The Experiment Report of Machine Learning



**SUBJECT:**SOFTWARE ENGINEERING

December 14, 2017

Grade:

Postgraduate

Student ID：

201721045909

Supervisor:

Qingyao Wu

Author:

Li Zhang

**SCHOOL:** SCHOOL OF SOFTWARE ENGINEERING

[[1]](#footnote-1)Linear Regression, Linear Classiﬁcation and Gradient Descent

Abstract—Linear regression and linear classification are the basic algorithm in the field of machine learning. Gradient Descent is an effective way to find the good model parameters for above algorithm. The experiments indicate that the algorithms are effective.

# INTRODUCTION

Linear regression and linear classification are the basic algorithm in the field of machine learning. Gradient Descent is an effective way to find the good model parameters for above algorithm. This report talks about the experiment of linear regression and linear classification. Besides, the gradient descent is used to find the good parameters.

The rest paper is organized as follow. Section II contains the experiment steps. Section III describes the code and result for the two experiments. Section IV concludes the report.

# METHODS AND THEORY

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.

Then the experiment will be performed by the following steps:

1. Download the dataset to local host machine.
2. Load dataset into memory.
3. Split dataset into training set and validation set.
4. Create and fill necessary data structures.
5. Write functions for calculating loss and gradient (different in regression and classification).
6. Set parameters (learning rate and the number of iterations).
7. Initiate weights (using normal distribution).
8. Calculate the gradient and update weights according to the gradient calculated in each iteration.
9. Change parameters and run again.
10. Draw plot for experiment result.

# Experiment

1. Codes of experiments

The codes of two experiments are shown as follow,

1. Linear Regression

|  |
| --- |
| #linear regression  import numpy as np  import matplotlib.pyplot as plt  from sklearn.datasets import load\_svmlight\_file  from sklearn.model\_selection import train\_test\_split  #read the data  housing\_path = './housing\_scale.txt'  X, y = load\_svmlight\_file(housing\_path)  #add 1 for each row  X = X.toarray()  row,col=X.shape  X=np.column\_stack((X,np.ones(row)))  #divide data into training and validation  X\_train, X\_validation, y\_train, y\_validation = train\_test\_split(X, y, test\_size=0.2, random\_state=30)  #initialize by zeros  W = np.zeros(col+1)  #define learn rate and itreation, loss array  learn\_rate = 0.02  iteration = 500  loss\_train = np.zeros(interation)  loss\_validation = np.zeros(interation)  #loss function  def get\_loss(X, W, y):  diff = y - np.dot(X, W)  loss = np.dot(diff,diff.T) / (2 \* X.shape[0])  return loss  #gradient function  def get\_gradient(X, W, y):  diff = y - np.dot(X, W)  G = - np.dot(diff,X)/ X.shape[0]  return G  #start interation  for i in range(iteration):  # get loss  loss\_train[i] = get\_loss(X\_train, W, y\_train)  loss\_validation[i] = get\_loss(X\_validation, W, y\_validation)    #get gradient and update W  G = get\_gradient(X\_train, W, y\_train)  W = W - learn\_rate \* G  #draw the result  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()  print(loss\_train[iteration-1])  print(loss\_validation[iteration-1]) |

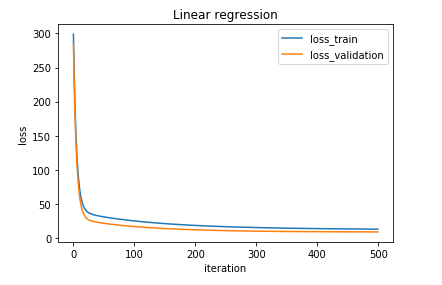
1. Linear Classification

|  |
| --- |
| #linear classification  import numpy as np  import matplotlib.pyplot as plt  from sklearn.datasets import load\_svmlight\_file  from sklearn.model\_selection import train\_test\_split  from sklearn.model\_selection import train\_test\_split  #read the data  housing\_path = './australian\_scale.txt'  X, y = load\_svmlight\_file(housing\_path)  #add 1 for each row  X = X.toarray()  row,col=X.shape  X=np.column\_stack((X,np.ones(row)))  #divide data into training and validation  X\_train, X\_validation, y\_train, y\_validation = train\_test\_split(X, y, test\_size=0.2, random\_state=30)  #initialize by zeros  W = np.zeros(X\_train.shape[1])  #define learn rate, itreation, lambdal, loss array, accuracy array  learn\_rate = 0.05  iteration = 500  lambdal = 0.01  loss\_train = np.zeros(iteration)  loss\_validation = np.zeros(iteration)  accuracy = np.zeros(iteration)  #loss function  def get\_loss(X, W, y, lambdal, W\_0):  diff = np.ones(y.shape[0]) - y \* np.dot(X, W)  diff[diff < 0] = 0  loss = np.sum(diff) / X.shape[0] + np.dot(W\_0,W\_0.T)/2\*lambdal  return loss  #gradient function  def get\_gradient(X, W, y, lambdal, W\_0):  diff = np.ones(y.shape[0]) - y \* np.dot(X, W)  y\_get = y.copy()  y\_get[diff <= 0] = 0  G = -np.dot(y\_get,X) / X.shape[0] + W\_0 \* lambdal  return G  #accuracy function  def get\_accuracy(x, W, y):  preY = np.dot(X,W)  count = np.sum(preY \* y >0)  Accuracy = count / X.shape[0]  return Accuracy  #start iteration  for i in range(iteration):  W\_0 = W.copy()  W\_0[col-1]= 0  # get loss  loss\_train[i] = get\_loss(X\_train, W, y\_train, lambdal, W\_0)  loss\_validation[i] = get\_loss(X\_validation, W, y\_validation, lambdal, W\_0)    #get accuracy  #accuracy[i] = get\_accuracy(X\_validation, W, y\_validation)    #get gradient and update W  G = get\_gradient(X\_train, W, y\_train, lambdal, W\_0)  W = W - learn\_rate \* G  #draw the result  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 classification")  plt.show()  print(loss\_train[iteration-1])  print(loss\_validation[iteration-1]) |

1. Experiment results

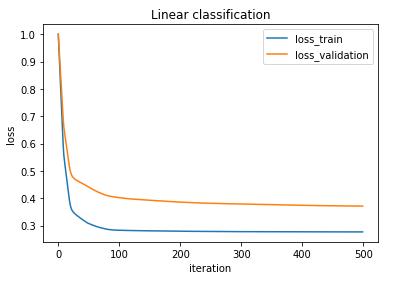
1) Linear Regression

In this set of experiments, we set the learning rate to 0.02 and iteration number to 500. The result is shown in the following figure. The validation loss can get to 9.08 after the iteration.



2)Linear Classification

In this set of experiments, we set the learning rate to 0.05 and iteration number to 500. The result is shown in the following figure. The validation loss can get to 0.37 after the iteration.



# conclusion

After the experiments, we know the parameters such as learning rate and iteration are very important. If the iteration number is less, the loss of the model would be high. If the learning rate is too high, the loss function would become not convergent.

1. [↑](#footnote-ref-1)