COMP527 - Assignment 1 - Data Classification

Rob Lamprell - 201459448

Due Date: 19th March 2021

Contents

List of Figures

Question 1

(20 marks) Explain the Perceptron algorithm for the binary classification case, providing its pseudo code.

Binary Perceptron

A binary perceptron is a linear classification algorithm model which in essence draws a line/hyperplane between plotted sets of features in order to make a best guess as to which class they belong to.

The perceptron is a model of a singular neuron, it's bio-inspired and learns based on its exposure to stimuli. It works by creating a set of zero valued start weights, which are equal to the number of features we wish to train on. We then take the sum product of these values and add a bias (b). This produces an activation score [1]:

$$\begin{aligned} a &= w_1x_1 + w_2x_2 + ... + w_ix_i + b \\ &= b + \sum_{i=1}^{n} w_ix_i \\ &= b + \overline{W}^T\overline{X} \qquad [1] \end{aligned}$$

Which can then be compared to a predefined threshold θ, to determine the output of the neuron when it fires. In the case of our Binary Perceptron this activation score produces either +1 or -1 (positive class or negative class) [1].

$$Prediction = \begin{cases} +1 & \text{if } a > 0, \\ -1 & \text{if } a \leq 0 \end{cases}$$

This prediction is then compared to the actual classification of the feature. If they match, nothing is done. If they do not match, the weights and bias are updated - in the hope that next time they will correctly predict the class. The weight update rules are as follows (excluding a learning rate, as it is not mentioned in the assignment or lecture notes) [1]:

$$\begin{aligned} w_i &= w_i + y * x_i \qquad \text{for } i = 1, ..., n \\ b &= b + y \qquad [1] \end{aligned}$$

This process is repeated for the number of iterations selected [1, 2].

Additional Remarks

It's worth noting that a bias, is essentially another weight but, does not pair with an input. The bias can be absorbed into the activation formula without explicit mention [1]. Which transforms the formula into:

$$\begin{aligned} a &= \sum_{i=0}^{n} w_ix_i \\ &= \overline{W}^T\overline{X} \qquad [1] \end{aligned}$$

Additionally, to attain a zero valued threshold θ, we could make our bias negative-θ. This would mean that our threshold is zero [1].

Binary Perceptron Pseudo-Code

Below is the pseudo-code necessary implementing a binary perceptron developed through reference of the lecture notes [1]. Note:

- X = input features
- y = input labels
- a learning rate of 1 is assumed, hence it has been omitted from the **Train** method - as it will not affect the weight calculations

Train(Training Data: X, y)

```
w_i = 0, for i=1,...,n          ▷ set weights to 0 where the number of weights is the count of features
b = 0                           ▷ set bias to 0 (the bias is not paired with an input)

for i=0 to maxIteration do
    shuffle()                   ▷ Randomly shuffle the ordering of the data's feature and label pairs

    for each row within the data do
        prediction = Predict(X) ▷ make a prediction

        if y * prediction <= 0 then   ▷ if prediction does not match the actual label
            w_i = w_i + y * x_i, for i=1,...,n   ▷ update the weights
            b = b+y                              ▷ update the bias
        end if
    end for
end for

return b, w_1,... ,w_n          ▷ return all weights, including the bias
```

Predict(Features: X)

```
a = W^T X + b                   ▷ activationscore = Weights^T Inputs + bias
if activation > 0 then          ▷ If greater than zero then predict the first (positive) class
    prediction = +1
else                            ▷ else predict the second (negative) class
    prediction = -1
end if

return prediction               ▷ the predicted class / sign (+/-)
```

Shuffle(Training Data: X, y)

```
Make a list of indices from X or y
Randomly order the indices
Apply the new indices ordering to X and y

return X, y
```

Question 2

(20 marks) Implement a binary perceptron.

See class **Perceptron()** within the attached file, **Assignment_1.py**

Note: the actual implementation is slightly different from the pseudo-code above. This is due to additions and alterations made to complete further questions and also to aid in exploration and understanding of the model.

Question 3

(20 marks) Use the binary perceptron to train classifiers to discriminate between:

- (1) class 1 and class 2
- (2) class 2 and class 3
- (3) class 1 and class 3

Report the train and test classification accuracies for each of the three classifiers after training for 20 iterations. Which pair of classes is most difficult to separate?

Results

Below are three sample outputs produced by the code from q3():

```
==========================================
Beginning Question 3_1 -- class-1 vs class-2
------------------------------------------
Training accuracies:
[0.8875, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.
0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.
0, 1.0, 1.0]
------------------------------------------
Testing accuracy = 1.0
==========================================



==========================================
Beginning Question 3_2 -- class-2 vs class-3
------------------------------------------
Training accuracies:
[0.625, 0.5125, 0.7875, 0.75, 0.6125, 0.6375,
0.825, 0.8125, 0.7375, 0.725, 0.6875, 0.8125,
0.8625, 0.875, 0.8, 0.8375, 0.8125, 0.725, 0.7
875, 0.85]
------------------------------------------
Testing accuracy = 0.95
==========================================



==========================================
Beginning Question 3_3 -- class-1 vs class-3
------------------------------------------
Training accuracies:
[0.9125, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.
0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.
0, 1.0, 1.0]
------------------------------------------
Testing accuracy = 1.0
==========================================
```

```
==========================================
Beginning Question 3_1 -- class-1 vs class-2
------------------------------------------
Training accuracies:
[0.8875, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.
0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.
0, 1.0, 1.0]
------------------------------------------
Testing accuracy = 1.0
==========================================



==========================================
Beginning Question 3_2 -- class-2 vs class-3
------------------------------------------
Training accuracies:
[0.5625, 0.625, 0.7, 0.625, 0.6625, 0.85, 0.77
5, 0.7375, 0.7875, 0.7875, 0.7625, 0.7125, 0.7
75, 0.9875, 0.9375, 0.8125, 0.825, 0.825, 0.87
5, 0.875]
------------------------------------------
Testing accuracy = 0.85
==========================================



==========================================
Beginning Question 3_3 -- class-1 vs class-3
------------------------------------------
Training accuracies:
[0.9125, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.
0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.
0, 1.0, 1.0]
------------------------------------------
Testing accuracy = 1.0
==========================================
```

```
==========================================
Beginning Question 3_1 -- class-1 vs class-2
------------------------------------------
Training accuracies:
[0.8625, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.
0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.
0, 1.0, 1.0]
------------------------------------------
Testing accuracy = 1.0
==========================================



==========================================
Beginning Question 3_2 -- class-2 vs class-3
------------------------------------------
Training accuracies:
[0.6375, 0.5, 0.5625, 0.8125, 0.725, 0.5875,
0.8, 0.75, 0.7875, 0.875, 0.775, 0.7125, 0.85,
0.825, 0.7875, 0.8125, 0.7875, 0.8875, 0.825,
0.9]
------------------------------------------
Testing accuracy = 0.6
==========================================



==========================================
Beginning Question 3_3 -- class-1 vs class-3
------------------------------------------
Training accuracies:
[0.9125, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.
0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.
0, 1.0, 1.0]
------------------------------------------
Testing accuracy = 1.0
==========================================
```

Figure 1: Outputs from q3() in Assignment_1.py - each column represents a separate run.

From the images above, it seems clear that within 20 iterations the algorithm is able to quickly distinguish between two classes when comparing either:

- (1) class-1 and class-2 or
- (3) class-1 and class-3

Within the limited sample, it can be seen that after the first epoch, there are no mismatched classifications for the remaining 19 within the training set. Additionally, in all three instances the models are able to accurately predict the classifications within the test data.

It looks to be far more difficult for the algorithm to distinguish between (2) class-2 and class-3. Even by the end of the 20th iteration, the prediction accuracy is still not 100% - ending somewhere between 80% and 90% in the three examples given. This difficulty is further evidenced when applying the testing data. Although it was capable of producing high results (95% and 85% in two of the instances above), it is also able to produce results of much lower accuracy - such as 60% as the third example demonstrates.

Potentially, class-2's and class-3's features are distinct from class-1, making it possible to reliably differentiate between the pairs. However, their features are not sufficiently distinct from one another's and so they cannot be distinguished consistently when utilising the binary classification algorithm - at least under the current parameters.

Note: The sample size of the above is very small. While testing, there have been observable instances of (1) class-1 vs class-2 only attaining an accuracy of 95% on the testing data. Similarly, (3) class-2 vs class-3 has produced models which achieve 100% accuracy on the test data.

Question 4

(20 marks) Extend the binary perceptron that you implemented in part 3 above to perform multi-class classification using the 1-vs-rest approach. Report the train and test classification accuracies after training for 20 iterations.

The One Vs Rest approach of the binary perceptron is effectively a collection of individually trained binary classifiers which can be used to create a set of confidence scores, of which the max value should be taken, to establish a best guess - the maximum activation score.

Results

Below is a sample output of from Assignment_Questions().q4() within Assignmnet_1.py:

```
============================================
Beginning Question 4 One vs Rest
--------------------------------------------
Training accuracy (class-1) = [0.91, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0]
--------------------------------------------
Training accuracy (class-2) = [0.62, 0.61, 0.55, 0.6, 0.65,
0.58, 0.64, 0.52, 0.6, 0.61, 0.62, 0.62, 0.63, 0.6, 0.63, 0.5
7, 0.62, 0.66, 0.55, 0.63]
--------------------------------------------
Training accuracy (class-3) = [0.75, 0.69, 0.82, 0.82, 0.78,
0.93, 0.82, 0.84, 0.84, 0.78, 0.87, 0.93, 0.88, 0.95, 0.89,
0.87, 0.85, 0.88, 0.92, 0.91]
--------------------------------------------
Testing accuracy =  0.67 (2dp)
============================================
```

Figure 2: Sample training and testing accuracies output for q4()

When observing the training accuracies, a similar pattern to question three is visible. The Multi-classifier:

- confidently differentiates class-1 from class-2/class-3 after little training
- struggles to establish a set of weights to reliably distinguish class-2 from class-1/class-3
- is able to build a somewhat reliable model to differentiate between class-3 and class-1/class-2

What is interesting, is that class-3 vs Rest performs significantly better than class-2 vs Rest. Potentially this means that, in addition to class-3's and class-2's feature's being similar, class-3's features are more distinct from class-1's (than class-2's).

Below is a table which represents accuracy figures recorded across a sampling of one-hundred runs of q4():

min	max	avg
33%	100%	73%

Table 1: Min, Max and Average accuracies observed from testing
100 separate runs of One Vs Rest (Multiclassifier).

Over the 100 runs, the One vs Many classifier produces an average of 73%, clearly it is learning to some extent as the accuracy is far higher than random. However, accuracies on the models created range from 33% to 100%. That is to say, sometimes after training, the model can correctly predict every single class in the test data. However, sometimes, it is also only as good as random - with a one in three chance of being correct.

After experimenting with positioning in the code, it appears as though the shuffling order plays a large part in the accuracy of outcomes. The distribution of these accuracies from every unique shuffling order is beyond the scope of this assignment.

Question 5

(20 marks) Add an l_2 regularisation term to your multi-class classifier implemented in part 4. Set the regularisation coefficient to 0.01, 0.1, 1.0, 10.0, 100.0 and compare the train and test classification accuracies.

Regularisation

This question mandates that we implement Ridge Regression. Using the lecture notes we know that the weight update rule for the Perceptron under L2 regularisation becomes:

$$\begin{aligned}\overline{W} &= \overline{W} - \mu(-y * \overline{X}_i + 2\lambda\overline{W}) \\ &= (1 - 2\lambda) * \overline{W} + y_i + \overline{X}_i \qquad \text{(where } \mu = 1\text{)[1]}\end{aligned}$$

Here λ is our regularisation coefficient and can be altered to affect the predictions our models make - these typically take on the form of logarithmic scale values. In-order to find a best fit a user will need to either experiment with values (as we do below) or utilise cross-validation methods, such as k-fold cross-validation [1, 3]. Note that Ridge Regression within the Perceptron algorithm does not influence the bias, and so the calculation remains the same:

$$b = b + y \qquad [1]$$

Common practice when training the model is to split the training data into a training set and validation set [1, 4]. However, that has not been the approach taken in this assignment.

Results

Using the values provided in the question λ = [0.01, 0.1, 1.0, 10.0, 100.0] we obtain these results:

λ	min	max	avg
10^{-2}	33%	100%	60%
10^{-1}	33%	67%	41%
10^{0}	33%	33%	33%
10^{1}	33%	33%	33%
10^{2}	33%	33%	33%

Table 2: Min, Max and Average accuracies observed from testing
100 separate iterations using Ridge Regression Described in Q5

It's clear that all coefficient values (λs) provided in question five perform worse on the test data than the model implemented in question four. There is a strong correlation between lower coefficient values and higher accuracy on test data. Anything at above 10^0 appears to be unable to correctly classify any of the labels correctly and is no better than random.

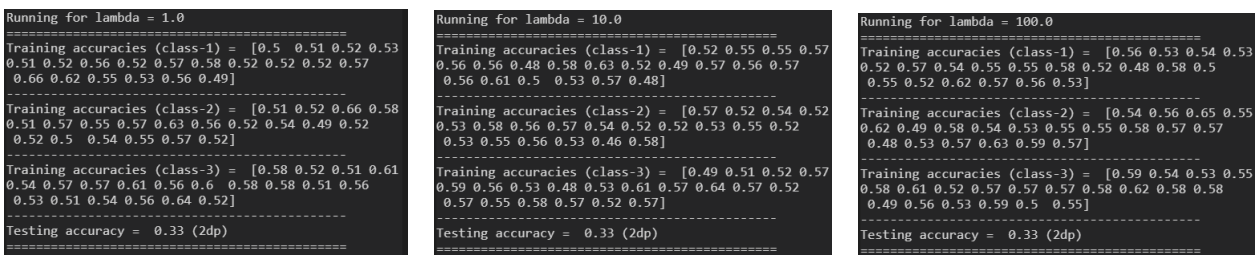

Figure 3: Outputs from q5() - each column represents a different λ value.

From the 20 training iterations, displayed in Figure-3, it becomes apparent as to why the test accuracy is so low. None of the binary classification models are able to achieve an accuracy much above 50%.

At 10^{-1} the model's average accuracy is only slightly higher at 41%, however, it's maximum is 67% - it is able to sometimes correctly classify two labels correctly across the dataset. This is still not a desirable result, as there are many errors. However, it makes sense when observing the accuracy outputs from the training data in Figure-4:

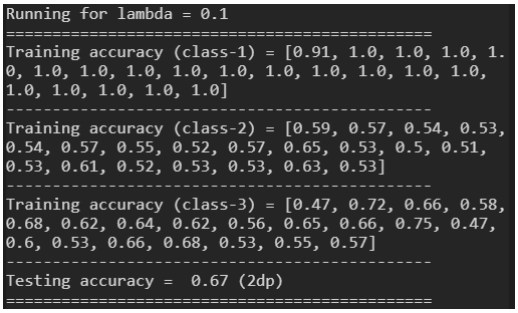

Figure 4: Sample output from q5(), where $\lambda = 0.1$

Class-1 is now able to be reliably distinguished from class-2 or class-3. Therefore, the model effectively now knows a third of the dataset, it knows what is class-1 and what is not. Additionally, on anything not class-1 it can now guess between class-2 and class-3, instead of all three.

Using 10^{-2} (the lowest provided in the question), the model's max is able to achieve 100% accuracy on the test data. However, when comparing the averages, it still does not mirror the performance seen from the implementation in question 4. 60% is substantially lower than 73%. The training accuracies of class-2 and class-3 still lag behind.

Out of curiosity, I also ran a coefficient value of 10^{-10}. This was able to produce results similar to that of question 4 - a 1% lower average:

λ	min	max	avg
10^{-10}	33%	100%	72%
10^{-2}	33%	100%	60%
10^{-1}	33%	67%	41%
10^{0}	33%	33%	33%
10^{1}	33%	33%	33%
10^{2}	33%	33%	33%

Table 3: Min, Max and Average accuracies observed from testing
100 separate iterations using Ridge Regression - including $\lambda = 10^{-10}$

Considering that:

- ridge regression is used to prevent over-fitting by further generalising a model [4]
- our training data set is fairly small and seems to be reasonably accurate when predicting on the testing data
- with the model already struggling to identify between class-2 and class-3 and class-2 vs Rest, generalising the model only exacerbates the issue

Potentially this means, that the testing and training sets are not a good fit for ridge regression. As with anything other than very small coefficient values, as shown in Table-3, large amounts of accuracy are lost.

Note: A min, max and average take of the data does not constitute a robust statistical analysis of the outputs. However, with the restriction of only using numpy and not wishing to drift too far out of the scope of the assignment. It at least gives us some indication as to the fitness of the models.

References

[1] Viktor Zamaraev. Lecture notes. 2021.

[2] Carsten Klein. From biology to artificial intelligence: The perceptron. URL `https://towardsdatascience.com/from-biology-to-ai-the-perceptron-81abfdc788bf`.

[3] Jose Manuel Pereira, Mario Basto, and Amelia Ferreira da Silva. The logistic lasso and ridge regression in predicting corporate failure. *Procedia Economics and Finance*, 39:634–641, 2016.

[4] Ashish Singhal. Machine learning: Ridge regression in detail. URL `https://towardsdatascience.com/machine-learning-ridge-regression-in-detail-76787a2f8e2d`.