Ben Clough

Perceptron Lab

Evaluation Data

With the algorithm run on the evaluation data, it ended with a final accuracy of 98.8%. The final weights of the perceptron after 10 epochs were [-3.81, -2.84, -3.07, -1.40, 4.9], corresponding to the four attributes in order and then the bias weight last.

Created Datasets

Both datasets were created in the ARFF format. The setup of these are shown below, with the linearly separable one on the left and the other on the right.

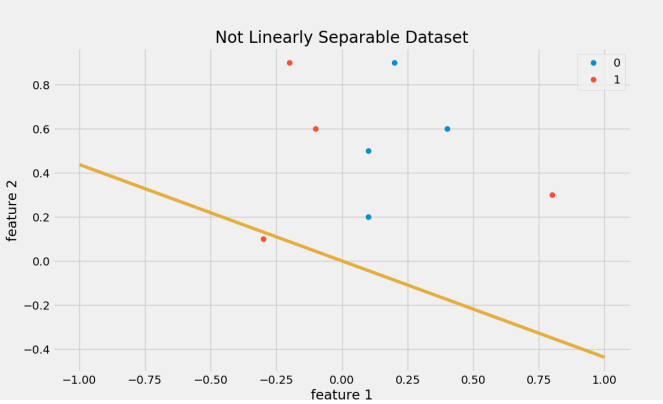
```
@relation linear
@attribute a1 real
@attribute a2 real
@attribute class {0,1}
@data
-0.5, -0.5, 1
-0.3, -0.2, 1
-0.2, -0.4, 1
-0.8, -0.1, 1
0.7, 0.5, 0
0.4, 0.9, 0
0.6, 0.6, 0
0.4, 0.8, 0
```

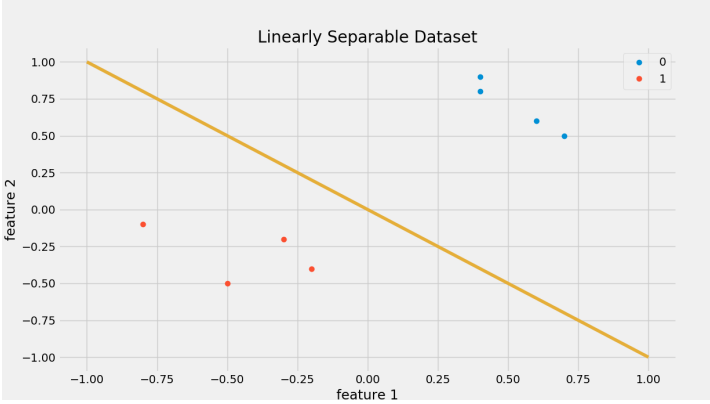
```
@relation nonlinear
@attribute a1 real
@attribute a2 real
@attribute class {0,1}
@data
-0.2, 0.9, 1
-0.3, 0.1, 1
0.8, 0.3, 1
-0.1, 0.6, 1
0.1, 0.5, 0
0.2, 0.9, 0
0.1, 0.2, 0
0.4, 0.6, 0
```

Using different learning rates had different effects on each of these datasets. For the linearly separable set, learning rates of 0.1, 0.0001, and 1 all resulted in the same number of epochs to reach convergence. However, the number of epochs varied greatly when changing the learning rate for the dataset that is not linearly separable. The linearly separable dataset's consistency seems to make sense. It is a simple dataset, and therefore requires few epochs to converge no matter the learning rate. However, it was interesting that any variation to the learning rate resulted in more epochs to converge for the nonlinear one. Setting the learning rate to 0.0001 did have a greater impact than that of 1. This is also logical, as lower learning rates

cause smaller steps. These smaller steps inherently require more steps to reach the same convergence in the end.

With learning rates of 0.1, the following solutions are produced.



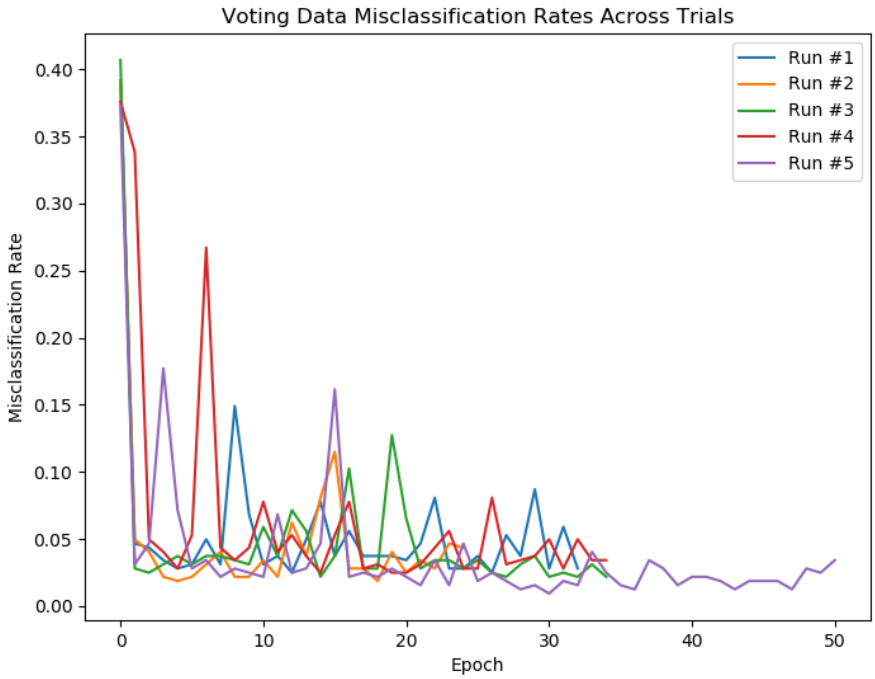


In these situations, and in other cases where the model is not trained deterministically, the stopping criteria is set by a threshold and a chosen number of rounds. When that many rounds in a row result in improvements below the threshold, training finishes and convergence is assumed complete.

Voting Data

With data shuffled each epoch, differing 70-30 splits, and a learning rate of 0.0001, the following table of results was produced for 5 attempts at fitting a perceptron to the voting dataset:

Attempt Number	Final Train Accuracy	Final Test Accuracy	Number of Epochs
1	97.20%	94.96%	32
2	95.65%	89.93%	24
3	97.83%	94.24%	34
4	96.58%	94.24%	34
5	96.58%	91.37%	50



The above graph shows a graphical representation of the misclassification rate for those same 5 runs. The average final training accuracy was 96.77% with an average final test accuracy of 92.95%. These results were found in an average of 35 epochs. In all 5 runs, the weight corresponding to the vote on the physician fee freeze had the largest magnitude and was positive. The weight for the vote on the budget resolution also consistently had a large magnitude, but was always negative. This relates stronger relationships between party affiliation and these votes than other votes, but shows that the two votes tend to indicate opposite parties. The attribute titled "superfund-right-to-sue" often had a very low magnitude, implying that that vote did not have a strong correlation with party.

Scikit-learn Results

Using the scikit-learn implementation of the perceptron on the entire voting dataset, with a learning rate of 0.0001 like the above 5 runs, a similar accuracy of 96.53% was achieved. Raising the learning rate by orders of magnitude slowly decreased the accuracy, while lowering it by an order of magnitude decreased the accuracy more quickly. Thus, it seems that this learning rate was a pretty good fit for the dataset. Additionally, the same 2 weights tended to have larger magnitudes and the same signs as those mentioned in the above section. This validates the functionality of the custom implementation and indicates that scikit-learn does not seem to be doing anything additional and fancy to make their perceptron work. With more parameters, however, greater fine tuning was possible. While utilizing L2 regularization had a

small negative effect on the accuracy, L1 regularization resulted in an increase up to 97.18%, a significant increase at this high level of accuracy.

Using scikit-learn's implementation on the evaluation dataset yielded similar results. By setting a maximum number of epochs to 10, a learning rate of 0.1, and refraining from shuffling the data, the perceptron's ending weights were very similar to those for my implementation as run deterministically above. The accuracy was also nearly the exact same at 98.83%. However, with L1 regularization, data shuffling, a learning rate of 0.001, and no limits on the number of epochs, this implementation achieved an accuracy of 99.05% with much smaller weights. The regularization most likely assisted the model in finding a more optimal solution by encouraging those small weights.

Overall, while similar results were seen, the breadth of customization allowed by the greater number of hyperparameters makes the scikit-learn implementation a clear winner. It is pretty cool that my custom implementation can hold its own against such a well-vetted system, though!