ECEN 765 Machine Learning with Networks
programming assignment 2

Jicheng Gong

November 3, 2017

1 Perceptron for Logistic Regression

1.1 Unit l_1, l_2 norm

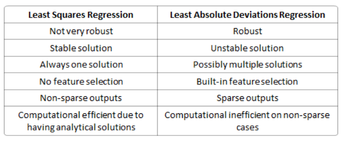
While practicing machine learning, we may have come upon a choice of deciding whether to use the L1-norm and L2-norm for regularization, or as a loss function, etc.
L1-norm is known as least absolute deviations (LAD), least absolute errors (LAE). It is basically minimizing the sum of the absolute differences (S) between the target value (Yi) and the estimated values (f(xi)):

$$S = \sum_{i=1}^{n} |y_i - f(x_i)|$$

L2-norm is also known as least squares. It is basically minimizing the sum of the square of the differences (S) between the target value (Yi) and the estimated values (f(xi):

$$S = \sum_{i=1}^{n} (y_i - f(x_i))^2$$

The differences of l1 and l2 can be summarized as follows:



Least Squares Regression	Least Absolute Deviations Regression
Not very robust	Robust
Stable solution	Unstable solution
Always one solution	Possibly multiple solutions
No feature selection	Built-in feature selection
Non-sparse outputs	Sparse outputs
Computational efficient due to having analytical solutions	Computational inefficient on non-sparse cases

Figure 1: L1 norm and L2 norm

Intuitively speaking, since a L2-norm squares the error (increasing by a lot if error $>$ 1), the model will see a much larger error (e vs e2) than the L1-norm, so the model is much more sensitive to this example, and adjusts the model to minimize this error. If this example is an outlier, the model will be adjusted to minimize this single outlier case, at the expense of many other common examples, since the errors of these common examples are small compared to that single outlier case.

1.2 Perceptron

Here we consider an approximation algorithm, let

$$\hat{y} = argmax_{y \in 0,1} p(y|x_i, \theta)$$

$$\hat{y} = \begin{cases} 1, & \text{if } p > 0.5 \\ 0, & \text{if } n < 0.5 \end{cases}$$

Then if $\hat{y_i} - y_i \neq 0$, we have made an error, if $\hat{y_i} - y_i = 0$ we guessed the right label. At each step, we update the weight vector by adding on the gradient. If we predicted correctly, then $\hat{y}_i = y_i$, so the approximate gradient is zero and we do not change the weight vector. But if x_i is misclassified, we update the weights.

$$\theta_k = \theta_{k-1} + \eta_k (y_i - \hat{y}_i) x_i$$

So this is the perceptron algorithm for logistic regression.

1.3 tests

Table 1: Perceptron error rates

Method	Perceptron L0	Perceptron L1	Perceptron L2
Training Error rate	0.035	0.04	0.04
Test Error rate	0.1447	0.1579	0.1579

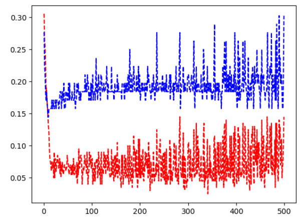


Figure 2: Perceptron l0 Blue is testing error, Red is training error

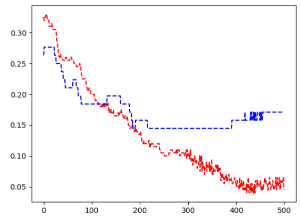


Figure 3: Perceptron l1

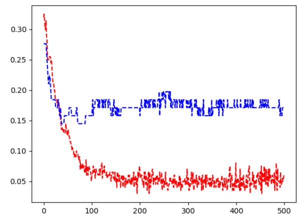


Figure 4: Perceptron l2

From the pictures we can see that Perceptron algorithm without regularization will result in oscillation. While l2 regularization is the best one for Perceptron because it has a good accuracy for test data and fast convergence rate.

Next Part is the average Perceptron

From the plots below we can find that the average Perceptron algorithm will have no oscillation and it has a good accuracy for both training and testing data even without any regularization.

Table 2: Average Perceptron error rates

Method	Perceptron L0	Perceptron L1	Perceptron L2
Training Error rate	0.025	0.04	0.03
Test Error rate	0.1447	0.1447	0.1447

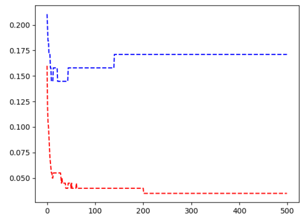


Figure 5: Average Perceptron l0

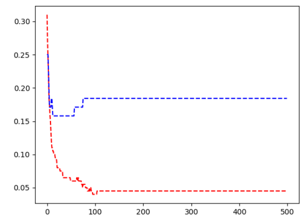


Figure 6: Average Perceptron l1

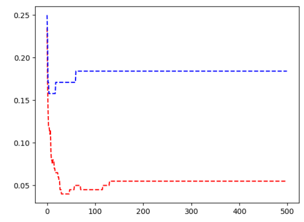


Figure 7: Average Perceptron l2

1.4 Difference

From the results above we can find that even with l1 or l2 regularization, the original Perceptron algorithm will have certain degree of oscillation, and it will not converge during the iteration. After regularization, the result will be much more stable, but may still have some oscillation in the end. And l2 regularization will achieve a better result than l1 regularization because it can converge quickly. However, the average Perceptron which maintains all the weights during the iteration will have a very reliable result no matter whether there is a regularization.

2 Gradient Descent for Logistic Regression

2.1 Gradient Descent

Similar to the Perceptron algorithm, the iterative updating θ is as follows:

$$\theta_k = \theta_{k-1} + \eta_k(h_i - y_i)x_i$$

h_i is what we predict using the sigmoid function

$$h_i = \frac{1}{1 + e^{-\beta^T x_i}}$$

2.2 test

Table 3: Gradient descent error rates

Method	GD L0	GD L1	GD L2
Training Error rate	0.045	0.2	0.125
Test Error rate	0.1579	0.197	0.1447

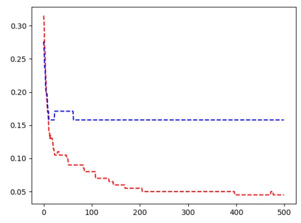


Figure 8: GD l0

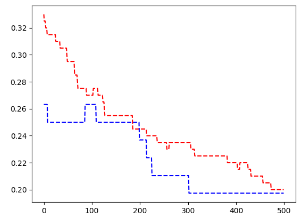


Figure 9: GD l1

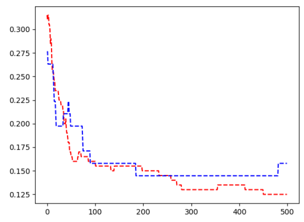


Figure 10: GD l2

Table 4: Average Gradient Descent error rates

Method	AGD L0	AGD L1	AGD L2
Training Error rate	0.05	0.245	0.15
Test Error rate	0.1579	0.23687	0.1447

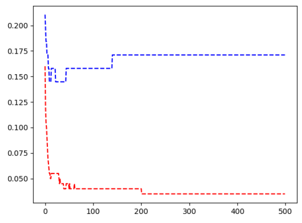


Figure 11: Average GD l0

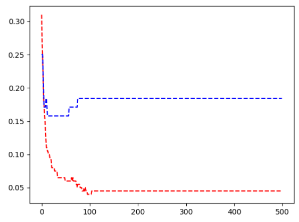


Figure 12: Average GD l1

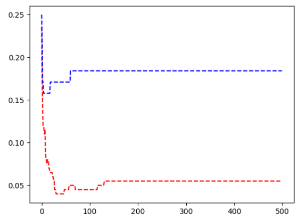


Figure 13: Average GD l2

From the results we can see that the gradient descent algorithm alone has a relatively good result compare to the Perceptron algorithm. With the l1 regularization, the converge rate will be much slower and the accuracy may decrease. The l2 regularization will increase the converge rate for training data but have no effect on testing data. And it can slightly increase the accuracy for testing data.
The averaged Gradient Descent is basically the same as the non averaged one. In summary, I would say gradient descent algorithm with l2 regularization is the most suitable one.

3 Locally weighted Newton's Method

3.1 Newton's method

First of all, if we leave the locally weight thing alone, we will use following updating algorithm:

$$g = \frac{\partial l(\beta)}{\partial \beta} = X^T(Y - p) - \lambda\beta$$

$$H = -XAX^T - \lambda I$$

$$\beta_{new} = \beta_{old} - H^{-1} * g$$

A is a diagonal matrix and each element is $p * (1 - p)$. It's error plot is as follows:

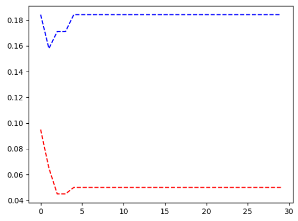


Figure 14: Newton's method without locally weight

We can find that it converges very fast with only less than 10 iterations. It's training error is 0.045 and testing error is 0.1842 which are not the best one.

3.2 Local weights

Local weights regression is a non-parametric learning method which means it needs us to store the training data while doing the prediction. Ever time we want to make a prediction, we fit the regression where the training data is close to the test data.
From the formula $w^i = exp(-\frac{||x-x^i||^2_{l_2}}{2\tau^2})$ we can find that data which is close will have a value close to 1 while faraway value will have a smaller weight. This enables us to choose important data to fit.
In common, the local weight algorithm can solve the over-fitting problem.

So in my algorithm I first calculate the local weight matrix for each tuple of training data. Then insert the weight matrix into the iteration process. So the new iteration formula will be:

$$g = X^T W(Y - p) - \lambda\beta$$

$$H = -XWAW^T X^T - \lambda I$$

$$\beta_{new} = \beta_{old} - H^{-1} * g$$

Where W is a 200×200 diagonal matrix where each $w^i = exp(-\frac{||x-x^i||^2_{l_2}}{2\tau^2})$.

3.3 Tests

Table 5: Newton's method error rates

Method	$\tau = 5$	$\tau = 1$	$\tau = 0.5$	$\tau = 0.1$	$\tau = 0.01$
Training Error rate	0.035	0.005	0.16	0.265	0.27
Test Error rate	0.171	0.1447	0.1447	0.25	0.27

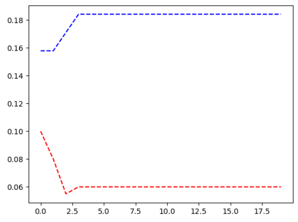


Figure 15: $\tau = 5$

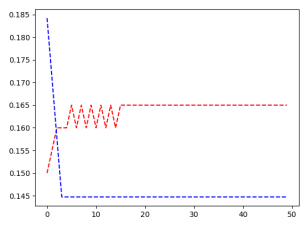


Figure 16: $\tau = 1$

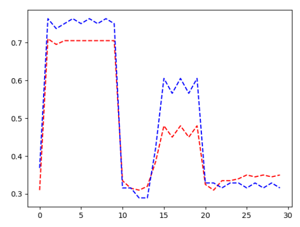


Figure 17: $\tau = 0.5$

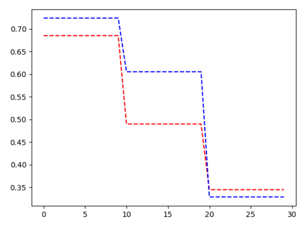


Figure 18: $\tau = 0.1$

Comparing to the Perceptron algorithm, the locally weighted logsitic regression has a faster convergence rate and the same accuracy. It only needs less than 10 iteration to achieve a relatively good result.
However, since it's a non-parametric algorithm, it needs to fit the regression for each particular sample and should always keep the training data in memory. This will make the algorithm very slow and not suitable for large scale data.

The parameter τ for locally weight is used to control the choosing process. The larger the τ is, the less distinctive the weight will be. When $\tau = 5$ the result is almost the same as newton's method without locally weights. However, according to experiment result, a too small τ will result in rise in error rate. So we need to be careful when choosing the τ value.