Assignment 1 - Neural network

Liverpool University
Robert Johnson
200962268

Contents

0.1 Introduction

This assignment revolved around creating a piece of software that had the capabilities to take 3 classes and create binary networks to distinguish between 2 separate class's, additionally we should have some way to distinguish between the 3 classes using 3 separate binary perceptron's.

0.2 Question 1: Explain the Perceptron algorithm for the binary classification case, providing its pseudo code (20)

0.2.1 Training

```
1  perceptTrain(MaxIter, Data[r1[0,1,2,..,f+1],r..[0,1,2,..,f,f+1],rn[0,1,2,..,f,f+1]])
2      weights[0,1,..,f] = 0
3      bias = 0
4      for(int i = 0; i < MaxIter; i++)
5          for(int j = 0; j < Data.length; j++)
6              a = (∑_{d=0}^{f} weights[d]*Data[j][d])+bias
7              if(Data[j][f+1]*a≤0)
8                  weights[d] = weights[d]+Data[j][f+1]*Data[j][d]          ∀ d = 0,..,f
9                  bias = bias + Data[j][f+1]
10     return weights[0,1,2...f], bias
```

Figure 0.1: Training pseudo code

The method for training the neural network takes

- The number of times to repeatedly process our Data as MaxIter
- The Data to update the weights of the network as Data

2. The weights for each feature in our dataset are then assigned a value of 0 to initialize the network.
3. The bias is then assigned a value of 0.
4. We then repeatedly flush our data through our neural network trainer maxIter times.
5. We increment I up to how many records we have
6. We assign the value of our weights multiplied by our datas features to a (our activation value)
7. If the value of Data[j][f+1] (if the current data is what we are searching for Data[j][f+1] = 1, else Data[j][f+1] = -1) multiplied by the current activation value is less than or equal to 0 then our perceptron has failed to activate and we need to alter our weight values
8. For each weight value we move the weight values so that they decrease the value of a (if the current data is not what we are looking for) or increase it (if it is what we were looking for).
9. We then increase our bias, to move our activation value by a reduced amount.
10. We then output our found weights and bias

Brief explanation

During training the algorithm takes a set of training data given in the form of records as well as class identifiers, using these values our aim is to find a vector that separates any two of these class's to do this we use a weight vector which describes how important each feature is to a given class. This can be used to work out an activation value which is either positive in which case our class is at one side of the vector else negative in which case it is on the other side of the vector.

To create this weight vector we push records through our neural network taking note if we classify a record in which case we update our weights such that the same record would be correctly identified if we were to push it through our system again this repeatedly happens until either we stop iterating through our data or we reach a point in which no miss-classifications are made.

0.2.2 Testing

```
1  perceptTest(weights[0,1,2...f], Data[0,1,2...f], bias)
2      a = (∑_{d=0}^{f} weights[d]*Data[d]) + bias
3          return sign(a)
```

Figure 0.2: Testing pseudo code

The method for testing the neural network takes

- Weights from our training algorithm
- A single record to be predicted
- A bias value from our training algorithm

2. We calculate our activation value by multiplying our training weights by our feature values and add our bias value.
3. We return the sign of our activation value, if it is above 1 then the record we are predicting is indeed the type of data the neural network was trained to find else it is something else.

Brief explanation

As explained previously during training we calculate a weight matrix which describes our vector separating two classes this makes testing very easy we simply take our data that needs to be classified and multiply our weights and features together, if we return a positive activation we have classified the item to be what the network was trained to identify else we have classified the opposite class.

0.3 Question 2: Implement a binary Perceptron(30 marks)

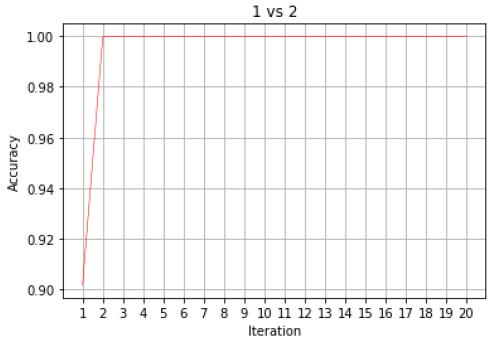
Below is the binary perceptron's code which was implemented in python and included with this document.

```
def perceptTrainOneVsOne(regCoef, maxIter, Data, ClassValue, ClassValue2, shuffleVal):
    weights = np.array([0,0,0,0])
    bias = 0
    for i in range(0, maxIter):
        if shuffleVal == 1: shuffle(Data)
        for features, classEx in Data:
            if(classEx == ClassValue or classEx == ClassValue2):
                tempSetNum = 1 if classEx==ClassValue else -1
                a = np.dot(weights,features) + bias
                if(tempSetNum*a<=0):
                    weights = np.subtract(np.add(weights,np.multiply(tempSetNum,features)),np.multiply(2*regCoef,(weights)))
                    bias += tempSetNum
    return weights,bias
```

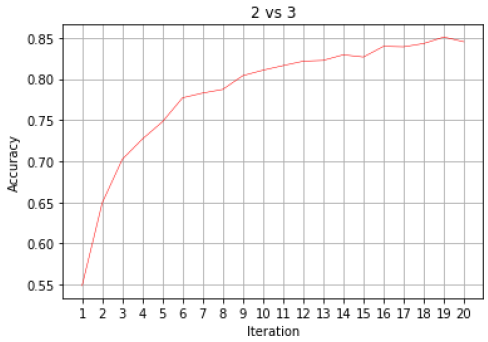
Figure 0.3: Binary Perceptron

0.4 Question 3: Use the binary Perceptron to train classifiers to discriminate between (a) class 1 and class 2, (b) class 2 and class 3 and (c) class 1 and class 3. Report the train and test classification accuracies for each of the three classifiers after 20 iterations. Which pair of classes is the most difficult to separate? (20 marks)

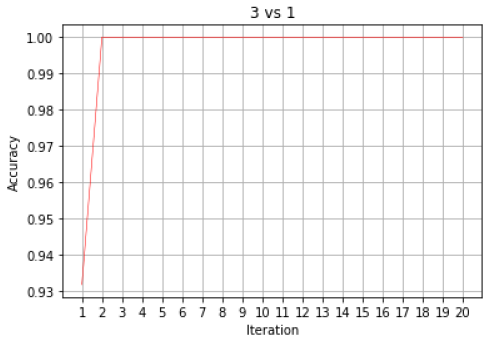
0.4.1 Train accuracies



(a) 1vs2



(b) 2vs3



(c) 3vs1

We can see from the Figure 0.4a that during training our classifier used to discriminate between class 1 and 2 took only 1 iteration to guarantee correct classification of our training dataset, our classifier used to discriminate between class 3 and 1 in Figure 0.4b also shown the same sort of trend. However our classifier for class 2 and 3 in Figure 0.4c seemed to struggle on the training only correctly classifying the data 85% of the time during training at 20 iterations.

0.4.2 Test accuracies

1 vs 2

Accuracy	0.996
Precision	0.992063492
Recall	1
falsePositiveRate	0.008
fscore	0.996015936

	Actually 1	Actually 2
Predicted 1	2000	16
Predicted 2	0	1984

Weights : [1.50, 4.30, -6.70, -3.60]
Bias : 1

2 vs 3

Accuracy	0.89125
Precision	0.886038481
Recall	0.898
falsePositiveRate	0.1155
fscore	0.891979141

	Actually 2	Actually 3
Predicted 2	1796	231
Predicted 3	204	1769

Weights : [38.90, 36.60, -60.30, -49.60]
Bias : 23

3 vs 1

Accuracy	1
Precision	1
Recall	1
falsePositiveRate	0
fscore	1

	Actually 3	Actually 1
Predicted 3	2000	0
Predicted 1	0	2000

Weights : [1.30, 5.40, -10.30, -5.60]
Bias : 1

We can see from the tables shown above that our accuracy follows on from our training data we almost correctly classified class 1vs2 100% of the time as did classify class 3vs1 100% of the time, however again our classifier 2vs3 struggles to classify correctly even 90% of the time which is a slight increase in performance than what we got in our training which only got an accuracy of 85%.

Because the training of the classifier for discriminating class 2 and 3 only received an accuracy of 85% compared to its competitors with 100% and during testing only got 90% compared to its competing 100% from the other 2 classifiers I have to say that class 2 and 3 are the hardest classes to separate.

0.5 Question 4: For the classifier (class 1 and 2) implemented in part (3) above, which feature is the most discriminative?(5 marks)

The weights returned from training were [1.5, 4.30, -6.70, -3.60].
To determine the most discriminative feature we simply need to look at what weight took the highest absolute value as this represents how important it was to either of the classes i.e the most discriminative. In the above case that would be feature 3 with an absolute weight of 6.7 followed by feature 2 with a weight of 4.3 and finally followed by feature 4 and then 1.

0.6 Question 5: Extend the binary Perceptron that you implemented in part (2) above to perform multi-class classification using the 1-vs-rest approach. Report the train and test classification accuracies for each of the three classes after training for 20 iterations.(15 marks)

0.6.1 Testing accuracy

Classification 1 vs all

	Actually 1	Actually 2	Actually 3
Predicted 1	1855	788	41
Predicted 2	145	973	278
Predicted 3	0	239	1681

Weights of 1 vs all = [1.70, 4.90, -7.90, -3.20] Bias = 1
Weights of 2 vs all = [9.20, -44.50, -2.60, -24.30] Bias = 20
Weights of 3 vs all = [-45.40, -35.40, 62.40, 46.00] Bias = -25

Looking at the above table we can see that the overall accuracy can be easily worked out by simply taking the classes we correctly predicted and dividing it by the population of our data however describing the accuracy of our individual class proves slightly more difficult due to the fact that a record is classified using the combination of each classifier because of this I have determined that the equation to work out the accuracy of each class should be:

$$Accuracy_1 = \frac{P_1A_1+P_{\neg 1}A_{\neg 1}}{P_1A_{\neg 1}+P_{\neg 1}A_1+P_1A_1+P_{\neg 1}A_{\neg 1}}$$

$$Accuracy_2 = \frac{P_2A_2+P_{\neg 2}A_{\neg 2}}{P_2A_{\neg 2}+P_{\neg 2}A_2+P_2A_2+P_{\neg 2}A_{\neg 2}}$$

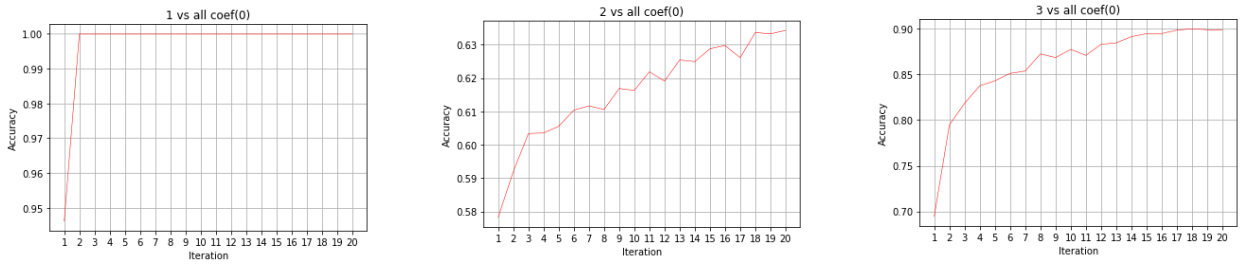
$$Accuracy_3 = \frac{P_3A_3+P_{\neg 3}A_{\neg 3}}{P_3A_{\neg 3}+P_{\neg 3}A_3+P_3A_3+P_{\neg 3}A_{\neg 3}}$$

This results in the following accuracy table:

Overall accuracy of predictions	0.7515
Accuracy of predicting class 1	0.837666667
Accuracy of predicting class 2	0.758333333
Accuracy of predicting class 3	0.907

We can see that our overall accuracy is much lower than all of our binary classifiers this is due to trying to separate one class vs all others making it harder to create a perfect linear vertex. Again it seems as though the hardest class to classify is class 2 as it has the lowest accuracy.

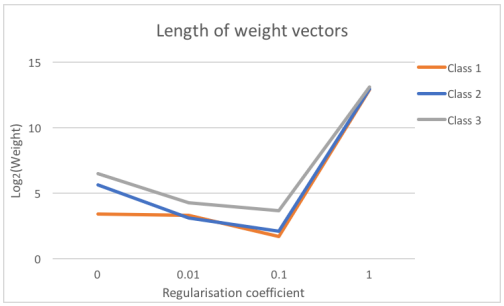
0.6.2 Training accuracy



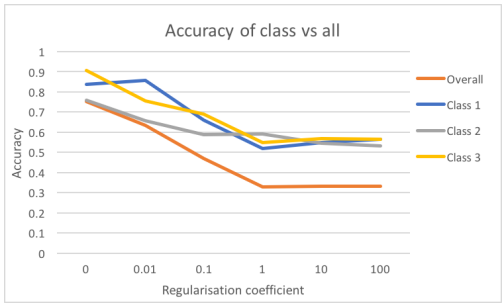
We can see from the above graphs that class 1 was the most successful during training only requiring 1 iteration to classify all instances correctly however class 2 and 3 seem to take much longer, class 2 only reaching a accuracy of 64% after 20 iterations and class 3 only reaching 90% after 20 iterations.

0.7 Question 6: Add an l_2 regularisation term to your multi-class classifier implemented in question (5). Set the regularisation coefficient to 0.01, 0.1, 1.0, 10.0, 100.0 and compare the train and test classification accuracy for each of the three classes

0.7.1 Testing accuracy



(a) Length Of Vectors



(b) Accuracy 1 vs all

As you can see from Figure 0.5a the x axis ends at 1, this was due to the length of the vectors being too large actually being classed as inf by python. Because of this i have removed them from points on the table.

Given enough iterations we would assume that a regularisation coefficient that is greater than 1 would also tend to infinity to show this take our update term.
$W_{t+1} = W_t + yx - 2\lambda W_t$
If we assume that λ is greater than 1 we can also state that
$|-2\lambda W_t| > |2W_t|$
Because of this we know that if yx is not large enough to cover the difference between these two values we will increase our weights due to the coefficient. If this happens even once the chance that it will be fixed in later cycles drops significantly as $-2\lambda W_t$ depends on W_{t-1}.

Because of this it is unlikely that any regularisation coefficient greater than 1 will cause a positive effect on weights this is backed up by Figure 1 which shows that our accuracy does not increase substantially when we increase our coefficient past 1.
We can see this in the overall accuracy of the network which drops to 33% the same as picking a class at random for a specific record. Anything less than or equal to 1 has a chance to increase our

accuracy however this was only the case for class 1 whos accuracy improved at a regularisation of 0.01 this was due to reducing the amount of misclassification's of class 1 rather than increasing the amount of classifications of class 1.

However all other classs accuracys dropped the main reason for this would be due to the lower weights given to each classifier making it harder for any of them to get the optimal vector to separate each class this restriction makes it harder for the classifier to fully fit our dataset if we were to decrease this regularisation coefficient it is likely that we could reduce weights from having a regularisation of 0 as well as increase the accuracy given by our system.This is shown by the table

below which shows results when using a regularisation of 0.000001.

Overall accuracy of predictions	0.766666667
Accuracy of predicting class 1	0.839833333
Accuracy of predicting class 2	0.776833333
Accuracy of predicting class 3	0.916666667

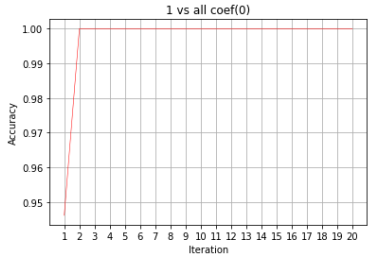
	Actually 1	Actually 2	Actually 3
Predicted 1	1843	743	61
Predicted 2	157	1055	237
Predicted 3	0	202	1702

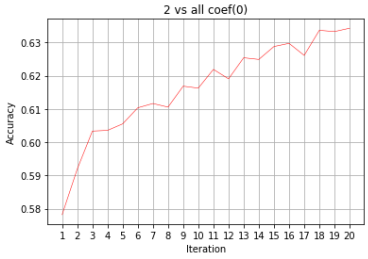
Weights of 1 vs all = [1.65, 5.04, -7.29, -3.24] Bias = 1
Weights of 2 vs all = [-4.34, -8.23, -3.92, -4.29] Bias = 3
Weights of 3 vs all = [-5.02, -3.43, 12.91, 9.82] Bias = -36

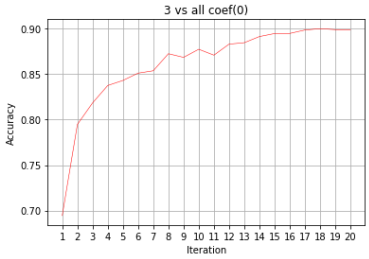
We can see from the weights that they are still considerably lower than when we had a coefficient of 0, we also see that our accuracy is very slightly better then when we had a regularisation of 0. This followed from our prediction.

0.7.2 Training accuracy graphs

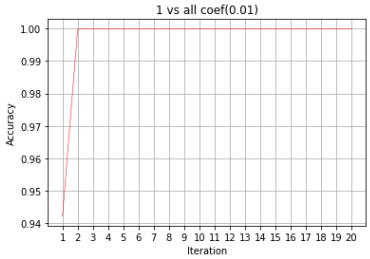
Regularisation:0

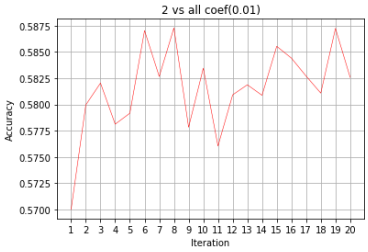


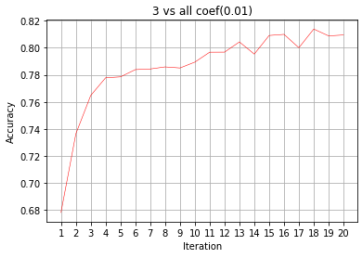




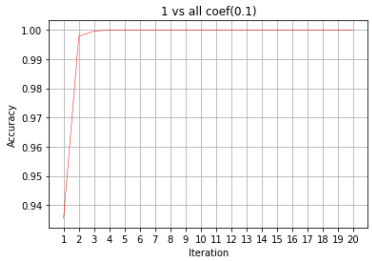
Regularisation:0.01

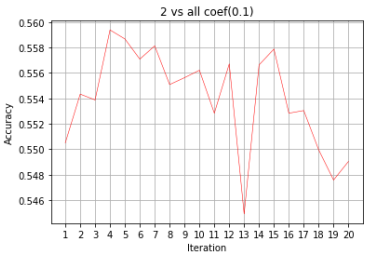


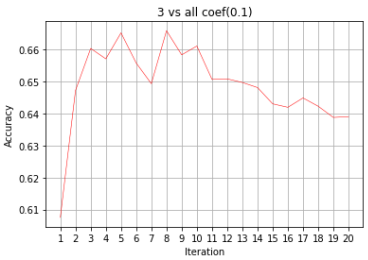




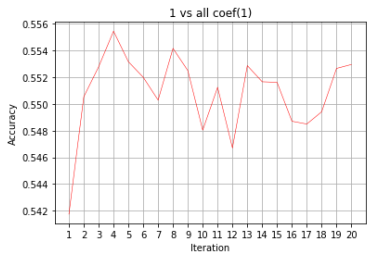
Regularisation:0.1

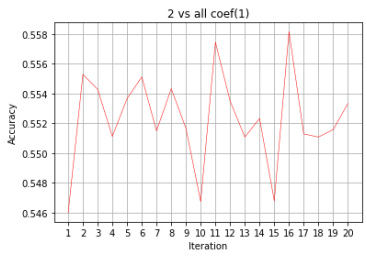


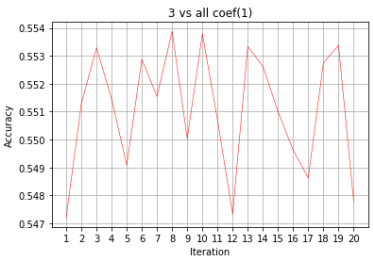




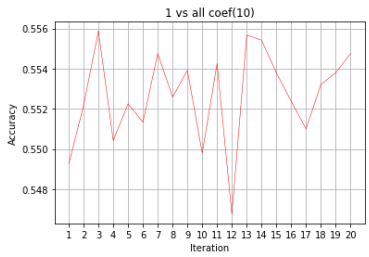
Regularisation:1

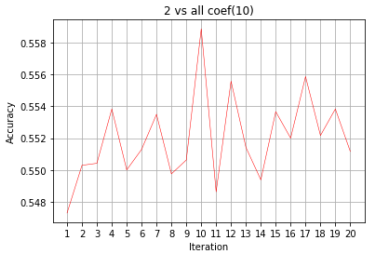


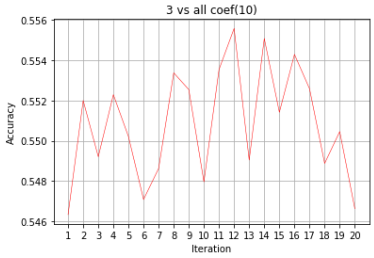




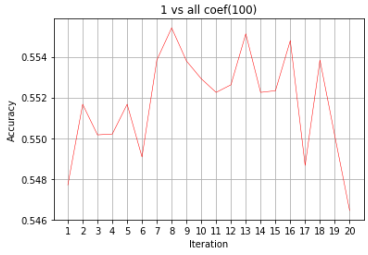
Regularisation:10

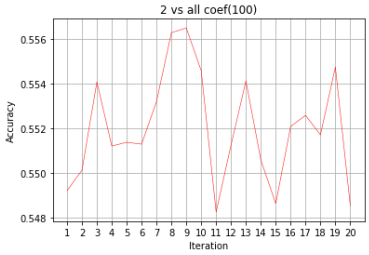


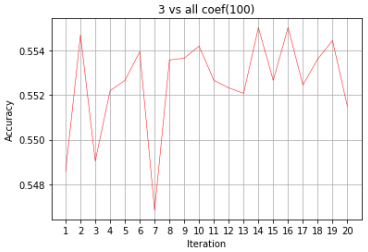




Regularisation:100







0.7.3 Review of training graphs

We can see from the above graphs that class 1 vs all deals the best with increasing regularisation still only taking 1 iteration to classify all test data correctly up until a coefficient of 0.1, however class 2 seems to become much more irregular in terms of accuracy at each iteration at only a regularisation of 0.01 and finally class 3 seems to become irregular at a regularisation of 0.1 actually decreasing its accuracy with every iteration past 5. After 1 as mentioned previously all training / testing becomes very inaccurate and very random from one iteration to the next.