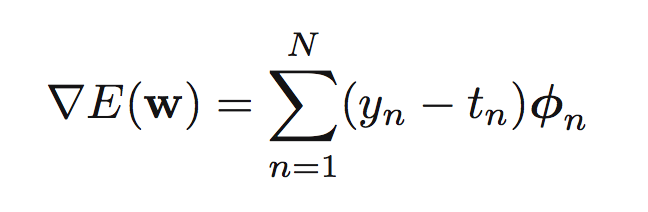
**MLIA Fall 2017 Homework 1: Logistic Regression**

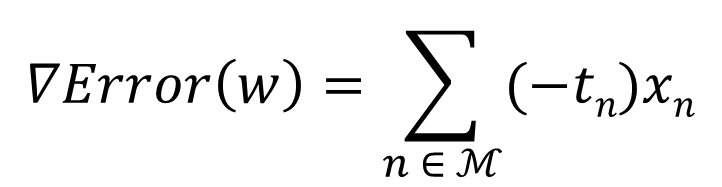
**Binary-label implementation:**

Formulas:

Gradient:

****

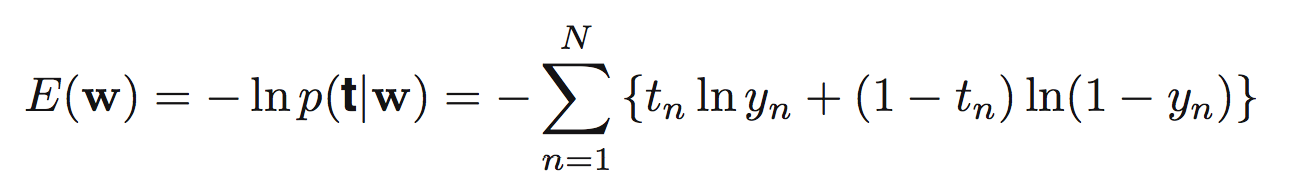
Percent Gradient:



Updating the weights:



Cross-entropy loss:



Sigmoid function:



Function implementations:

mytrain\_binary implementation:

First, I try to implement the mytrain\_binary function which takes X\_train and y\_train as parameters. I initialize the weight vector w1 to zeros and with (X\_train.shape[1] + 1, 1) dimensions which is (3,1), then I create a new variable called X and I insert a columns of ones in the X\_train matriz and store it in the X matrix. Then I initialize some helper variables as optW and newW. Then I proceed to make a loop until the gradient converges. Inside the loop, I calculate the gradient for every iteration, assign the new weight using the update formula. In this case, I used a constant of 0.5 to calculate each new weight, finally I assign the new weight to the weight vector and finally check that the ||gradW|| is less than 0.000001. Finally, it will return the optimized weight vector optW.

gradComp implementation:

the gradComp function takes X, tr and w as parameters. Then, it calculates the percentage gradient and uses to rest it with the training data. Finally, the grad is calculated doing a multiplication between the transpose of X and the difference we just calculated divided by the size of the training data. Finally, it returns the calculated gradient “grad”.

Sigmoid implementation:

This function takes a matrix and applies the sigmoid computation to the given matrix.

mypredict\_binary:

it computes a matrix by taking the sigmoid function of the multiplication of the X matrix and weight. It goes through a loop for each column of the matrix and finally outputs that matrix of number between 0 and 1

mytest\_binary:

This function returns a matrix of N elements, each of them between 0 and K-1. It takes the multiplication of X and w, and then it selects if the number is greater than 0.5. If the number is greater than 0.5, it outputs a 1, 0 otherwise. \

Accuracy and average running time:

1. Easy dataset:

Accuracy: 100%

Average running time: 30.165584087371826s

1. Medium dataset:

Accuracy: 86%

Average running time: 1.0057978630065918s

1. Hard dataset

Accuracy: 74%

Average running time: 0.5329837799072266s

1. Moons dataset

Accuracy: 86%

Average running time: 1.1596500873565674s

1. Circles dataset

Accuracy: 40%

Average running time: 0.11732745170593262s

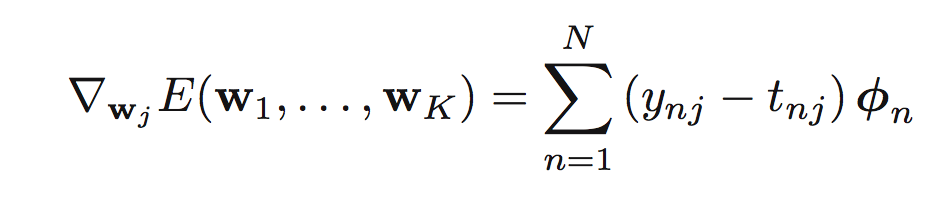
Convergence rate:

1. Easy dataset: 0.00001
2. Medium dataset: 0.0000001
3. Hard dataset: 0.000000001

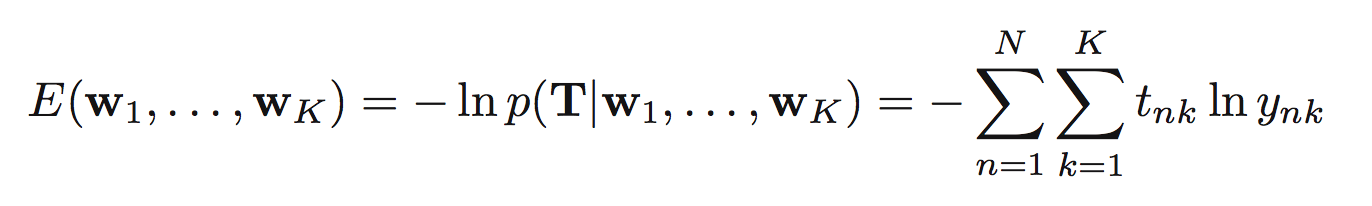
**Multi-label implementation:**

Formulas:

Gradient:



Cross-entropy loss



Function implementation:

mytrain\_multi implementation:

First, I initiate the weight vector with size of (D+1, k), then I insert a column of 1s in X, then I calculate the gradient of the multi-label matrix, I set the convergence rate to 1 because if I put a smaller number, the program will run for too long. Finally, I output the new gradient.

gradComp implementation:

the gradComp function takes X, tr and w as parameters. Then, it calculates the percentage gradient and uses to rest it with the training data. For each column in the weight vector, the function calculates a gradient and then stores it in every column of the new gradient. Finally, it returns the gradient matrix.

mypredict\_multi:

it computes a matrix by taking the sigmoid function of the multiplication of the X matrix and weight. It goes through a loop for each column of the matrix and finally outputs that matrix of number between 0 and 1

mytest\_multi:

This function returns a matrix of N elements, each of them between 0 and K-1. It take the multiplication of X and w, and then it selects if the number is greater than 0. If the number is greater than 0, it outputs a 1, 0 otherwise. Then, it goes through an iteration until it finish with all the K-1 elements. Finally, it outputs that result.

Accuracy and average running time:

1. Easy dataset:

Accuracy: 30%

Average running time: 0.025570392608642578s

1. Medium dataset:

Accuracy: 28%

Average running time: 0.006518840789794922s

1. Hard dataset

Accuracy: 30%

Average running time: 0.006518840789794922s

Convergence rate:

1. Easy dataset: 1
2. Medium dataset: 1
3. Hard dataset: 1