Machine Learning, 2024 Spring
Assignment 3

Notice

Plagiarizer will get 0 points.
LaTeXis highly recommended. Otherwise you should write as legibly as possible.

Problem 1 For logistic regression, show that

$$\nabla E_{\text{in}}(\mathbf{w}) = -\frac{1}{N}\sum_{n=1}^{N}\frac{y_n\mathbf{x}_n}{1+e^{y_n\mathbf{w}^{\mathrm{T}}\mathbf{x}_n}} = \frac{1}{N}\sum_{n=1}^{N}-y_n\mathbf{x}_n\theta\left(-y_n\mathbf{w}^{\mathrm{T}}\mathbf{x}_n\right) \tag{1}$$

Argue that a 'misclassified' example contributes more to the gradient than a correctly classified one.
Solution

$$E_{\text{in}}(\mathbf{w}) = -\frac{1}{N}\sum_{n=1}^{N}\ln(1+e^{-y_i w^T x_i})$$

$$\Leftrightarrow \nabla E_{\text{in}}(\mathbf{w}) = -\frac{1}{N}\sum_{n=1}^{N}\frac{y_n x_n}{1+e^{y_n\mathbf{w}^{\mathbf{T}}x_n}} = \frac{1}{N}\sum_{n=1}^{N}-y_n x_n\theta(-y_n\mathbf{w}^{\mathbf{T}}x_n)$$

$$\theta_w(x) = \frac{1}{1+e^{-\mathbf{w}^{\mathbf{T}}x}}$$

For a misclassified sample, if its true label is $y = 1$, but the model's predicted value $\theta_w(x)$ is close to 0 (i.e., the model incorrectly classifies it as negative), or conversely, the loss function gives a higher value. This is because the cross-entropy loss function will have a large negative log-likelihood value in this case, and its gradient will vary more from the parameters.

Problem 2 For linear regression, the out-of-sample error is

$$E_{\text{out}}(h) = \mathbb{E}[(h(x) - y)^2]. \tag{2}$$

Show that among all hypotheses, the one that minimizes E_{out} is given by

$$h^*(x) = \mathbb{E}[y|x]. \tag{3}$$

The function h^* can be treated as a deterministic target function, in which case we can write $y = h^*(x) + \epsilon(x)$ where $\epsilon(x)$ is an (input dependent) noise variable. Show that $\epsilon(x)$ has expected value zero.

Solution

$$\frac{d}{dh(x)} E\left[(y - h(x))^2\right] = -2E[y - h(x)|x]$$

$$= -2E[y|x] + 2E[h(x)|x] = 0$$

$$\Leftrightarrow E[y|x] = E[h(x)|x]$$

$$\Leftrightarrow h^*(x) = E[y|x]$$

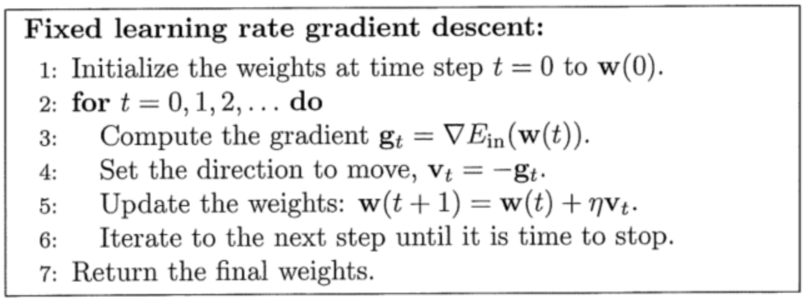
$$E\left[\epsilon(x)\right] = E\left[(y - h^*(x))\right]$$

$$\Leftrightarrow E(y|x) - h^*(x)$$

$$\Leftrightarrow E(y|x) - E(y|x) = 0$$

Problem 3 According to the iterative format provided as follow:

- Use the SUV dataset to implement (using Python or MATLAB) the Gradient Descent method to find the optimal model for logistic regression.
- What is your test error? What are the sizes of the training set and test set, respectively?
- What is your learning rate? How was it chosen? How many steps were iterated in total?
- Present the results of the last 10 steps produced by your algorithm, including the loss, learning rate, the L2 norm of the gradient, and the number of function evaluations and gradient evaluations.

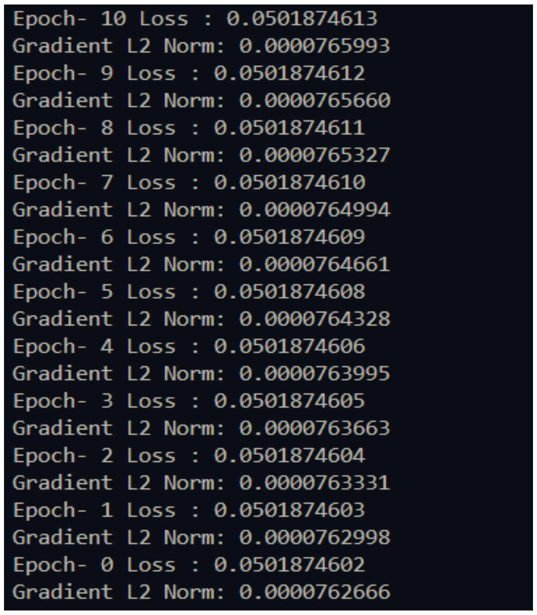


```
Fixed learning rate gradient descent:
1: Initialize the weights at time step $t = 0$ to $\mathbf{w}(0)$.
2: for $t = 0, 1, 2, \dots$ do
3:     Compute the gradient $\mathbf{g}_t = \nabla E_{\text{in}}(\mathbf{w}(t))$.
4:     Set the direction to move, $\mathbf{v}_t = -\mathbf{g}_t$.
5:     Update the weights: $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \mathbf{v}_t$.
6:     Iterate to the next step until it is time to stop.
7: Return the final weights.
```

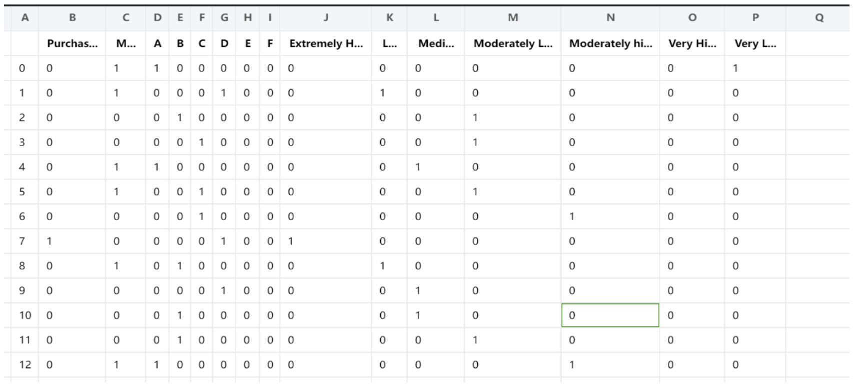
Dataset reference and description
Dataset and download

Solution

- $$test \quad error = 0.0501874602$$
  traing set size is 80 percents of dataset,test set size is 20 percents of dataset.
- learnin rate = 0.02 ,This method simplifies the training process by keeping the learning rate constant from the beginning to the end of training. The fixed learning rate can be determined based on empirical knowledge, previous experimentation, or common practices in the field. iterate 89126 steps in total.
- last 10 steps :



```
Epoch- 10 Loss : 0.0501874613
Gradient L2 Norm: 0.0000765993
Epoch- 9 Loss : 0.0501874612
Gradient L2 Norm: 0.0000765660
Epoch- 8 Loss : 0.0501874611
Gradient L2 Norm: 0.0000765327
Epoch- 7 Loss : 0.0501874610
Gradient L2 Norm: 0.0000764994
Epoch- 6 Loss : 0.0501874609
Gradient L2 Norm: 0.0000764661
Epoch- 5 Loss : 0.0501874608
Gradient L2 Norm: 0.0000764328
Epoch- 4 Loss : 0.0501874606
Gradient L2 Norm: 0.0000763995
Epoch- 3 Loss : 0.0501874605
Gradient L2 Norm: 0.0000763663
Epoch- 2 Loss : 0.0501874604
Gradient L2 Norm: 0.0000763331
Epoch- 1 Loss : 0.0501874603
Gradient L2 Norm: 0.0000762998
Epoch- 0 Loss : 0.0501874602
Gradient L2 Norm: 0.0000762666
```



A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
	Purchas...	M...	A	B	C	D	E	F	Extremely H...	L...	Medi...	Moderately L...	Moderately hi...	Very Hi...	Very L...	
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	
1	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	
2	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	
3	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	
4	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	
5	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	
6	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	
7	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	
8	0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	
9	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	
10	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	
11	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	
12	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	

- final-data.csv as this graphic,
-

```
#include <vector>
#include "rapidcsv.h"
#include <math.h>
#include <iomanip>
double predict(std::vector<double> X, std::vector<double> weight)
{
    double sum = 0.0f;
    for (int i = 0; i < weight.size(); i++)
        sum += 1.0 / (1 + exp(-X[i] * weight[i]));

    return sum;
}
double loss(std::vector<std::vector<double>> X, std::vector<double> y,
     std::vector<double> weight)
{
    int n = X.size();
    int m = X[0].size();
    double sum = 0.0;
    for (int i = 0; i < n; i++)
    {
        sum += y[i] * log(predict(X[i], weight)) + (1 - y[i]) * log(1 -
            predict(X[i], weight));
    }
    return -sum / n;
}

double lossDerivation(std::vector<std::vector<double>> X,
     std::vector<double> y, std::vector<double> weight, int j, int
     train_rate = 0.8)
{
    int n = X.size() * train_rate;
    int m = X[0].size();

    double derivation = 0.0;
    for (int i = 0; i < n; i++)
    {
        derivation += (predict(X[i], weight) - y[i]) / n * X[i][j];
    }

    return derivation;
}
double calculateGradientL2Norm(std::vector<std::vector<double>> X,
     std::vector<double> y, std::vector<double> weight)
{
    double l2norm = 0.0;
    int dim = weight.size();
    for (int j = 0; j < dim; j++)
    {
        double gradient = lossDerivation(X, y, weight, j);
        l2norm += gradient * gradient;
    }
    return std::sqrt(l2norm);
}

void gradient_descent(std::vector<std::vector<double>> X,
     std::vector<double> y, std::vector<double> &weights, const int &epoch,
     int train_rate = 0.8)
{
    int epoch_ = epoch;
    double learning_rate = 0.02;
    double pre_loss = 0.0;
    while (epoch_--)
    {
        std::cout << "Epoch- " << epoch_ << std::fixed <<
             std::setprecision(10) << " Loss : " << loss(X, y, weights) <<
             std::endl;

        for (int j = 0; j < weights.size(); j++)
        {
            weights[j] -= learning_rate * lossDerivation(X, y, weights, j,
                 train_rate);
        }
        // if (abs(pre_loss - loss(X, y, weights)) < 1e-10)
        //     break;
        std::cout << "Gradient L2 Norm: " << calculateGradientL2Norm(X, y,
             weights) << std::endl;
        pre_loss = loss(X, y, weights);
        // std::cout << "Loss : " << loss(X, y, weights) << std::endl;
    }
    for (auto &weight : weights)
        std::cout << std::fixed << std::setprecision(4) << weight << " ";
    std::cout << std::endl;
}
int main()
{
    rapidcsv::Document doc("/home/mywork/cxxTorch/source/final_data.csv",
         rapidcsv::LabelParams(0, 0));

    int dim = doc.GetColumnCount();
    int row = doc.GetRowCount();
    std::vector<double> weight(dim, 1.0);
    std::vector<std::vector<double>> x_data;
    std::vector<double> y_data;
    for (int i = 0; i < row; i++)
    {
        auto original = std::vector<double>(doc.GetRow<double>(i));
        // x_data.push_back(tmp);
        std::vector<double> tmp = std::vector<double>(original.begin() + 1,
             original.end());
        tmp.push_back(1.0);
        // x_data[x_data.size() - 1].push_back(1.0);
        x_data.push_back(tmp);
        y_data.push_back(doc.GetRow<double>(i)[0]);
    }
    gradient_descent(x_data, y_data, weight, 10);

    double cnt = 0;
    for (int i = 0; i < 100; i++)
    {
        double predict_value = predict(x_data[i], weight);
        double index = 0;
```

```cpp
        if (predict_value > 0.5)
        {
            index = 1.0;
        }
        else
        {
            index = 0.0;
        }
        if (index == y_data[i])
        {
            cnt++;
        }
    }
    std::cout << "Accuracy : " << cnt / 100 << std::endl;

    return 0;
}
```