classification for each class. I would then have each model vote and the one with the highest value from the predict was the output class.

This model was able to get 87.3% percent accuracy. The following is the generated confusion matrix from the data. From this we can see that classifications involving class 1 were the hardest to distinguish.