COURSEWORK 1

Data minning and visualisation

Shoukat, Usman
201537600

Q1. Explain the Perceptron algorithm for the binary classification case, providing its pseudocode.

Perceptron Algorithm:

Perceptron algorithm takes the motivation from the human nervous system. This algorithm helps in binary classification.

The model is trained by the external stimuli in the form of training data, and learning is done by changing the synaptic connections(weights) between the neuron and input data. The number of connections depends upon the number of features of input data. Then the neuron is trained to fire on some specific threshold which means that the neuron fires if the activation score gained through the linear equation ($a = \mathbf{W}^T\mathbf{X} + b$) is greater than that threshold θ, otherwise it does not fire. It is convenient to make the threshold equal to zero; for that purpose, we use bias and make it equal to $-\theta$, which adjusts the threshold equal to zero.

The algorithm works in two phases;

Training mode: In this mode, we train our model by adjusting the parameters according to the training data.

Testing mode: In this mode, we test our model on some unseen data and quantifies its performance on the unseen data.

Below is the pseudocode for both phases:

Training Phase:

```
1.  Training(dataset, iterations):
2.      W = np.zeros(X.shape[1]) Size depends upon the number of dimensions
3.      b = 0
4.      for _ in range(iterations):
5.          for X, y ε (dataset):
6.              a = W^T X + b
7.              if a * y <= 0:
8.                  W = weights + η * y * X
9.                  b = b + η * y
10.     return W, b
```

From the above pseudocode, you can see we pass on the dataset and the number of iterations for which we want to train our model to training function.

First, we initialise the weights vector **W** with size depending upon the number of dimensions of input data, either with zero or some random values. We also initialise the bias with either zero or some random value.

Then we run a loop for the number of iterations with which we want to train our Perceptron.

Then we have two options to send our data to the model to train it: online algorithm and batch training. In the online algorithm, we send our data points one by one to train our model, but for the batch training model, we send our data in batches.

After sending the data, we get its activation score using the linear equation a = $\mathbf{W}^T\mathbf{X}$ + b. And as we have set a threshold function for the neuron to fire, if it receives a score greater than this threshold, the neuron fires and returns 1, and if the activation score is less than the threshold, then the neuron does not fire and returns -1.

After getting this activation score, we compare it with the given labels, if the predictions are correct, then we do not do anything, and if the predictions are wrong, then we adjust(update) our weights accordingly depending upon which side was predicted wrong.

Once we have gone through all the data for enough iterations or if the model is converged, then we stop training and returns the weights and bias, which are actually our trained parameters.

Testing Phase:

```
1. Testing(testingDataPoint, W, b):
2.         a = W^T X + b
3.         If a >= 0:
4.                 return 1
5.         else:
6.                 return -1
```

Once done with training then we start testing our model. We send trained parameters (**W**, b) and the data which we want to test in the testing function. For the given test data, we compute its score using the linear equation (a = $\mathbf{W}^T\mathbf{X}$ + b) and check it against threshold 0. If the score is greater than zero, then we declare it as a positive class. Otherwise, we declare it as a negative class. We also take account of how many correct predictions have been given by the model to evaluate the accuracy of our model.

Q2 Implement a binary perceptron.

I have implemented a Perceptron and have attached its code with the file.
The perceptron is basically a class with methods including, fit(), netInput(), predict() and testing(). The details of each method and its parameters is written in the python file in the form of docstrings and commenting.

Q3. 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 classi_cation accuracies for each of the three classi_ers after 20 iterations. Which pair of classes is most difficult to separate?

Ans: I used the Perceptron that I implemented for the last question and changed it a little bit to load and label the data. I created three different objects of class Perceptron and trained them with three different pairs of classes using the training data. Later, I checked the training and testing accuracy of the models using the training data, testing data, and the three models' parameters. The accuracies were as follows:

Accuracies for Binary Classification

Accuracy Type	Class1 & Class2	Class2 & Class3	Class1 & Class3
Training	**100.0%**	**82.5%**	**100.0%**
Testing	**100.0%**	**75.0%**	**100.0%**

As the first and third models perfectly classifing the given test data and training data, we can say that the models for the 1st and 3rd pair are trained well and the pairs of respective models were easy to separate out. But the 2nd model, which was for the pair of class 2 and class 3, has low accuracy. As the 2nd model was able to predict fewer number correct labels as compared to the other models. So, we can say that class 2 and class 3 are difficult to separate.

Q4. 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.

Ans: I extended the Perceptron implemented for the previous question and trained the model using the 1 vs rest approach on training data and took the predictions of the model which gave me the highest activation score. After testing it on testing and training data, the accuracies of the model was as follows:

Accuracies for Multiclassification Classification

Accuracy Type	Class1	Class2	Class3	Overall
Training	**100.0%**	**72.5%**	**95.0%**	**89.17%**
Testing	**100.0%**	**50.0%**	**100.0%**	**83.33%**

From these accuracies, I can see that the model is able to predict the class 1 and class 3 very well, but its preformance for predicting the class 2 is not satisfactory.

Q5. Add an l2 regularisation term to your multi-class classifier implemented in question. 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.

By using the l2 regularisation term, the accuracies for each class were as follows:

Accuracies for Multiclassification Classification

Accuracy Type	λ	Class1	Class2	Class3	Overall
Training	0.01	100.0%	65.0%	100.0%	88.33%
Testing	0.01	100.0%	80.0%	100.0%	93.33%
Training	0.1	100.0%	0.0%	100.0%	66.67%
Testing	0.1	100.0%	0.0%	100.0%	66.67%
Training	1.0	0.0%	0.0%	100.0%	33.33%
Testing	1.0	0.0%	0.0%	100.0%	33.33%
Training	10.0	0.0%	0.0%	100.0%	33.33%
Testing	10.0	0.0%	0.0%	100.0%	33.33%
Training	100.0	0.0%	0.0%	100.0%	33.33%
Testing	100.0	0.0%	0.0%	100.0%	33.33%

For the lambda value of 0.01, we can see that the training accuracy is reduced by approximately one percent and the testing accuracy is increased by approximately 10 percent as compared to the model for which we were not using the regularisation. So, we can say that our model was a little bit overfitted before and now regularisation has smoothen the model and helped it improving its performance on test data with a little bit of compensation on test data. But as I started increasing the value of lambda, the model started reducing the accuracy and resulted in underfitting.

After increasing the lambda value over 1, the weights started increasng drastically which made the model to biased over one class i.e., class-3. At lambda value 10 and 100, after training the weights became infinite.

For the given dataset as the model performed well with the lamda value of 0.01. So, we can chose this value as our design parameter.

And from here we can also conclude that If our lambda value is too high, it might will result in underfitting of model. And model will be simple and won't learn enough about the training data to make useful predictions.

If our lambda value is too low, we run the risk of *overfitting* our data. Which means that our model will be more complex, and will learn too much about the particularities of the training data, and won't be able to generalize to unseen data.

Note: The acuuracies were varying depending upon different shuffling, the results presented here were for seed value of 100.

For the regularisation.py file, some values goes to infinity, so compiler might through a run time error. To avoid that use the following command followed by the location of the file **python -W ignore regularisation.py.**