SMAI Assignment2: Report

By: *Rashika Kheria*
201101101

Part I

The assignment aimed at finding linear decision boundary using various algorithms (Assumption: Given data is linearly separable). The **a** evaluated is perpendicular to the decision boundary.

Algorithms:

All the algorithms help us to learn a vector **a**, which linearly classifies the given training sample. We then use the learned **a** to classify any new data that we come across. The idea behind learning a is to reduce the space used for storage of feature vector of all the training samples. One can easily store the value of vector a, and can classify the test samples by computing the dot product of a and the feature vector of test sample. These algorithms basically applies on two-class problems but can be extended for the multi-class problem as well. These algorithms differ in the way of calculation of the perceptron function (J(**a**)).

We calculate the gradient of **J** in order to find the value of a for which J is minimum (Since, we get the correct values of **a** when the value of **J** is 0.). Then we update the value of a using equation:

$$\mathbf{a}_{k+1} = \mathbf{a}_k - n.(\text{gradient of } \mathbf{J})$$

here n = some constant value to control the increment in **a**(used 1/looping- variable for the assignment)

The vector y used in all the algorithms are attained by augmenting 1 in the feature vectors of all the training samples. Thus, we convert a d-dimensional vector into d+1 dimensional vector space. The feature vector of one class is multiplied with -1 in order to shift them to the same side of the decision boundary. We then try to find **a** which puts all the modified samples on same side.

Here, the dimension of **a** is d+1.

a) Batch Perceptron -

The batch perceptron algorithm treates all misclassified sample as a batch and hence it's definition of perceptron function is -

$$J(\mathbf{a}) = \text{sumall}(\mathbf{a}.\mathbf{y}_i^T)$$
$$\text{for all values for which } \mathbf{a}.\mathbf{y}_i^T < 0$$

The function J(a) sums the values of all the misclassified samples. The gradient function for the J(a) is

$$\text{gradient of } J(\mathbf{a}) = \text{sumall}(\mathbf{y}_i)$$

b) Single Sample Perceptron -

The single sample perceptron algorithm takes any random misclassified sample and compute the value of **J(a)** and hence it's definition of perceptron function is -

$$J(\mathbf{a}) = \mathbf{a}.\mathbf{y}_i^T$$
$$\text{for any random value of } \mathbf{y} \text{ for which } \mathbf{a}.\mathbf{y}_i^T < 0$$

The function J(a) takes any one misclassified sample. The gradient function for the J(a) is

$$\text{gradient of } J(\mathbf{a}) = \mathbf{y}_i$$

c) Batch perceptron with margin -

The batch perceptron with margin algorithm treates all misclassified sample as a batch and hence it's definition of perceptron function is -

$$J(\mathbf{a}) = \text{sumall}(\mathbf{a}.\mathbf{y}_i^T - b)$$
$$\text{for all values for which } \mathbf{a}.\mathbf{y}_i^T - b < 0$$

$$\text{here, } b = \text{margin (value between 5 and 10)}$$

The function J(a) sums the values of all the misclassified samples. The value of b determines the minimum distance from which the training data feature vector is placed. The gradient function for the J(a) is

$$\text{gradient of } J(\mathbf{a}) = \text{sumall}(\mathbf{y})$$

d) Single sample perceptron with margin -

The single sample perceptron with margin algorithm takes any random misclassified sample and compute the value of **J(a)** and hence it's definition of perceptron function is -

$$J(\mathbf{a}) = \mathbf{a}.\mathbf{y}^T - b$$
$$\text{for any random value of } \mathbf{y} \text{ for which } \mathbf{a}.\mathbf{y}^T - b < 0$$

$$\text{here, } b = \text{margin (value between 5 and 10)}$$

The function J(a) takes any random misclassified sample. The value of b determines the minimum distance from which the training data feature vector is placed. The gradient function for the J(a) is

$$\text{gradient of } J(\mathbf{a}) = \mathbf{y}$$

e) Batch Relaxation algorithm with margin -

The batch relaxation algorithm treates all misclassified sample as a batch and hence it's definition of perceptron function is -

$$J(\mathbf{a}) = (1/2)*\text{sumall}((\mathbf{a}.\mathbf{y}^T - \mathbf{b})^2)/(\mathbf{y}^T)^2)$$
$$\text{for all values for which } \mathbf{a}.\mathbf{y}^T - b < 0$$

The function J(a) sums the values of all the misclassified samples. Here, the value of b determines the minimum distance from which the training data feature vector is placed. The function is squared to make it smooth so that gradient descent can work efficiently and divided by $(\mathbf{y}^T)^2$ to normalize the J(a) function(in order to ensure that the large values of **y** does not greatly affect the change in **a**). The gradient function for the J(a) is

$$\text{gradient of } J(\mathbf{a}) = \text{sumall}(((\mathbf{a}.\mathbf{y}^T - \mathbf{b})*\mathbf{y}^T)/(\mathbf{y}^T)^2)$$

Implementation of Algorithms:

The general format of the implementation is :

a = arbitrary initialization

function algorithm (training data):

    for all training samples:
        calculate the value of $\mathbf{a}.\mathbf{y}_i^T$ / $\mathbf{a}.\mathbf{y}_i^T$ - **b**

        if (misclassified)
            add y_i sample to **Y**
        end if

    end for

    update a with the appropriate gradient value multiplied with n

    if(check for the breaking condition == true)
        break;
    end if

        return a;

end function

Stopping Criteria:

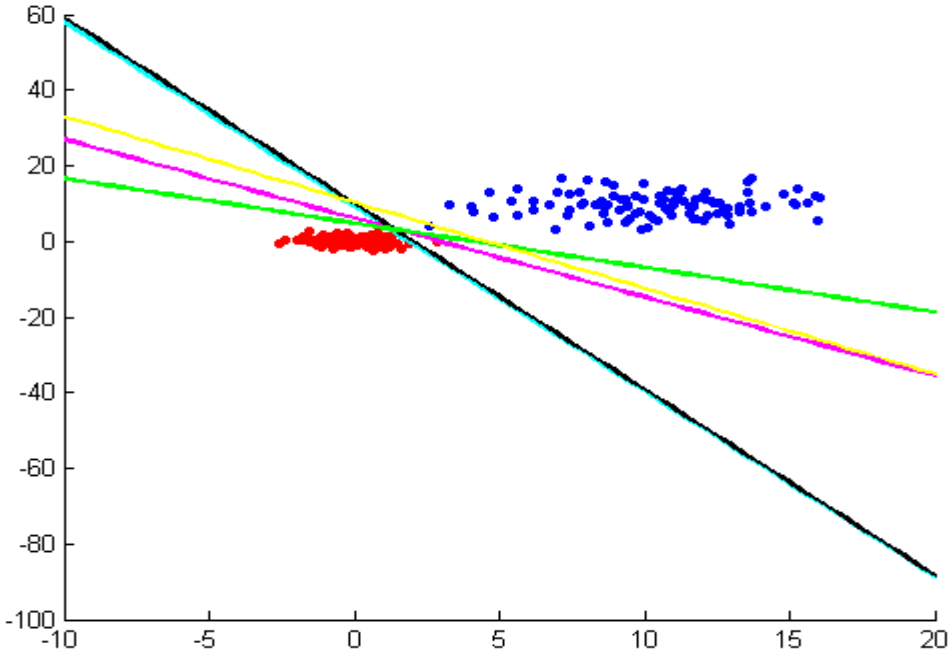
a) No misclassified sample is left i.e. **Y** is empty.
b) The number of iterations is too large i.e. #iterations > 5000.
c) The value which we increment in A is too small i.e. (abs(**J(a)***n) > 0.1) .

Results:

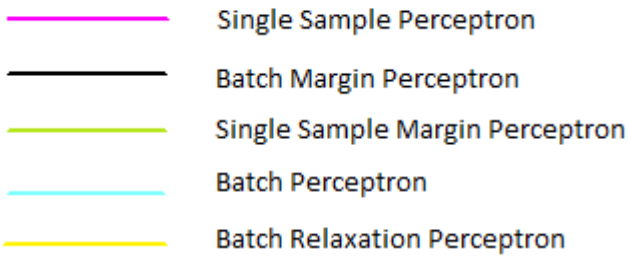
The value of **a** received from each algorithm is plotted against the training sample points on a graph. All the learned **a** are correctly classifying the training samples.

Red Dots: Class 0
X-axis: One component of feature vector

Blue Dots: Class 1
Y-axis: Other component of feature vector



Index :



*Accuracy attained by implementing all learned **a** and 1-NN on test samples :*

	Batch Perceptron	Single Sample Perceptron	Batch With Margin Perceptron	Single Sample With Margin Perceptron	Batch Relaxation With Margin Perceptron	1 – NN algorithm
Accuracy Rate	98.5%	100%	99%	100%	100%	100%

Training Data Classification Error Rate :

	Batch Perceptron	Single Sample Perceptron	Batch With Margin Perceptron	Single Sample With Margin Perceptron	Batch Relaxation Perceptron	1 – NN algorithm
Error Rate	0.5%	0.0%	0.5%	0.0%	0.5%	0%

Part II

Algorithm Implemented :

Pairwise Classification (One vs One Classification) with *Majority Voting*.

In Pairwise Classification, we take 2 classes at a time and train the data for those two classes and obtain the **a.** We repeat this process for $^{10}C_2$ times.

After getting the values of all **a,** we check the testing data against all classifiers and it is provided with the label of the class for which it is classified in majority.

Accuracy attained by implementing each algorithm and 1-NN in pairwise classification :

	Batch Perceptron	Single Sample Perceptron	Batch With Margin Perceptron	Single Sample With Margin Perceptron	Batch Relaxation Perceptron	1 – NN algorithm
Accuracy Rate	86.19%	85.60%	86.19%	85.60%	86.19%	100%

Training Data Classification Error Rate for each algorithm:

	Batch Perceptron	Single Sample Perceptron	Batch With Margin Perceptron	Single Sample With Margin Perceptron	Batch Relaxation Perceptron	1 – NN algorithm
Error Rate	6.1%	0.0%	6.3%	0.0%	12.17%	0%

Why choose pairwise classification over 1 vs rest ?

Problems with more classes like digit recognition in MNIST dataset, the one against one strategy is more preferable because -

a) The pairwise strategy is more modular and hence speeds up the decision making process by combining with other classifiers.

b) If the number of training samples are large, the training time can become problematic and hence pairwise strategy may help more.

c) Though the number of decision boundaries that we have to store is more than one vs one classification but it provides more accurate decision boundary since it deals with 2 classes at a time.

Limits of Linear Classifier:

If the problem is not linearly classifiable and its class boundaries cannot be approximated well with linear hyperplanes, then linear classifiers can prove fatal as then they cannot converge.