The Experiment Report of Machine Learning



**SUBJECT:**SOFTWARE ENGINEERING

December 14, 2017

Grade:

Post Graduate

Student ID：

201721045893

Supervisor:

 Qingyao Wu

Author:

Shengyan Wen

**SCHOOL:** SCHOOL OF SOFTWARE ENGINEERING

[[1]](#footnote-0)

Linear Regression, Linear Classiﬁcation and Gradient Descent

Abstract— **In this experiment, we aim to further understand the linear regression and gradient descent. We conduct some experiments under small scale data sets, where 'Housing' is used for linear regression and gradient descent and ‘australian’ is used for linear classification and gradient descent. The best results of linear regression shows that validation loss is near to 11.3 while the learning rate is 0.2. The best results of linear classification shows that validation loss is near to 0.53 and its accuracy reaches 0.85 while the learning rate is 0.21 and regularization parameter is 0.5.**

# INTRODUCTION

The motivation of the experiment is further understanding of linear regression, linear classification and gradient descent. Linear regression uses 'Housing' in LIBSVM Data, including 506 samples and each sample has 13 features. Linear classification uses 'australian' in LIBSVM Data, including 690 samples and each sample has 14 features.

All the experiment steps is as follows. After downloading, we divided data sets into training set, validation set firstly. Secondly we initialize linear model parameters. After that we set all parameter into zero and initialize it with normal distribution.

Thirdly, we defined the loss function of the linear regression to be least squared loss, and defined the loss function of the linear classification to be hinge loss. Fourthly compute the gradient of the loss function with respect to the weight W and bias b. Fifthly update the parameters W and b.

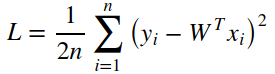
Then repeat above steps for several times until convergence.

While doing experiment, we could realize the process of optimization and adjust parameters to find the best case.

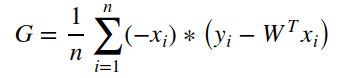
# METHODS AND THEORY

*A)Linear regression and gradient descent*

We defined the loss function of the linear regression to be least squared loss:



The gradient of the loss function is:



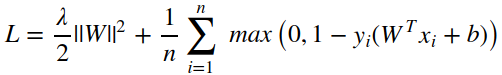
Update the parameter W use:



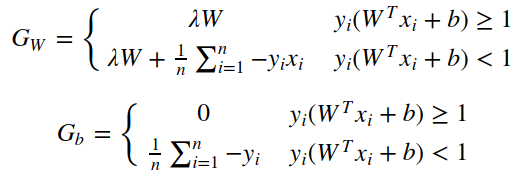
Where η is the pre-defined learning rate.

*B)Linear classification and gradient descent*

We defined the loss function of the linear classification to be Hinge loss:



The gradient of the loss function is:

Update the parameters W and b:

Where η is the learning rate, λ is the regularization parameter.

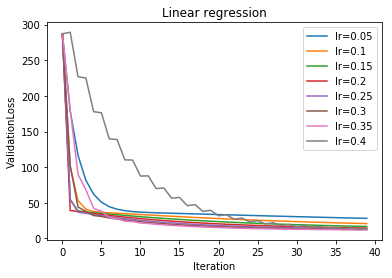
# Experiment

The code is as follows.

1. *Linear regression and gradient descent*

|  |
| --- |
| from sklearn.externals.joblib import Memory  from sklearn.datasets import load\_svmlight\_file  from sklearn.datasets import load\_svmlight\_file  mem = Memory("./mycache")  @mem.cache  #load data  def get\_data():  data = load\_svmlight\_file("housing\_scale.txt")  return data[0], data[1]  X, y = get\_data()  X = X.toarray()  #X = [X,1]  import numpy as np  addone= np.ones(X.shape[0])  X= np.column\_stack((X,addone))  #divide data to traning part and validation part  from sklearn.model\_selection import train\_test\_split  from numpy import random  X\_train, X\_validation, y\_train, y\_validation = train\_test\_split(X, y, test\_size=0.22, random\_state=25)  In [251]:  N = X\_train.shape[1]  W\_zeros = np.zeros(N)  W\_random = random.random(size=N)  #use NumPy random.normal fuction to get datas in normal distribution  W\_normal = np.random.normal(size=N)  In [252]:  #Choose Least squared loss function  def cal\_Loss(X,W,y):  preY = np.dot(X,W)  diifY = y - preY  Loss = np.dot(diifY,diifY.T)/(2 \* X.shape[0])  return Loss  #Calculate the gradient  def cal\_G(X,W,y):  preY = np.dot(X,W)  diifY = y - preY  G = - np.dot(diifY,X)/ X.shape[0]  return G  def draw\_plot(Loss\_train,Loss\_validation):  plt.plot(Loss\_train,label="Loss\_train")  plt.plot(Loss\_validation,label="Loss\_validation")  plt.legend()  plt.xlabel("Iteration")  plt.ylabel("Loss")  plt.title("Linear regression")  plt.show()  In [253]:  lr = 0.15  iteration = 100  W = W\_normal  Loss\_train = np.zeros(iteration)  Loss\_validation = np.zeros(iteration)  for j in range(0,iteration):  #the training loss  Loss\_train[j] = cal\_Loss(X\_train,W,y\_train)  #the gradient of the loss function  G = cal\_G(X\_train,W,y\_train)  #the validation loss  Loss\_validation[j] = cal\_Loss(X\_validation,W,y\_validation)  #update the parameter W  W = W - G \* lr  #draw the result  draw\_plot(Loss\_train,Loss\_validation) |

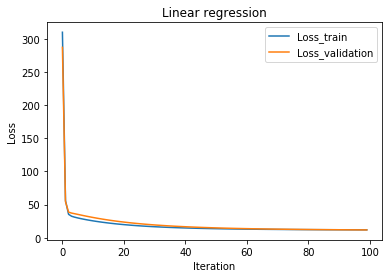
We chose 8 different learning rates [0.05, 0.1, ..., 0.4] to optimize the linear regression algorithm, the results are as follows：



If the learning rate is too small, the decline of the curve is slow and the number of the iteration to reach convergence will be large.If the learning rate is too large, the loss curve will be oscillating.

The best learning rate is 0.2, which is illustrated as the red line.

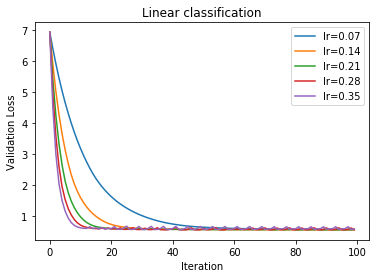
The best result is shown as follows, with 11.29208 as the validation loss.



*B)Linear classification and gradient descent*

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  from numpy import random  from sklearn.externals.joblib import Memory  from sklearn.datasets import load\_svmlight\_file  from sklearn.model\_selection import train\_test\_split  mem = Memory("./mycache2")  @mem.cache  #load data  def get\_data():  data = load\_svmlight\_file("australian\_scale.txt")  return data[0], data[1]  X, y = get\_data()  X = X.toarray()  #X = [X,1]  addone= np.ones(X.shape[0])  X= np.column\_stack((X,addone))  In [154]:  #divide data to traning part and validation part  from sklearn.model\_selection import train\_test\_split  from numpy import random  X\_train, X\_validation, y\_train, y\_validation = train\_test\_split(X, y, test\_size=0.25, random\_state=25)  In [155]:  N=X\_train.shape[1]  #Normal distribution initialized  W\_nor = np.random.normal(size=N)  print(W.shape)  print(W)  #calculate the loss  def cal\_Loss(X,W,y,lambdal,W\_0):  preY = np.dot(X,W)  diifY = np.ones(y.shape[0]) - y \* preY  diifY[diifY < 0] =0  Loss =np.sum(diifY) / X.shape[0] + np.dot(W\_0,W\_0.T)/2\*lambdal  return Loss  #calculate the gradient  def cal\_G(X,W,y,lambdal,W\_0):  preY = np.dot(X,W)  diifY = np.ones(y.shape[0]) - y \* preY  y\_get = y.copy()  y\_get[diifY <= 0] =0  G = -np.dot(y\_get,X) / X.shape[0] + W\_0 \*lambdal  return G  #calculate the accuracy  def cal\_Accuracy(X,W,y):  preY = np.dot(X,W)  count = np.sum(preY \* y >0)  Accuracy = count / X.shape[0]  return Accuracy  def draw\_plot(Loss\_train,Loss\_validation,Accuracy):  fig = plt.figure()  ax1 = fig.add\_subplot(111)  ax1.plot(Loss\_train,label="Loss\_train")  ax1.plot(Loss\_validation,label="Loss\_validation")  ax1.set\_ylabel("Loss")  ax1.set\_xlabel("Iteration")  ax1.legend()  ax2 = ax1.twinx()  ax2.plot(Accuracy,label="Accuracy",color = 'r')  ax2.legend()  ax2.set\_ylabel("Accuracy")  ax2.set\_title("Linear classification")  plt.show()  plt.close()  lr = 0.21  lambdal = 0.5  iteration = 400  #get different kinds of initial data（W\_zeros,W\_random or W\_normal）  W = W\_nor  Loss\_train = np.zeros(iteration)  Loss\_validation = np.zeros(iteration)  Accuracy = np.zeros(iteration)  for j in range(0,iteration):  W\_0 = W.copy()  W\_0[N-1]= 0  #the training loss  Loss\_train[j] = cal\_Loss(X\_train,W,y\_train,lambdal,W\_0)  #the gradient of the loss function  G = cal\_G(X\_train,W,y\_train,lambdal,W\_0)  #the validation loss  Loss\_validation[j] = cal\_Loss(X\_validation,W,y\_validation,lambdal,W\_0)  #accuracy  Accuracy[j] = cal\_Accuracy(X\_validation,W,y\_validation)  #update the parameter W  W = W - G \* lr  #draw the result  draw\_plot(Loss\_train,Loss\_validation,Accuracy)  plt.close() |

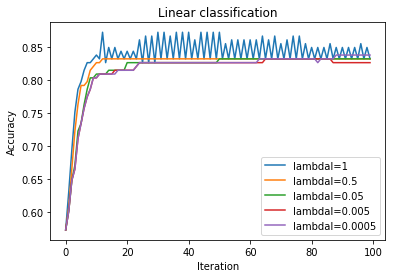
We chose 5 different learning rates [0.07, 0.14, ..., 0.35] to optimize the linear regression algorithm with specific regularization parameter 0.5, the results are shown as follows：



If the learning rate is too small, the decline of the curve is slow and the number of the iteration to reach convergence will be large.If the learning rate is too large, the loss curve will be oscillating.

The best learning rate is 0.21, which is illustrated as the green line.

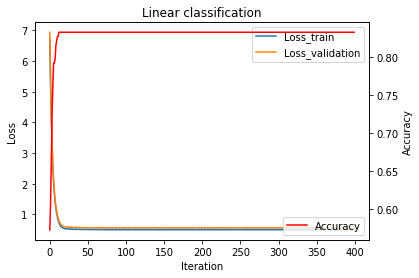
We chose 5 different regularization parameter [1, 0.5, 0.05, 0.005, 0.0005] to optimize the linear regression algorithm, the results are shown as follows:



If the regularization parameter is too small, the model might be over-fitting.If the regularization parameter is too large, the model may be under-fitting.

The best regularization parameter is 0.5, which is illustrated as the yellow line.

The best result is shown as follows, with 0.527841 as the validation loss and 0.84826 as the accuracy.



# conclusion

There are two kinds of loss function for the linear regression and classification. Linear regression uses least squared loss, while linear classification updates the parameters by Hinge loss.

For the linear regression, we learn a model to simulate the mapping between input X and output y. We compare the validation loss with the train loss to evaluate the model. We find that the best learning rate is 0.2 where the decline of curve is quickly without oscillating.

For the linear classification task, we find a hyper plane to separate the different target. We evaluate the model by calculating the accuracy and comparing the validation loss with the train loss. We find the best learning rate is 0.21 where the decline of curve is quickly without oscillation. And we also find that the best regularization parameter is 0.5 where the rise of curve is quickly without oscillation.

1. [↑](#footnote-ref-0)