South China University of Technology

The Experiment Report of

Machine Learning

College Software College

Subject Software Engineering

Student ID 201722800100

E-mail Az.yamen@gmail.com

Tutor Mingkui Tan

Date submitted 2017.12 .08

Linear Regression, Linear Classification and Gradient Descent

Time: 2017-12-08

Reported By:

Ezzaddin Ahmed Othman Saeed

Purposes:

- Use data set to identify relationships among variables and use these relationships to make predictions(Regression).
- To estimate likely performance of a model on out-of-sample data.
- To realize the process of optimization and adjusting parameters and further understand of liner regression and gradient descent.

Data sets and data analysis:

In this work, I used two LIBSVM datasets which are pre-processed data originally from UCI data repository.

- *Data set 1:* Housing (Boston) :

  - (data: 506 , features: 13):

Linear Regression uses (Housing) data set in LIBSVM Data, including 506 samples and each sample has 13 features.

This data set contains information about the housing values in suburbs of Boston.

- *Data set 2* : Australian

  - (classes: 2 , data: 690 of features: 14)

Linear classification uses (Australian) data set in LIBSVM Data, including 690 samples and each sample has 14 features.

Experimental steps:

Experiment 1 : (Linear Regression and Gradient Descent)

step 1: Load the experiment data. You can use load_svmlight_file function in sklearn library.

step 2: Divide dataset. Here I divide dataset into training set and validation set using train_test_split function. Of course Test set is not required in this experiment.

step 3: Initialize linear model parameters. Here I choose normal distribution to initialize the model .

step 4: Choose loss function and derivation: Find more detail in PPT.

step 5: from all samples I calculate gradient toward loss function.

step 6: Denote the opposite direction of gradient as G -> D.

step 7: Update model.

step 8: Get the loss under the training set and by validating under validation set.

step 9: Repeat step 5 to 8 for several times, and drawing graph of as well as with the number of iterations.

Experiment 2 : (Linear Classification and Gradient Descent)

step 1: Load the experiment data.

step 2: Divide dataset into training set and validation set by using train_test_split function .

step 3: Initialize SVM model parameters with normal distribution.

step 4: I identify loss function and derivation.

step 5: Calculate gradient toward loss function from all samples.

step 6: Denote the opposite direction of gradient G as D .

step 7: Update model.

step 8: Get the loss under the train in set and by validating under validation set.

step 9: repeat step 5 to 8 for several times, and drawing graph of as well as with the number of iterations.

Code:

For linear regression :

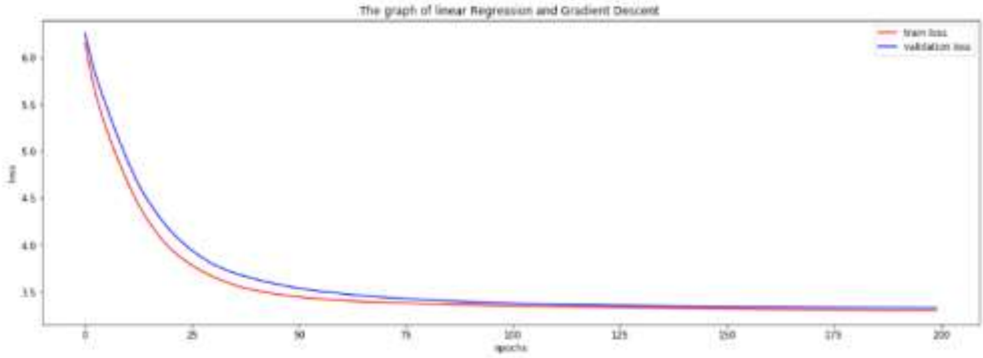
```
#This code  Done By:
#           Ezzaddin Ahmed Othman Saeed
#                  Taiz-Yemen
#-------------------------------------------------

from sklearn.externals.joblib import Memory

#    STEP 1:

# Load the experiment data by import load_svmlight_file function in sklearn library

from sklearn.datasets import load_svmlight_file
mem = Memory("./mycache")

mem.cache
def get_data():
    data = load_svmlight_file("./datasets/housing_scale")

    return data

X, Y = get_data()
X = X.data.reshape(X.shape)


##################################       #    STEP 2:   #          ########################################
# devide the dataset into traning set && validation set by train_test_split function

                              #  ===>> validation set isn't requier here  <===

from sklearn.model_selection import train_test_split
X_train, X_validation, Y_train, Y_validation = train_test_split(X, Y, test_size=0.5)
#************************************
print(X_train.shape,type(X_train)) #
#************************************

##################################
```

```
#There are several ways to implement a linear regression
import numpy # here I used numpy  To do linear regression

n_samples_train, n_features = X_train.shape ### n_samples_train is trainning sample & n_features is number of features

X_train = numpy.concatenate((X_train,numpy.ones((n_samples_train,1))),axis=1)
Y_train = numpy.reshape(Y_train,(n_samples_train,1))

n_samples_validation, _ = X_validation.shape
X_validation = numpy.concatenate((X_validation,numpy.ones((n_samples_validation,1))),axis=1)
Y_validation = numpy.reshape(Y_validation,(n_samples_validation,1))

factor = 0.5
learning_rate = 0.0008


    # #  # STEP 3:

              ######################## Initialize linear model parameters ########################

W = numpy.random.normal(0.01,0.1,size=(n_features + 1,1)) # initialize with zero normal distribution

losses_train = []
losses_val = []

max_loop = 200 ## total Iterations - No. Loops=200

for epoch in range(max_loop):
    diff = numpy.dot(X_train,W) - Y_train
    G = factor * W + numpy.dot(X_train.transpose(),diff) # calculate the gradient
    G = -G
    W = W + learning_rate * G # update the parameters

    Y_predict = numpy.dot(X_train,W) # predict under the train set
    loss_train = numpy.average(numpy.abs(Y_predict-Y_train)) # calculate the absolute differences
    losses_train.append(loss_train)
```

```
    %matplotlib inline
import matplotlib.pyplot as plt

epoches = range(len(losses_train))

plt.figure(figsize=(18,6))
plt.plot(epoches,losses_train,"-",color="r",label="train loss")
plt.plot(epoches,losses_val,"-",color="b",label="validation loss")
plt.xlabel("epochs")
plt.ylabel("loss")
plt.legend()
plt.title("The graph of linear Regression and Gradient Descent ")
```

```
(253, 13) <class 'numpy.ndarray'>
```

Out[9]: `Text(0.5,1,'The graph of linear Regression and Gradient Descent ')`



For linear classification

```
#This code  Done By:
#            Ezzaddin Ahmed Othman Saeed
#                  Taiz-Yemen
#---------------------------------------------------

from sklearn.externals.joblib import Memory

 #    STEP 1:

     # Load the experiment data.
from sklearn.datasets import load_svmlight_file

mem = Memory("./mycache")

mem.cache
def get_data():
    data = load_svmlight_file("./datasets/australian_scale")

    return data

X, y = get_data()
X = X.toarray()
```

```
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.25, random_state=42)
```

```python
import numpy
n_samples_train,n_featues_train = X_train.shape


def svm(X_train,y_train,W):
    g = numpy.ones([n_featues_train,1])
    for i in range(n_samples_train):
        X = X_train[i].reshape(1,n_featues_train)
        y = y_train[i].reshape(1,1)
        h = 1-numpy.multiply(y,numpy.dot(X,W))
        if(h >= 0):
            g += W-numpy.dot(X.T,y)
        else:
            g += W

    W = W - lr * g / n_samples_train
    return W

```

```python
def svm_loss(X_train,y_train,f,W):
    n_samples_train,n_featues_train = X_train.shape
    loss = 0
    for i in range(n_samples_train):
        X =X_train[i]
        y =y_train[i]
        h = 1-numpy.multiply(y,numpy.dot(W.T,X))
        if(h >= 0):
            loss += f*h
    return loss/n_samples_train+numpy.dot(W.T,W)/2
```

```python
f = 0.9
#learning rate
lr = 0.01
losses_train = []
losses_val = []
total_loop = 100
n_samples_train,n_featues_train = X_train.shape
W = numpy.ones((n_featues_train,1))

for i in range(total_loop):

    W=svm(X_train,y_train,W)
    loss=svm_loss(X_train,y_train,f,W)
    losses_train.append(loss[0][0])
    loss=svm_loss(X_val,y_val,f,W)
    losses_val.append(loss[0][0])

```

```python
import matplotlib.pyplot as plt

plt.figure(figsize=(16,9))
plt.plot(losses_train,color="r",label="train loss")
plt.plot(losses_val,color="b",label="validation loss")
plt.legend()
plt.xlabel("epoch")
plt.ylabel("loss")
plt.title("Linear classification & Gradient Descent")
plt.show()
```

The initialization method of model parameters:

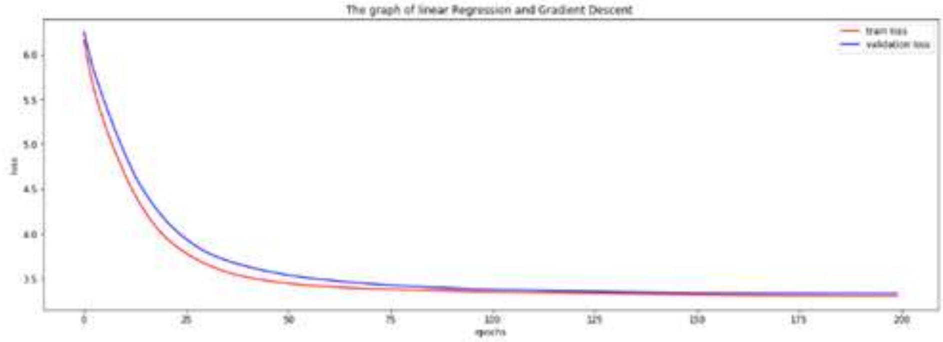
```python
factor = 0.5
learning_rate = 0.0008


    # #  # STEP 3:

                ######################### Initialize linear model parameters #########################

W = numpy.random.normal(0.01,0.1,size=(n_features + 1,1)) # initialize with zero normal distribution

losses_train = []
losses_val = []

max_loop = 200 ## total Iterations - No. Loops=200

for epoch in range(max_loop):
    diff = numpy.dot(X_train,W) - Y_train
    G = factor * W + numpy.dot(X_train.transpose(),diff) # calculate the gradient
    G = -G
    W = W + learning_rate * G # update the parameters

    Y_predict = numpy.dot(X_train,W) # predict under the train set
    loss_train = numpy.average(numpy.abs(Y_predict-Y_train)) # calculate the absolute differences
    losses_train.append(loss_train)

    Y_predict = numpy.dot(X_validation,W) # predict under the validation set
    loss_val = numpy.average(numpy.abs(Y_predict-Y_validation)) # calculate the absolute differences
    losses_val.append(loss_val)

    ###########################################

    %matplotlib inline
import matplotlib.pyplot as plt

epoches = range(len(losses_train))
```

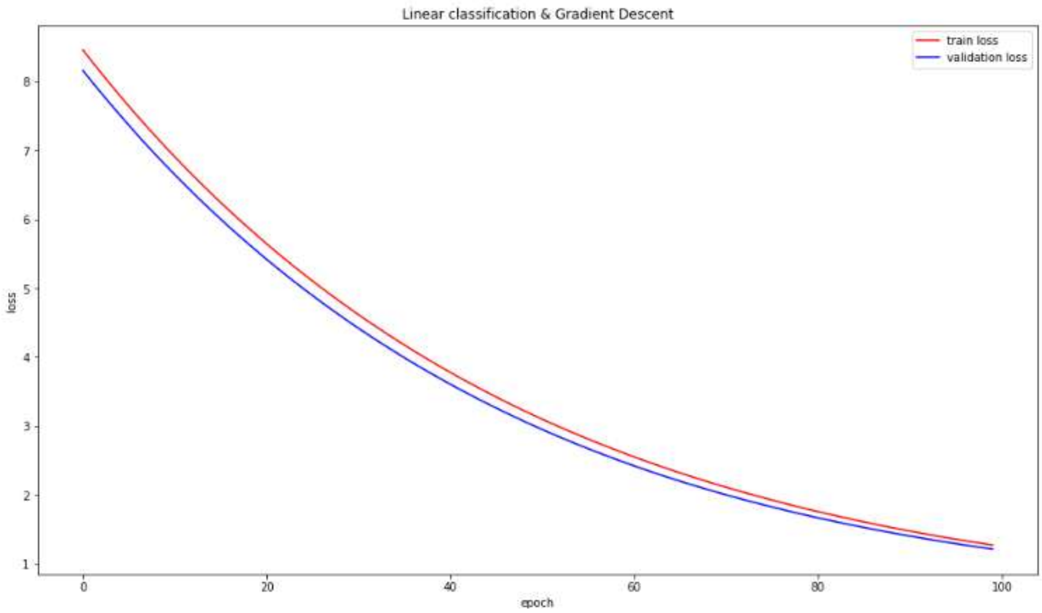
Experimental results and curve:

Hyper-parameter selection (η , epoch, etc.): η = 0.0008 and 0.01 , epoch=200 and 100

Assessment Results (based on selected validation):

```
Out[9]: Text(0.5,1,'The graph of linear Regression and Gradient Descent ')
```





Results analysis:

From the graph its easily understandable that the train curve and the test curve are almost same.

Similarities and differences between linear regression and linear classification:

Type	**linear regression**	**linear classification**
type of output	Continuous	discrete
Category	supervised	supervised

Regression is used to predict **continuous values**. Classification is used to predict which class a data point is part of (**discrete value**).

Both regression and classification problems belong to the supervised category of machine learning. In Supervised machine learning, a model or a function is learnt from the data to predict the future data. In simple terms, y=f(x) is a predictive model learnt from the data set D = {(X1,y1),...(Xn,y2} where X is the input vector and y is the output.

Based on the type of output y, the learning problems are classified into regression and classification. In case of classification, the output variable is discrete and in regression, the output variable is continuous.

Summary:

In this Report, after apply Experiment, we understand similarity and differences between linear regression and gradient descent through conduct some experiments under small scale dataset. In Experiment 1 we use regression for predicting housing prices in the boston dataset present in the sklearn datasets .