South China University of Technology

The Experiment Report of Machine Learning

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
Chao Li

Supervisor:
Qingyao Wu

Student ID：
201821039158

Grade:
Graduate

December 24, 2018

Linear Regression, Linear Classification and Gradient Descent

Abstract—The experimental report mainly introduces the Housing Data set in LIBSVM Data with the linear regression algorithm and the a9a Data set in LIBSVM Data with the linear classification algorithm.
The experiment showed the process of random initialization and zero initialization of parameters, updating parameters through SGD and batch -SGD, as well as the selection and debugging of various super parameters.
Further deepen the understanding of regression algorithm and classification algorithm principle.

I. Introduction

Experient1—Linear Regression and Stochastic Gradient Descent

Motivation of Experiment
1.Further understand of linear regression，closed-form solution and Stochastic gradient descent.
2.Conduct some experiments under small scale dataset.
3.Realize the process of optimization and adjusting parameters.

Experient2—Logistic Regression and Support Vector Machine

Motivation of Experiment
1.Compare and understand the difference between gradient descent and batch random stochastic gradient descent.
2.Compare and understand the differences and relationships between Logistic regression and linear classification.
3.Further understand the principles of SVM and practice on larger data.

II. Methods and Theory

Experient1—Linear Regression and Stochastic Gradient Descent

Experiment Step
*closed-form solution of Linear Regression *
1.Load the experiment data. I used load_svmlight_file function in sklearn library.
2.Devide dataset. I divided dataset into training set and validation set using train_test_split function.
3.Initialize linear model parameters. I choose to set all parameter into zero.
4.Select a Loss function and calculate the value of the Loss function of the training set,denoted as .
Loss function:

$$L(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^\top(\mathbf{y} - \mathbf{X}\mathbf{w})$$

Derivation:

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} = -\mathbf{X}^\top\mathbf{y} + \mathbf{X}^\top\mathbf{X}\mathbf{w}$$

5.Get the formula of the closed-form solution.

$$\mathbf{w}^* = (\mathbf{X}^\mathrm{T}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\mathrm{T}\mathbf{y}$$

6. Get the value of parameter W by the closed-form solution, and update the parameter W.

$$\mathbf{w} = \mathbf{w} - \eta\frac{\partial L_{\mathbf{x}^{(i)}, y^{(i)}}(\mathbf{w})}{\partial \mathbf{w}}$$

7.Get the Loss， loss_train under the training set and loss_val by validating under validation set.
8.Output the value of Loss,loss_train and loss_val .

Linear Regression and Stochastic Gradient Descent
1.Load the experiment data. I used load_svmlight_file function in sklearn library.
2.Devide dataset. I divided dataset into training set and validation set using train_test_split function.
3.Initialize linear model parameters. I choose to set all parameter into zero.
4.Choose loss function and derivation:
Loss function:

$$L(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^\top(\mathbf{y} - \mathbf{X}\mathbf{w})$$

Derivation:

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} = -\mathbf{X}^\top\mathbf{y} + \mathbf{X}^\top\mathbf{X}\mathbf{w}$$

5.Calculate G toward loss function from each sample.
6.Denote the opposite direction of gradient G as D .
7.Update model: Wt-1=Wt+ηD. η is learning rate, a hyper-parameter that we can adjust.
8.Get the loss_train under the training set and loss_val by validating under validation set.
9.Repeate step 5 to 8 for several times, andand output the value of as well as .

Experient2—Logistic Regression and Support Vector Machine
Experiment Step
Logistic Regression and Batch Stochastic Gradient Descent

1.Load the training set and validation set.
2.Initialize logistic regression model parameter. I choose to set all parameter into zero.
3.Select the loss function and calculate its derivation.
Loss function:

$$\hat{y}^{(i)} = f(\mathbf{x}^{(i)}) = sigmoid(\mathbf{w}^{\mathrm{T}}\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}}}$$

$$L_{\mathbf{X},\mathbf{y}}(\mathbf{w}) = -\frac{1}{n}\sum_{i=1}^{n}[y^{(i)}\log\hat{y}^{(i)} + (1-y^{(i)})\log(1-\hat{y}^{(i)})]$$

Derivation:

$$\begin{aligned}\frac{\partial L_{\mathbf{X},\mathbf{y}}(\mathbf{w})}{\partial \mathbf{w}} &= -\frac{1}{n}\sum_{i=1}^{n}\frac{\partial}{\partial \mathbf{w}}[y^{(i)}\log f(\mathbf{x}^{(i)}) + (1-y^{(i)})\log(1-f(\mathbf{x}^{(i)}))] \\ &= -\frac{1}{n}\sum_{i=1}^{n}(y^{(i)}\cdot\frac{1}{f(\mathbf{x}^{(i)})}\cdot\frac{\partial f(\mathbf{x}^{(i)})}{\partial \mathbf{w}} - (1-y^{(i)})\cdot\frac{1}{1-f(\mathbf{x}^{(i)})}\cdot\frac{\partial f(\mathbf{x}^{(i)})}{\partial \mathbf{w}}) \\ &= -\frac{1}{n}\sum_{i=1}^{n}(\frac{\mathbf{x}^{(i)}y^{(i)}}{f(\mathbf{x}^{(i)})} - \frac{\mathbf{x}^{(i)}(1-y^{(i)})}{1-f(\mathbf{x}^{(i)})})\cdot f(\mathbf{x}^{(i)})\cdot(1-f(\mathbf{x}^{(i)})) \\ &= -\frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - f(\mathbf{x}^{(i)}))\mathbf{x}^{(i)} = \frac{1}{n}\sum_{i=1}^{n}(f(\mathbf{x}^{(i)}) - y^{(i)})\mathbf{x}^{(i)}\end{aligned}$$

4.Determine the size of the batch_size and randomly take some samples,calculate gradient G toward loss function from partial samples.
5.Use the SGD optimization method to update the parametric model and encourage additional attempts to optimize the Adam method.
6.Select the appropriate threshold, mark the sample whose predict scores greater than the threshold as positive, on the contrary as negative. Predict under validation set and get the loss .
7.Repeat step 4 to 6 for several times, and drawing graph of with the number of iterations.

Logistic Regression and Batch Stochastic Gradient Descent
1.Load the training set and validation set.
2.Initialize logistic regression model parameter,I choosed to initialize parameter randomly.
3.Select the loss function and calculate its derivation.
Loss function:

$$Hinge\ loss = \xi_i = max(0, 1 - y_i(\mathbf{w}^{\top}\mathbf{x}_i + b))$$

Derivation:

$$\frac{\partial(\sum_{i=1}^{N} max(0, 1 - y_i(\mathbf{w}^{\top}\mathbf{x}_i + b)))}{\partial \mathbf{w}} = \begin{cases} -\mathbf{y}^{\top}\mathbf{X} & 1 - y_i(\mathbf{w}^{\top}\mathbf{x}_i + b) >= 0 \\ 0 & 1 - y_i(\mathbf{w}^{\top}\mathbf{x}_i + b) < 0 \end{cases}$$

4.Determine the size of the batch_size and randomly take some samples,calculate gradient G toward loss function from partial samples.
5.Use the SGD optimization method to update the parametric model and encourage additional attempts to optimize the Adam method.
6.Select the appropriate threshold, mark the sample whose predict scores greater than the threshold as positive, on the contrary as negative. Predict under validation set and get the loss .
7.Repeat step 4 to 6 for several times, and drawing graph of with the number of iterations.

III. EXPERIMENT

Experient1—Linear Regression and Stochastic Gradient Descent

A.Dataset

Linear Regression uses Housing in LIBSVM Data, including 506 samples and each sample has 13 features,the property value is [-1,1].In this experiment, I used load_svmlight_file function in sklearn library to load the experient data, and train_test_split function was used to randomly shard 67% of the data set into training set and 33% into verification set.

B. Implementation

Code
RegressionExperiment.ipynb

```
import numpy as np
import matplotlib.pyplot as plt
import random
from sklearn.datasets import load_svmlight_file
from sklearn.model_selection import train_test_split

# 定义损失函数
def function_loss(w, X, y):
    loss = 0.5 * (y.T.dot(y) - 2 * w.T.dot(X.T).dot(y) +
w.T.dot(X.T).dot(X).dot(w))
    return loss

# 定义闭式解
def close_form_solution(w, X_train, y_train, X_val, y_val):
    new_w =
(X_train.T.dot(X_train)).I.dot(X_train.T).dot(y_train)
    new_w = new_w.reshape(w.shape[0], 1)  # 更新参数 w
    loss_train = function_loss(new_w, X_train, y_train)
    loss_val = function_loss(new_w, X_val, y_val)
    print(loss_train)  # 输出训练集的损失值
    print(loss_val)  # 输出测试集的损失值

# 随机梯度下降
def SGD(w, X, y):
    dw = np.dot(X.T, (X.dot(w) - y))
    return dw

# 线性回归
def linear_regression(w, X_train, y_train, X_val, y_val):
    iterations = 100  # 迭代次数
    alpha = 0.01  # 学习率
    loss_train_his = []
    loss_val_his = []
    for i in range(iterations):
```

```
        index = random.randint(0, X_train.shape[0] -1)
        X0 = X_train[index]
        y0 = y_train[index]
        loss_train = function_loss(w, X_train, y_train)
        loss_val = function_loss(w, X_val, y_val)
        loss_train_his.append(loss_train[0, 0])
        loss_val_his.append(loss_val[0, 0])
        w = w - alpha * SGD(w, X0, y0)

    plot(iterations, loss_train_his, loss_val_his)

# 画出损失函数的图像
def plot(iterations,loss_train, loss_val):
    plt.plot(np.arange(iterations), loss_train, label='Train_loss')

    plt.plot(np.arange(iterations),loss_val,label='Validation_los
s')
    plt.xlabel('Iteration')
    plt.ylabel('Loss')
    plt.title('The loss chart of Linear Regression')
    plt.legend(loc='best')
    plt.show()

if __name__ == '__main__':
    # 读取实验数据
    X, y = load_svmlight_file("G:\datasets\housing_scale.txt")
    X = X.todense()

    # 添加 w0 对应的 x0 方便矩阵化计算
    X = np.column_stack((X, np.ones(X.shape[0])))
    y = y.reshape((y.size, 1))

    # 数据集划分训练集和验证集
    X_train, X_val, y_train, y_val = train_test_split(X, y,
test_size=.33, random_state=42)

    # 线性模型参数全 0 初始化
    w = np.zeros(X.shape[1])
    w = w.reshape(w.shape[0], 1)

    loss = function_loss(w, X, y)

    #输出损失函数
    print(loss)

    close_form_solution(w, X_train, y_train, X_val, y_val)

    linear_regression(w, X_train, y_train, X_val, y_val)
```

Loss function and Derivation

Loss function:

$$L(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^\top(\mathbf{y} - \mathbf{X}\mathbf{w})$$

Derivation:

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} = -\mathbf{X}^\top\mathbf{y} + \mathbf{X}^\top\mathbf{X}\mathbf{w}$$

Initialization method of model parameters

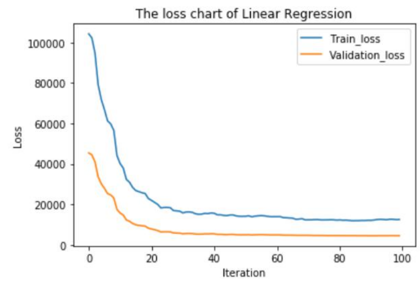
```
 # 线性模型参数全 0 初始化
  w = np.zeros(X.shape[1])
```

Parameters

iterations = 100 # 迭代次数
alpha = 0.01 # 学习率
 # 数据集划分训练集和验证集
val_size = 0.33,train_size = 0.67

Result

```
[[149813.17]]
[[3895.96022548]]
[[1730.45598442]]
```



Experient2—Logistic Regression and Support Vector Machine

A.Dataset

Experiment uses a9a of LIBSVM Data, including 32561/16281(testing) samples and each sample has 123/123 (testing) features. the property value is [-1,1].In this experiment, I used load_svmlight_file function in sklearn library to load the experient data, and train_test_split function was used to randomly shard 67% of the data set into training set and 33% into verification set.

B.Implementation

Code

ClassificationExperiment.ipynb

```
import numpy as np
import pandas as pd
import math
import random
from sklearn.datasets import load_svmlight_file
import matplotlib.pyplot as plt

def get_batch(X, y, batch_size):
    """
    将输入的 X 和 y 切割出 batch_size 大小
    :param X:特征
    :param y:标签
    :param batch_size:小组大小
    :return:切割结果
    """
    tn, tm = X.shape
    array_index = np.arange(0, tn, 1)
    np.random.shuffle(array_index)
    array_index_batch = array_index[:batch_size]
    X_batch = X[array_index_batch, :]
    y_batch = y[array_index_batch, :]
```

```
    return X_batch, y_batch

def sigmoid(x, w):
    """
    逻辑回归梯度激活函数
    :param x:特征
    :param w:参数
    :return:g(x)
    """
    wx = x.dot(w)
    return 1 / (1 + np.exp(wx * -1.0))

def logistic_regression_batch_gradient(X_train, y_train, X_val,
y_val):
    """
    逻辑回归+小批量梯度下降
    :param X_train:训练集特征
    :param y_train:训练集标签
    :param X_val:验证集特征
    :param y_val:验证集标签
    :return:
    """

    # 转换数据格式
    X_train = X_train.todense()
    sample_train, label_train = X_train.shape
    y_train[y_train == -1] = 0
    X_val = X_val.todense()
    sample_val, label_val = X_val.shape
    X_val = np.column_stack((X_val, np.zeros(sample_val)))
    y_val[y_val == -1] = 0

    # 参数 w 全零初始化
    w = np.zeros(shape=(label_train, 1))

    # batch 大小
    batch_size = 2 ** 4

    # 学习率
    alpha = 0.001

    # 迭代次数
    iterations = 200

    # 存储损失函数集
    loss_batch_history = []
    loss_val_history = []

    # 迭代
    for i in range(iterations):
        # 获取小批量数据集
        X_train_batch, y_train_batch = get_batch(X_train, y_train,
batch_size)

        # 计算训练集的损失函数
        gz_train = sigmoid(X_train_batch, w)
        matrix_loss=np.array(y_train_batch)*
np.array(np.log(gz_train)) + np.array(1 - y_train_batch) *
np.array(np.log(1 - gz_train))
        loss_batch = np.sum(matrix_loss) / batch_size * -1.0
        loss_batch_history.append(loss_batch)

        # 计算验证集的损失函数
        gz_val = sigmoid(X_val, w)
        matrix_loss_val=np.array(y_val)*np.array(np.log(gz_val))
                    + np.array(1 - y_val) * np.array(np.log(1
                    - gz_val))
        loss_val = np.sum(matrix_loss_val) / sample_val * -1.0
        loss_val_history.append(loss_val)

        # 梯度下降更新参数 w

        w=w-alpha/batch_size*X_train_batch.T.dot((sigmoid(
        X_train_batch, w)) - y_train_batch)

    # 作图
    plot(loss_batch_history, loss_val_history)

def  plot(loss_train_history, loss_val_history):
    """
    画图函数
    :param loss_train_history:一个 batch 小组的损失集
    :param loss_val_history:验证集的损失
    :return: 无
    """
   plt.plot(list(range(len(loss_train_history))),loss_train_history,
    label='Train_loss')
    plt.plot(list(range(len(loss_val_history))),   loss_val_history,
    label='Validation_loss')
    plt.xlabel('Iteration')
    plt.ylabel('Loss')
    plt.title('The loss chart of Classification Experient')
    plt.legend(loc='best')
    plt.show()

def sigmoid_svm(x, w, b):
    """
    SVM 核函数，此为线性不可分
    :param x:特征
    :param w:特征对应参数
    :param b:为 w0
    :return:g(x)
    """
    wx = x.dot(w)
    return wx + b

def svm_batch_gradient(X_train, y_train, X_val, y_val):
    """
    SVM + 小批量梯度下降
    :param X_train:训练集特征
    :param y_train:训练集标签
    :param X_val:验证集特征
    :return:
    """
```

```
# 转换数据格式
X_train_svm = X_train.todense()
sample_train, label_train = X_train.shape
y_train[y_train == -1] = 0

X_val_svm = X_val.todense()
sample_val, label_val = X_val.shape
X_val_svm=np.column_stack((X_val_svm,
  np.zeros(sample_val)))
y_val[y_val == -1] = 0

# 参数随机初始化
# w = np.zeros(shape=(label_train, 1))

w_svm = np.random.rand(label_train, 1)
b = random.random()

# 学习率
alpha = 0.001

# batch 大小
batch_size = 2 ** 2

# 惩罚参数
C = 1

# 迭代次数
iterations = 500

# 损失函数值的集
loss_batch_train_history = []
loss_batch_val_history = []

# 迭代
for i in range(iterations):
  # 获取小批量数据

  X_train_svm_batch,y_train_svm_bacth=get_batch(X_trai
  n_svm, y_train, batch_size)
  # 核函数
  kernel = sigmoid_svm(X_train_svm_batch, w_svm, b)
  # 计算训练集损失函数

  tsigmod=1-np.array(kernel)*np.array(y_train_svm_bacth
  )
  ttsigmod = tsigmod.copy()
  ttsigmod[ttsigmod < 0] = 0
  loss_train=w_svm.T.dot(w_svm)/2+C* np.sum(ttsigmod)
  loss_batch_train_history.append(loss_train[0, 0])
  # 核函数
  kernel_val = sigmoid_svm(X_val_svm, w_svm, b)
  # 计算验证集损失函数
  tsigmod_val = 1 - np.array(kernel_val) * np.array(y_val)
  ttsigmod_val = tsigmod_val.copy()
  ttsigmod_val[ttsigmod_val < 0] = 0

  loss_val=w_svm.T.dot(w_svm)/2+C*np.sum(ttsigmod_v
  al)
    loss_batch_val_history.append(loss_val[0, 0])

    # 更新参数 w and b
    gwx = np.mat(np.zeros(X_train_svm_batch.shape))
    gbx = np.random.random(batch_size)

    for i in range(batch_size):
      if (tsigmod[i] >= 0):

          gwx[i]=-np.array(X_train_svm_batch[i])*y_train_
          svm_bacth[i]
        gbx[i] = - y_train_svm_bacth[i]
      else:
        gwx[i] = np.zeros((1, X_train_svm_batch.shape[1]))
        gbx[i] = 0
    w_svm = w_svm * (1 - alpha) - alpha * C * np.sum(gwx,
axis=0).T
    b = b - alpha * C * np.sum(gbx)

#    print(loss_batch_train_history)
#    print(loss_batch_val_history)
  svm_plot(loss_batch_train_history,loss_batch_val_history)

def svm_plot(loss_train_history,loss_val_history):
    ax1 = plt.subplot(1, 2, 1)
    plt.plot(range(len(loss_train_history)),loss_train_history,
c='g')
    ax1.set_xlabel('iters')
    ax1.set_ylabel('loss train history')
    ax2 = plt.subplot(1, 2, 2)
    plt.plot(range(len(loss_val_history)),loss_val_history,
c='r')
    ax2.set_xlabel('iters')
    ax2.set_ylabel('loss val history')
    plt.show()

if __name__ == '__main__':
  # 读取数据
  data_train = load_svmlight_file("a9a.txt")
  data_val = load_svmlight_file("a9a.t")

  X_train = data_train[0]
  y_train = data_train[1]

  X_val = data_val[0]
  y_val = data_val[1]

  y_train = y_train.reshape((y_train.size, 1))
  y_val = y_val.reshape((y_val.size, 1))

  logistic_regression_batch_gradient(X_train, y_train, X_val,
y_val)

  svm_batch_gradient(X_train, y_train, X_val, y_val)
```

Loss function and Derivation

Loss function:

$$\hat{y}^{(i)} = f(\mathbf{x}^{(i)}) = sigmoid(\mathbf{w}^{\mathrm{T}}\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}}}$$

$$L_{\mathbf{x},\mathbf{y}}(\mathbf{w}) = -\frac{1}{n}\sum_{i=1}^{n}[y^{(i)}\log\hat{y}^{(i)} + (1-y^{(i)})\log(1-\hat{y}^{(i)})]$$

Derivation:

$$\begin{aligned}\frac{\partial L_{\mathbf{x},\mathbf{y}}(\mathbf{w})}{\partial \mathbf{w}} &= -\frac{1}{n}\sum_{i=1}^{n}\frac{\partial}{\partial \mathbf{w}}[y^{(i)}\log f(\mathbf{x}^{(i)}) + (1-y^{(i)})\log(1-f(\mathbf{x}^{(i)}))] \\ &= -\frac{1}{n}\sum_{i=1}^{n}(y^{(i)}\cdot\frac{1}{f(\mathbf{x}^{(i)})}\cdot\frac{\partial f(\mathbf{x}^{(i)})}{\partial \mathbf{w}} - (1-y^{(i)})\cdot\frac{1}{1-f(\mathbf{x}^{(i)})}\cdot\frac{\partial f(\mathbf{x}^{(i)})}{\partial \mathbf{w}}) \\ &= -\frac{1}{n}\sum_{i=1}^{n}(\frac{\mathbf{x}^{(i)}y^{(i)}}{f(\mathbf{x}^{(i)})} - \frac{\mathbf{x}^{(i)}(1-y^{(i)})}{1-f(\mathbf{x}^{(i)})})\cdot f(\mathbf{x}^{(i)})\cdot(1-f(\mathbf{x}^{(i)})) \\ &= -\frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - f(\mathbf{x}^{(i)}))\mathbf{x}^{(i)} = \frac{1}{n}\sum_{i=1}^{n}(f(\mathbf{x}^{(i)}) - y^{(i)})\mathbf{x}^{(i)}\end{aligned}$$

Loss function(SVM):

$$Hinge\ loss = \xi_i = max(0, 1 - y_i(\mathbf{w}^\top\mathbf{x}_i + b))$$

Derivation:

$$\frac{\partial(\sum_{i=1}^{N} max(0, 1 - y_i(\mathbf{w}^\top\mathbf{x}_i + b)))}{\partial \mathbf{w}} = \begin{cases} -\mathbf{y}^\top\mathbf{X} & 1 - y_i(\mathbf{w}^\top\mathbf{x}_i + b) >= 0 \\ 0 & 1 - y_i(\mathbf{w}^\top\mathbf{x}_i + b) < 0 \end{cases}$$

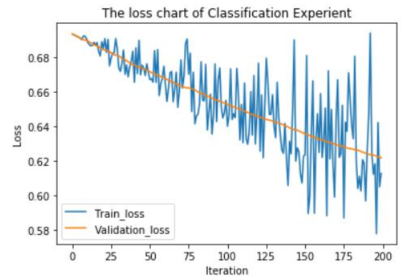
Initialization method of model parameters

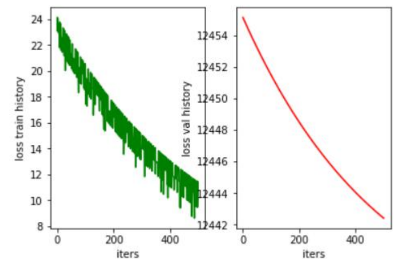
```
# 二分类参数 w 全零初始化
w = np.zeros(shape=(label_train, 1))

# SVM 参数 w 随机初始化，参数 b 随机初始化
w_svm = np.random.rand(label_train, 1)
b = random.random()
```

Parameters

```
# 学习率
alpha = 0.001
# batch 大小
batch_size = 2 ** 2
# 惩罚参数
C = 1
# 迭代次数
iterations = 500
```

Result





IV. CONCLUSION

This experiment enabled me to further understand the principles of linear regression and logistic regression, especially the amazing performance of SVM in classification.At the same time, I have mastered the principle and method of gradient descending parameter modulation more skillfully.By applying the linear regression algorithm to the Housing dataset in LIBSVM Data and the classification algorithm to the Australian dataset in LIBSVM Data, I realized the process of optimization and parameter tuning.

However, in this experiment, I also encountered many difficulties.One of the most prominent is the ability to tune and tune parameters and Debug code.The optimization model generally adjusts the learning rate, iteration frequency and data division proportion of the model, and these three parameters all influence each other and the adjusted step size is uncertain.In this experiment, I used fixed data to divide proportion and iteration times, and then adjusted the learning rate.Because the learning rate affects the convergence time, I assumed a large learning rate at the beginning, and then adjusted it continuously. However, the proportion of data division has little influence, so I just adjusted several values for comparison.In addition, some problems occurred when running code, I usually Debug for a long time to solve, so my engineering ability needs to be further strengthened.However, through these experiments, I also feel the charm of algorithms in machine learning and have a better understanding of machine learning.