华南理工大学

South China University of Technology

The Experiment Report of Machine Learning

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
Liu Rui

Supervisor:
Qingyao Wu

Student ID：
201721045626

Grade:
PostGraduate

December 9, 2017

Linear Regression, Linear Classification and Gradient Descent

Abstract—In this experiment we aim to further understand of linear regression and gradient descent and conduct some experiments under small scale data set. We aim to realize the process of optimization and adjusting parameters during this experiment.

I. INTRODUCTION

In statistics, linear regression is a linear approach for modeling the relationship between a scalar dependent variable y and one or more explanatory variables (or independent variables) denoted X. The case of one explanatory variable is called simple linear regression.

Considerate of the method used to meet the goal of statistical classification is to use an object's characteristics to identify which class (or group) it belongs to. A linear classifier achieves this by making a classification decision based on the value of a linear combination of the characteristics.

In machine learning, support vector machines (SVM) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis.Here are the experimental steps:

1) Load the experiment data and divide the dataset into training set and validation set.
2) Initialize linear model parameters. Set all parameter into zero, initialize it randomly or with normal distribution.
3) Define the loss function of the linear regression to be Least squared loss, and the loss function of the linear classification to be Hinge loss.
4) Compute the gradient of the loss function with respect to the weight W and bias b.
5) Update the parameters W and b.
6) Repeat above steps for several times until convergence.

In this experiment we using these linear models introduced above to training dataset and validation the training results. Part II introduce the methods and theory of linear regression and linear classification and the theory of gradient descent, part III shows the experimental coda and results, finally make some conclusions in part IV.

II. METHODS AND THEORY

A. Linear Regression and Gradient Descent

In linear regression, the relationships are modeled using linear predictor functions whose unknown model parameters are estimated from the data. Such models are called linear models. Most commonly, the conditional mean of y given the value of X is assumed to be an affine function of X; less commonly, the median or some other quantile of the conditional distribution of y given X is expressed as a linear function of X. Like all forms of regression analysis, linear regression focuses on the conditional probability distribution of y given X, rather than on the joint probability distribution of y and X, which is the domain of multivariate analysis.

Linear regression models are often fitted using the least squares approach, but they may also be fitted in other ways, such as by minimizing the "lack of fit" in some other norm (as with least absolute deviations regression), or by minimizing a penalized version of the least squares loss function:

$$L = \frac{1}{2n}\sum_{i=1}^{n}(y_i - W^T x_i)^2$$

Gradient descent is a first-order iterative optimization algorithm for finding the minimum of a function. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point. In linear regression model ,the gradient descent function with the respect of the W is:

$$G = \frac{1}{n}\sum_{i=1}^{n}(-x_i) * (y_i - W^T x_i)$$

B. Linear Classification and Gradient Descent

If the input feature vector to the classifier is a real vector **x**, then the output score is

$$y = f(\vec{w} \cdot \vec{x}) = f\left(\sum_j w_j x_j\right)$$

where **w** is a real vector of weights and f is a function that converts the dot product of the two vectors into the desired output. (In other words, w is a one-form or linear functional mapping **x** onto R.) The weight vector **w** is learned from a set of labeled training samples. Often f is a simple function that maps all values above a certain threshold to the first class and all other values to the second class. A more complex f might give the probability that an item belongs to a certain class.

For a two-class classification problem, one can visualize the operation of a linear classifier as splitting a high-dimensional input space with a hyperplane: all points on one side of the hyperplane are classified as "yes", while the others are classified as "no".

An optimization algorithm that is given a training set with desired outputs and a loss function that measures the discrepancy between the classifier's outputs and the desired outputs. Thus, the learning algorithm solves an optimization problem of the form:

$$L = \frac{\lambda}{2}||W||^2 + \frac{1}{n}\sum_{i=1}^{n}\max(0, 1 - y_i(W^T x_i + b))$$

And the gradient with the respect of w:

$$G_W = \begin{cases} \lambda w & y_i(W^T x_i + b) \geq 1 \\ \lambda W + \frac{1}{n}\sum_{i=1}^{n} -y_i x_i & y_i(W^T x_i + b) < 1 \end{cases}$$

$$G_b = \begin{cases} 0 & y_i(W^T x_i + b) \geq 1 \\ \frac{1}{n}\sum_{i=1}^{n} -y_i & y_i(W^T x_i + b) < 1 \end{cases}$$

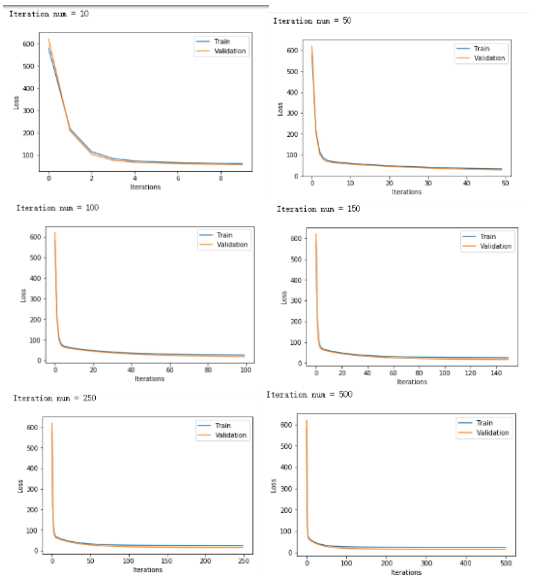
III. EXPERIMENT

A. Linear Regression

Experimental Code：

```
def Linear_regression(X_train, X_validation, y_train, y_validation, learning_rate,max_iteration):
    Loss_train = []
    Loss_validation = []
    b_train = 0.0
    W_train = np.zeros((1,X_train.shape[1]))
    b_validation = 0.0
    W_validation = np.zeros((1,X_validation.shape[1]))

    for i in range(max_iteration):
        Loss_train.append(np.sum((y_train - (X_train * np.transpose(W_train) + b_train)) ** 2) / X_train.shape[0])
        Loss_validation.append(np.sum((y_validation - (X_validation * np.transpose(W_validation)
                                + b_validation)) ** 2) / X_validation.shape[0])

        #update
        #D_b = - np.sum((y_train - (X_train * np.transpose(W) + b)))
        #D_W = - np.transpose(y_train - (X_train * np.transpose(W) + b)) * X_train
        #b = b - learning_rate * D_b
        #W = W - learning_rate * D_W
        D_b_train = np.sum(- 2 * (y_train - X_train * np.transpose(W_train) - b_train) / X_train.shape[0])
        D_W_train = - 2 * np.transpose(y_train - X_train * np.transpose(W_train) - b_train) * X_train / X_train.shape[0]
        b_train = b_train - (learning_rate * D_b_train)
        W_train = W_train - (learning_rate * D_W_train)

        D_b_validation = np.sum(- 2 * (y_validation - X_validation * np.transpose(W_validation) - b_validation) / X_validation.shape[0])
        D_W_validation = - 2 * np.transpose(y_validation - X_validation* np.transpose(W_validation)
                            - b_validation) * X_validation / X_validation.shape[0]
        b_validation = b_validation - (learning_rate * D_b_validation)
        W_validation= W_validation - (learning_rate * D_W_validation)

    print('Iteration num = %d'%(max_iteration))
    ax = plt.subplot()
    line_train = ax.plot(range(max_iteration), Loss_train, label='Train')
    line_validate = ax.plot(range(max_iteration), Loss_validation, label='Validation')
    ax.set(xlabel='Iterations', ylabel='Loss')
    plt.legend()
    plt.show()
return
```

Results:

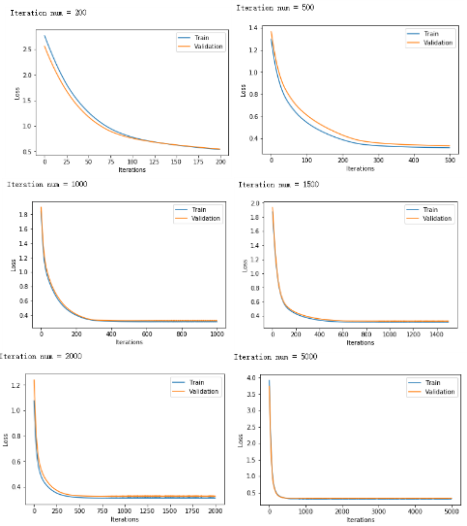


B. Linear Classification

Experimental Code：

```
learning_rate = 0.05
lamda = 0.05
def Linear_classification(X_train, X_validation, y_train, y_validation, learning_rate,max_iteration,lamda):
    num_records, num_features = np.shape(X)
    loss_train = []
    loss_validation = []
    W = np.random.normal(size=(num_features,1))

    for i in range(max_iteration):
        num_records_train, num_features_train = np.shape(X_train)
        num_records_validation, num_features_validation = np.shape(X_validation)
        #compute loss
        #Hinge loss = max(0, 1- y * (Wt * X ))
        #W = np.random.normal(loc = 0, scale = 1, size = (np.shape(X)))
        #hinge_loss_train = np.max[np.zeros(np.shape(X_train)), 1 - y_train * np.transpose(W_train) * X_train + b_train]
        #hinge_loss_validation = np.max[np.zeros(np.shape(X_validation)), 1 - y_validation * np.transpose(W_validation) * X_validation + b_validation]
        hinge_loss_train = 1.0 / float(num_records_train) * \
                    np.sum(np.max([np.zeros((num_records_train,1)),1 - y_train \
                    * (X_train.dot(W))],axis=0)) + 0.5 * lamda \
                    * W.transpose().dot(W)
        loss_train.append(hinge_loss_train[0][0])

        hinge_loss_validation = 1.0 / float(num_records_validation) * \
                    np.sum(np.max([np.zeros((num_records_validation,1)), \
                    1 - y_validation * (X_validation.dot(W))], axis = 0)) + 0.5 \
                    * lamda * W.transpose().dot(W)
        loss_validation.append(hinge_loss_validation[0][0])

        hinge_loss_gradient = np.max([np.zeros((num_records_train,1)),1 - y_train * (X_train.dot(W))],axis=0)
        indicator = np.zeros((num_records_train, 1))
        indicator[np.nonzero(hinge_loss_gradient)] = 1
        gradient_W = - 1.0 / float(num_records_train) * X_train.transpose().dot(y_train \
            * indicator).sum(axis = 1).reshape((num_features_train, 1)) + lamda * W
        W = W - learning_rate * gradient_W

    print('Iteration num = %d'%(max_iteration))
    ax = plt.subplot()
    line_train = ax.plot(range(max_iteration), loss_train, label = 'Train')
    line_validation = ax.plot(range(max_iteration), loss_validation, label = 'Validation')
    ax.set(xlabel = 'Iterations', ylabel = 'Loss')
    plt.legend()
    plt.show()
return
```

Results:



IV. CONCLUSION

A. Results analysis

1) If the learning rate is too small , using gradient descent to update the model will be very slow, as show in plot, the loss will converge after many iterations.
2) If the learning rate is too large, the updating step of gradient will be too large for the model and finally miss the minimal solution in this situation .
3) If the regularization parameter is too large, the proportion of the regularization term will be large and the model may be easy to fall into over fitting. On the other hand, if the regularization parameter is too small, the proportion of the regularization term will be small and the model may be easy to fall into under fitting.

B. Similarities and differences between linear regression and linear classification

Linear regression and linear classification both use the linear model. Both of them define a model to predict an output and compute the loss between real output, using gradient descent to update and adjust model to predict dataset better.

The most important different between regression and classification is that, regression model predict the continuous value of input data and classification model will classify the input date into two classes according to the attribute which is called discrete prediction.

Discrete Linear regression uses Least squared loss as the loss function, but linear classification updates the parameters by Hinge loss.