

**The Experiment Report of**

***Machine Learning***

**College Software College**

**Subject Software Engineering**

**Members**   Kenan Ye

**Student ID 201530613467**

**E-mail Conan.ye@gmail.com**

**Tutor**   **Mingkui Tan**

**Date submitted** **2017.12.07**

1. **Topic:**

Linear Regression, Linear Classification and Gradient Descent

1. **Time:**

2017-12-02 9:00-12:00 AM B7-138/238

1. **Reporter:**

Kenan Ye

1. **Purposes:**
2. Further understand of linear regression and gradient descent.
3. Conduct some experiments under small scale dataset.
4. Realize the process of optimization and adjusting parameters.
5. **Data sets and data analysis:**

Linear Regression uses Housing in LIBSVM Data, including 506 samples and each sample has 13 features. You are expected to download scaled edition. After downloading, you are supposed to divide it into training set, validation set.

Linear classification uses australian in LIBSVM Data, including 690 samples and each sample has 14 features. You are expected to download scaled edition. After downloading, you are supposed to divide it into training set, validation set.

1. **Experimental steps:**

Linear Regression and Gradient Descent

1. Load the experiment data. You can use load\_svmlight\_file function in sklearn library.
2. Divide dataset. You should divide dataset into training set and validation set using train\_test\_split function. Test set is not required in this experiment.
3. Initialize linear model parameters. You can choose to set all parameter into zero, initialize it randomly or with normal distribution.
4. Choose loss function and derivation: Find more detail in PPT.
5. Calculate gradienttoward loss function from all samples.
6. Denote the opposite direction of gradient as *D* .
7. Update model: .  *i*s learning rate, a hyper-parameter that we can adjust.
8. Get the loss under the training set and by validating under validation set.
9. Repeat step 5 to 8 for several times, and drawing graph of as well as with the number of iterations.

Linear Classification and Gradient Descent

1. Load the experiment data.
2. Divide dataset into training set and validation set.
3. Initialize SVM model parameters. You can choose to set all parameter into zero, initialize it randomly or with normal distribution.
4. Choose loss function and derivation: Find more detail in PPT.
5. Calculate gradienttoward loss function from all samples.
6. Denote the opposite direction of gradient as *D*.
7. Update model: .  *i*s learning rate, a hyper-parameter that we can adjust.
8. Select the appropriate threshold, mark the sample whose predict scores greater than the threshold as positive, on the contrary as negative. Get the loss under the training set and by validating under validation set.
9. Repeat step 5 to 8 for several times, and drawing graph of s as well as with the number of iterations.
10. **Code:**
11. Code of Classification (Only include gradient decent)

Figure Code of Classification (Model without b)

*# gradient decent*

*train\_loss = []*

*validation\_loss = []*

*acc\_rate = []*

*for i in range(iteration\_num):*

*# print current iteration number*

*print("\rProducing (" + str(i + 1) + "/" + str(iteration\_num) + ")", end="")*

*# define C*

*C = 0.9*

*# calculate gradient*

*gradient = np.zeros(15)*

*gradient\_of\_w = np.zeros(14)*

*gradient\_of\_b = 0*

*for cur\_x, cur\_y in zip(X\_train, y\_train):*

*gradient\_of\_w = gradient\_of\_w + C \* partial\_derivative\_of\_w(cur\_x.toarray(), cur\_y)*

*gradient\_of\_b = gradient\_of\_b + C \* partial\_derivative\_of\_b(cur\_x.toarray(), cur\_y)*

*gradient\_of\_w = omega[:14] + gradient\_of\_w*

*gradient = np.append(gradient\_of\_w, gradient\_of\_b)*

*# gradient divided by number of samples*

*for j in range(len(gradient)):*

*gradient[j] = gradient[j] / y\_train.size*

*# update parameters*

*omega = omega - eta \* gradient*

Figure Code of Regression (Model with b)

*# gradient decent*

*train\_loss = []*

*validation\_loss = []*

*for i in range(iteration\_num):*

*# print current iteration number*

*print("\rProducing (" + str(i + 1) + "/" + str(iteration\_num) + ")", end="")*

*# calculate gradient*

*gradient = np.zeros(14)*

*for curr\_x, curr\_y in zip(X\_train, y\_train):*

*curr\_x = curr\_x.toarray()*

*# calculate Partial derivatives*

*for j in range(13):*

*gradient[j] = gradient[j] + (model\_function(curr\_x) - curr\_y) \* curr\_x[0][j]*

*gradient[13] = gradient[13] + (model\_function(curr\_x) - curr\_y)*

*# average gradient of samples*

*for j in range(len(gradient)):*

*gradient[j] = gradient[j] / y\_train.size*

*# update parameters*

*omega = omega - eta \* gradient*

1. Code of Regression (Only include gradient decent)

Figure Code of Regression

*# gradient decent*

*train\_loss = []*

*validation\_loss = []*

*for i in range(iteration\_num):*

*# print current iteration number*

*print("\rProducing (" + str(i + 1) + "/" + str(iteration\_num) + ")", end="")*

*# gradient = -X^T \* y + X^T \* x \* w*

*# = - A \* y + B \* w*

*# = - A \* y + A \* X \* w*

*# calculate A*

*A = X\_train.toarray().T*

*# calculate B*

*B = np.dot(A, X\_train.toarray())*

*# calculate gradient*

*# gradient = np.dot(B, theta) - np.dot(A, y\_train)*

*gradient = np.subtract(np.dot(B, omega), np.dot(A, y\_train))*

*# gradient divided by number of samples*

*for j in range(len(gradient)):*

*gradient[j] = gradient[j] / y\_train.size*

*# decent function, update parameters*

*omega = omega - eta \* gradient*

1. **Selection of validation (hold-out, cross-validation, k-folds cross-validation, etc.):**

Both experiment use hold-out as validation method.

1. **The initialization method of model parameters:**

Both experiment initialize parameters with Zeros

1. **The selected loss function and its derivatives:**
2. Linear Regression:

* Model without b

Loss function:

Derivative of loss function:

* Model with b

Loss function:

Derivative of loss function:

1. Linear Classification:

For getting best classification performance in unseen data set, we use support vector machine to do the linear classification (SVM), which selects two parallel hyperplanes that separate the two classes of data and let the distance between them as large as possible. The region bounded by these two hyperplanes is called the "margin"

Margin:

Therefore, learning the SVM can be formulated as an optimization:

However, data set given might, and usually certainly, have noises. If just maximum the margin the model will overfitting. In general, there is a trade-off between the margin and the number of mistakes on the training data.

Therefore, we introduce , for each , which represents how much example is on wrong side of margin boundary.

* + - If then it is ok.
    - If it is correctly classified, but with a smaller margin than
    - If then it is in correctly classified.

The optimization problems become:

Hinge loss =

The optimization problems become

Gradient of optimization function:

Therefore:

Let , so:

Let , so:

Consequently,

1. **Experimental results and curve:**

## Assessment Results (based on selected validation) and Hyper-parameter selection (η, epoch, etc.):

1. Linear Regression:

Selected:

Assessment:

iteration number = 100

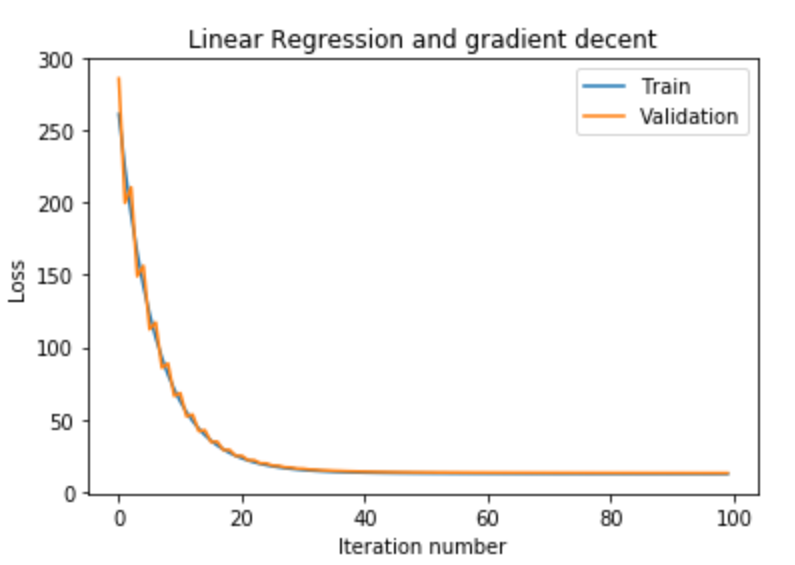


Figure Loss with iteration number(η= 0.5, iteration number =100)

iteration number = 100

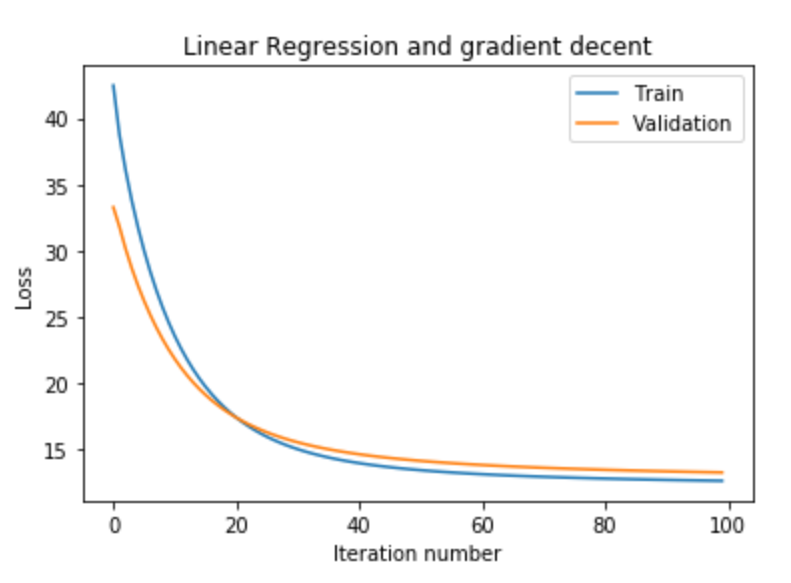


Figure Loss with iteration number(η= 0.25, iteration number =100)

iteration number = 200

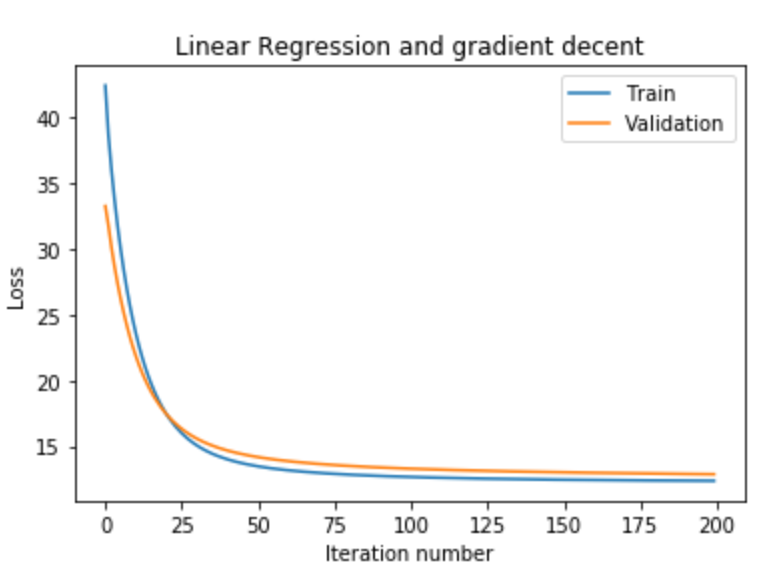


Figure Loss with iteration number(η= 0.25, iteration number =200)

* Model with b

iteration number = 200

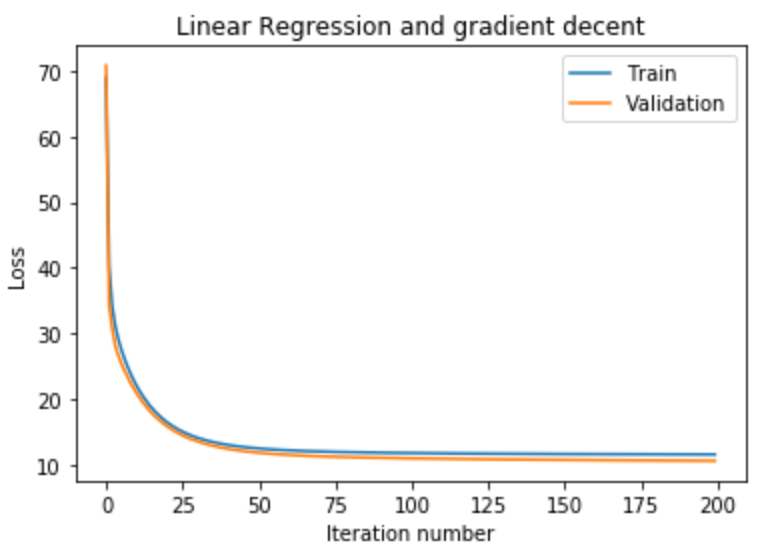


Figure Loss with iteration number(η= 0.25, iteration number =200)

1. Linear Classification:

Selected: iteration number = 200

Assessment:

iteration number = 200

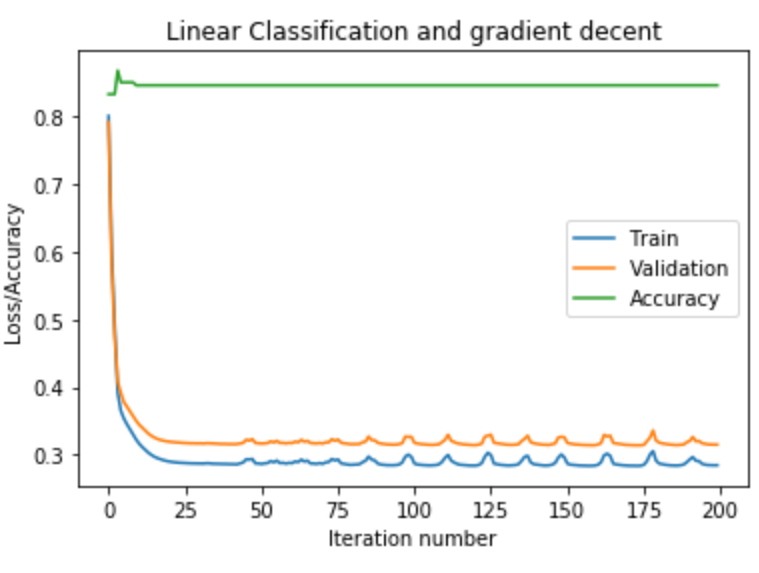


Figure Loss and Acuracy (η=0.25, C=0.9, iteration number = 200)

iteration number = 200

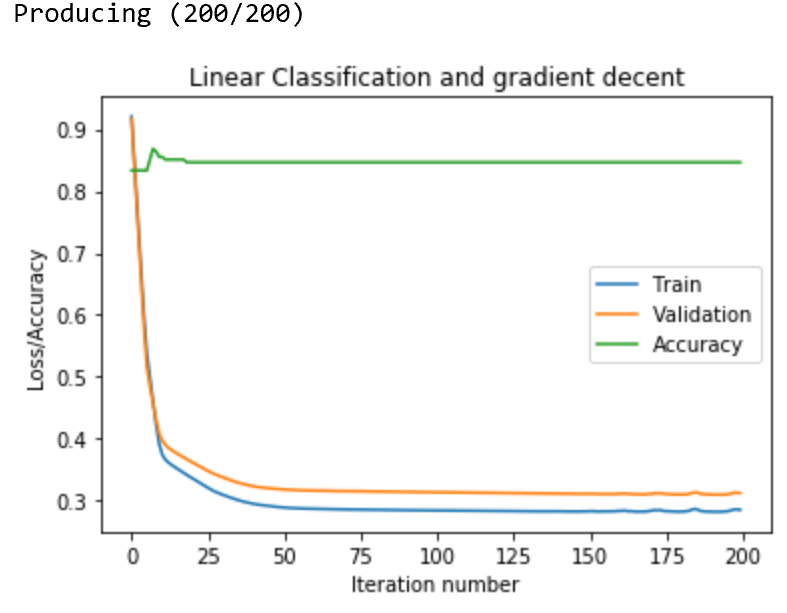


Figure Loss and Acuracy (η=0.1, C=0.9, iteration number = 200)

## Predicted Results (Best Results) and Loss curve:

1. **Results analysis:**

**13. Similarities and differences between linear regression and linear classification:**

**14. Summary:**