Project 1
Single Layer Perceptron

CS457 - Computational Intelligence
Bryce Harmsen - Eric Engman - Luis Torres - Nisser Aldossary

Introduction

Project 1 is based around the single layer perceptron. The perceptron is implemented twice: both through the WEKA interface and through Python code written by our team. For the reader who is purely seeking our answers to CS 457 Project 1, only the Analysis section should be read. Though for the reader who wishes to see a more complete picture of our process, we have provided a basis and background for our empirical analysis.

Implementation

WEKA

Our first implementation was done in WEKA. This implementation used the built-in Simple Logistic classifier, used for building linear logistic regression binary models. The WEKA and Python implementations shared sample data for the L and I matrix representations (3x3 and 5x5). We ran six different tests on both matrices, one of the tests being on data that the classifier was not trained on. Not all of the tests were necessary (such as cross-validation tests); some were done to see how much the results varied. The datasets for 3x3 were presented in an .arff file format, with nine attributes, one class {I,L}, and eleven instances for training. The 5x5 classifier had 25 attributes, one class {I,L}, and 29 instances for training. The primary two data sets used on the classifiers were the data used to train it and new data it had not seen before. Some of the results may have been skewed based on the quantity of the test data. Specifically, when comparing these two test results to each other the 3x3 matrix had 11 instances of training data, and 5 instances of test data it had not seen before. For the 5x5, the classifier was trained and tested on 29 instances, and only tested 10 unfamiliar instances. This is partly due to the fact there is a limit to the amount of configurations you can provide, as we could not rotate the datasets. Furthermore, the datasets used to train may have confused the classifier. If the training data consisted of I’s and L’s with a height of at least 3, and in the new test data it was given ones with a height of 2 then it may misclassify. This seemed to happen when given the new test data, as it would classify all of the vectors as either I or L but not both nor neither.

Python

Our second implementation is in Python 3. This implementation was built from scratch based on Stephen Marsland’s perceptron algorithm. The implementation allowed for varying resolutions of L and I matrix representations, (i.e. 3x3, 5x5). The perceptron kept track of a single set of weights, one weight for each element in the matrix, with the addition of one extra weight for the bias node. The bias node was set to a value of negative one. The perceptron could be tuned through a yaml file, where the user could set parameters such as the learning rate and the maximum number of iterations allowed during training. The program required that 50% of the sample data was used for training. This training-to-testing ratio requirement was potentially detrimental to the perceptron’s performance. The ability to tune other factors, especially the learning rate, provided increased performance.

Results

WEKA

Below are some of the results using WEKA. Provided are the results for the testing on training data and the unfamiliar data. Other tests were also done (66% Split, 50% Split, and Cross-validation with 5 and 10 folds). The results from the percentage splits and the cross-validation with 10 folds for 3x3 were all 50-60% accuracy, and the cross-validation with 5 folds had 72% accuracy. For 5x5, the split percentage as well as the cross-validation tests were all in the 60-70% range accuracy. The accuracy of the tests using the training data performs better for both sizes (the 3x3 noticeably better than the 5x5 at 100% and 89% respectively, may be due to 3x3 not having enough instances). Another observation is that the tests using unfamiliar data were lopsided for both the 3x3 and 5x5 as both mostly classified one or the other but not both. As mentioned earlier, the data in the new tests usually had a different height than the data it was trained on, which was necessary to be able to make new configurations. This may have influenced or skewed some of the results for the new data.

3x3 Resolution

```
Number of data samples: 11
Training and testing based on 3x3mat.arff and 3x3matTest.arff

Results from testing on the training data:
```

- `Accuracy: 100%`
- `Instances: 11`

```
Results from testing on new test data:
```

- `Accuracy: 60%`
- `Instances: 5`

```
Confusion matrices:
```

a	b	
6	0	a=i
0	5	b=l

Training data tests

a	b	
3	0	a=i
2	0	b=l

Unfamiliar data tests

5x5 Resolution

```
Number of data samples: 29
Training and testing based on 5x5mat.arff and 5x5matTest.arff

Results from testing on the training data:
```

- `Accuracy: 89%`
- `Instances: 29`

```
Results from testing on new test data:
```

- `Accuracy: 40%`
- `Instances: 10`

```
Confusion matrices:
```

a	b	
14	1	a=i
2	12	b=l

Training data tests

a	b	
4	1	a=i
5	0	b=l

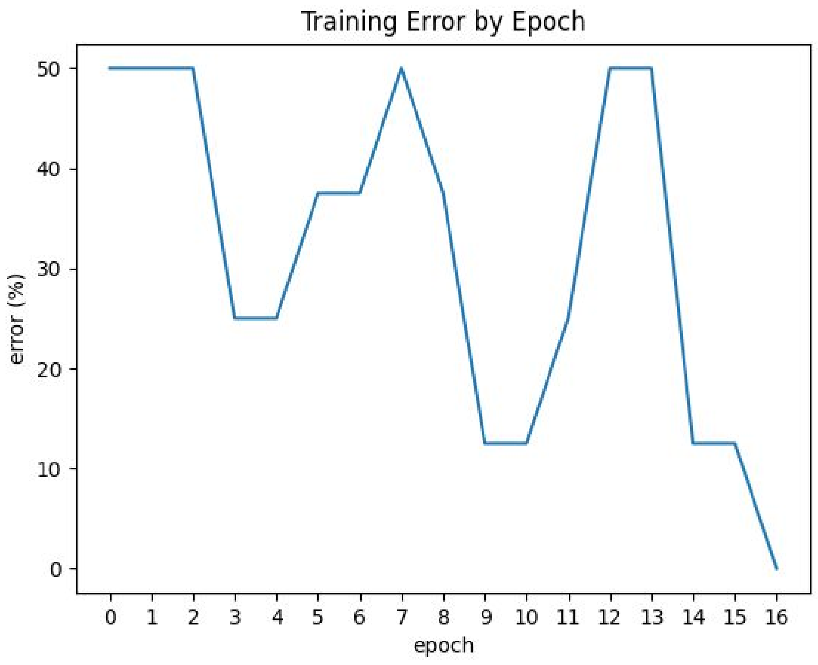
Unfamiliar data tests

Python

Below are the results from a training session using the python code attached with this project. The plots show the percent error at each epoch of the training session. The information below are the training and testing results. The program could exit a training session either because the maximum allowed number of iterations was reached or the outputs from the recall phase matched the targets for each sample of training data.

3x3 Resolution

The 3x3 resolution data set contained a total of 16 samples. These samples were split, 50% for training, 50% for testing. As shown in the plot, the hyperplane shifted dramatically at twice. This may be a reason to lower the learning rate. For this session a learning rate of 0.2 was used. The testing results show a 50% success rate. This is quite poor. This is equivalent to a random selection of weights. The python 3x3 success rates tended to fluctuate between 30% and 60%.



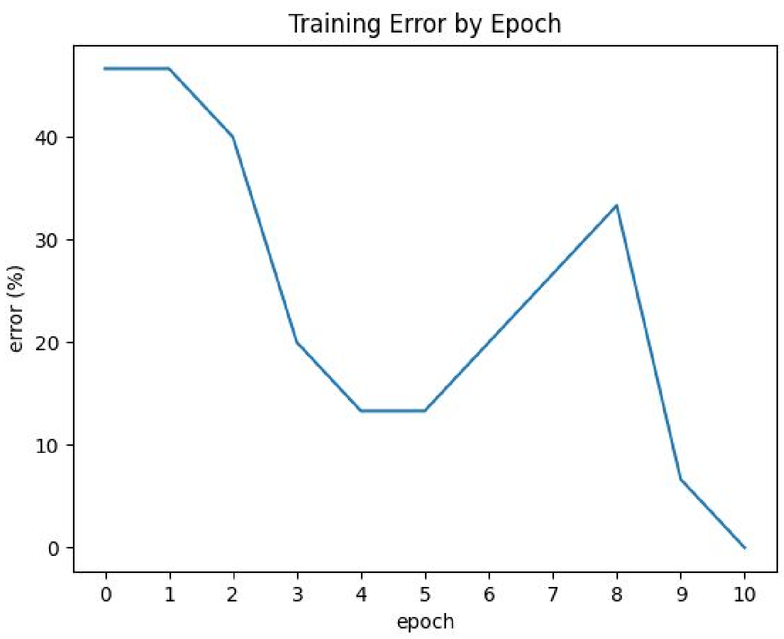
```
Number of data samples: 16
Training and testing based on supervised3by3.csv data

TRAINING:
Exit cause:
     outputs matched targets
```

```
TESTING:
     targets: ['i', 'l', 'i', 'l', 'i', 'l', 'i', 'l']
     outputs: ['l', 'l', 'l', 'l', 'l', 'i', 'i', 'l']
     accuracy: 50%
```

5x5 Resolution

For the 5x5 Resolution, a learning rate of 0.2 was also used. This proved to be a better tuning for this resolution than for the 3x3. There were 29 data samples, again with 50% of the samples used for training. While this particular training session shows a peak error around the eighth epoch, some of training sessions showed a very consistent decline, keeping any progress toward a stable hyperplane throughout each epoch. In general the python test results were consistently better for the 5x5. This session shows an accuracy of 85%. This was the highest accuracy recorded. The lowest was just above 70%.



```
Number of data samples: 29
Training and testing based on supervised5by5.csv data

TRAINING:
Exit cause:
        outputs matched targets
```

```
TESTING:
        targets:
['i', 'l', 'i', 'l', 'i', 'l', 'i', 'l', 'i', 'l', 'i', 'l', 'i',
'l']
        outputs:
['l', 'l', 'i', 'l', 'i', 'l', 'i', 'l', 'i', 'l', 'i', 'l', 'l',
'l']
        accuracy: 85%
```

Analysis

Determining Convergence

To properly determine if our training will always converge, we must determine if the data set is linearly separable. According to the Perceptron Convergence Theorem presented by Marsland, if the data is linearly separable, then training will converge within a finite, determined number of iterations.

So here our determination of convergence is based on whether there is some hyperplane, explained by our weight vector, that can properly separate the training data in class L from the training data in class I. Based on empirical evidence, our determination is yes, there are weight vectors that can be trained to correctly classify all training data. This is shown in the Python implementation by the final state of the outputs after the training occurred, where the prediction error was reduced to zero percent. This can be interpreted as all of the class L data on one side of the hyperplane, and all of the class I data on the other side.

This does not mean that the hyperplane determined in training will appropriately separate all L and I matrix representations correctly, but that for our clean, prepared data set linear separation is possible. Testing is necessary to further determine how well this hyperplane interpolates to other matrix representations of L and I. Testing provides further insight into whether the weight vectors found are useful beyond the training set, and it turns out that resolution may have an effect on the ability of the single layer perceptron to correctly predict targets of data outside the training set.

Effect of Resolution

For the Python data, the 3x3 matrix gravitated at a lower accuracy, falling around 60% on average and the occasional less than 40% success. However, adding noise to a couple cases did increase the testing accuracy to around 80% and a low of 60% accuracy.

By expanding the matrix to a 5x5 the test results shot up and stuck around 85-92% accuracy on average with the rare 78% drop. However with noise, we saw a dip in accuracy around 71-85%.

Based on these results, I believe that increasing the matrix size (resolution) would give more accurate readings. It is worth noting that the noise was put in randomly, so maybe by adding well-crafted noise in the training sets, it would help increase the 5x5 accuracy in testing as well.

Works Cited

Marsland, Stephen. Machine Learning: an Algorithmic Perspective. CRC Press, Second Edition, 2015.