1 Linear Classifiers

1. About linear classifiers: $\bar{x}$ is a vector of real-valued numerical input features; we will say there are p of them. The response is a binary class Y; it will simplify book-keeping to say that the two classes are +1 and -1. In linear classification, we seek to divide the two classes by a linear separator in the feature space. If $p = 2$, the separator is a line, if $p = 3$ it's a plane, and in general it's a $(p-1)$-dimensional hyper-plane in a p-dimensional space; I will say "plane" without loss of generality.

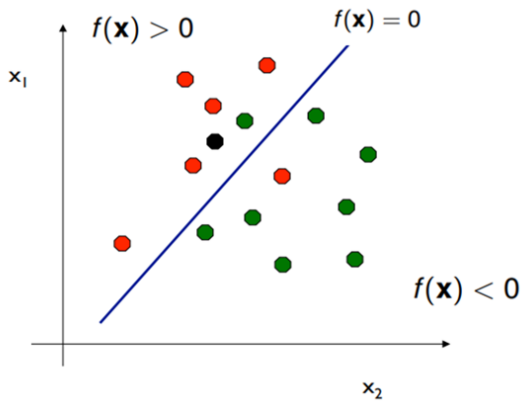

Figure 1

Please refer link to gain a good understanding of Linear Classifiers

(`http://www.stat.cmu.edu/~cshalizi/350/lectures/25/lecture-25.pdf`)

2. Tuning the hyper parameters is critical in Perceptron. The hyper parameters concerned here is the learning rate. You need to use the techniques such as "Loss vs Epoch" graph for different learning rate to come up with the right learning rate.

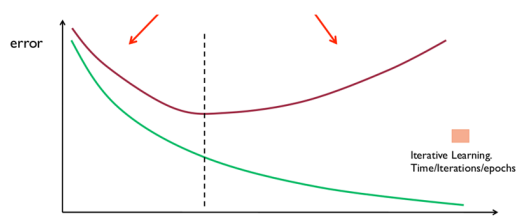

Figure 2

3. For a linear classifier it is important to choose the loss function. A simple example of loss function in linear classifier is multplication of the hypothesis function $f(x)$ with the binary

labels L(taking value 1 and -1). I.e. classification happens based on whether $f(x)*L$ is greater than or less than 0.

(Please check the following for different loss functions that can be used:

`https://en.wikipedia.org/wiki/Loss_functions_for_classification`, which further pre-empts the manipulation of the labels accordingly.

4. It is critical to select the value of the number of epochs based on stabilization of loss value. It might be incorrect to prematurely stop the training (and thus resulting in a wrong model), as it is possible for loss to fluctuate in later epochs, this could happen especially if the problem is not linearly separable. Please find a sample Python code snipped of training a linear classifier. This shows how through various iterations, the values of the weight keep getting manipulated, while being compared against the loss function.

```
def train_weights(train, l_rate, n_epoch):
    weights = [0.0 for i in range(len(train[0]))]
    for epoch in range(n_epoch):
        sum_error = 0.0
        for row in train:
            prediction = predict(row, weights)
            error = row[-1] - prediction
            sum_error += error**2
            weights[0] = weights[0] + l_rate * error
            for i in range(len(row)-1):
                weights[i + 1] = weights[i + 1] + l_rate * error * row[i]
        if epoch % 50 == 0:
            print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate, sum_error))
    return weights
```

Figure 3