

**计算机与信息 学院实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验课程： | Python机器学习实验 | | | | |
| 实验编号： | 实验四 | | | | |
| 实验名称： | **SVM在线性分类中的应用** | | | | |
| 实验人员： | 学号 | 18111207248 | | | |
| 姓名 | 吴钰 | | | |
| 班级 | 创新班 | | | |
| 实验日期： | 2021.6.5 | | | | |
| 实验室： | 2060301 | | | | |
|  |  | | | | |
| 实验评价： |  | | | | |
| 实验成绩： | |  | 评价日期： |  |
|  | 指导教师： | |  | | |

# 实验目的

掌握机器学习的几种模型

掌握SVM在线性分类中的应用

# 实验要求

编程实现SVM在线性分类中的应用。提交实验报告、训练集、测试集，均为附件提交开题报告、训练集和测试集

# 实验内容

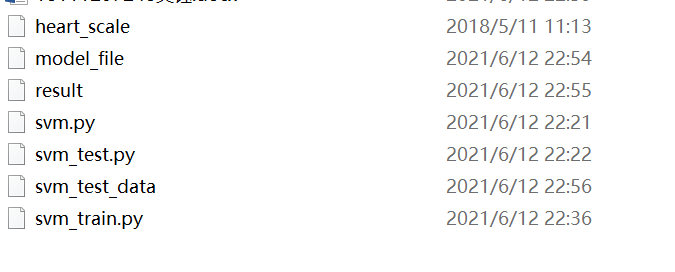
提交实验报告、训练集、测试集，均为附件提交开题报告、训练集和测试集

# 实验过程及结果

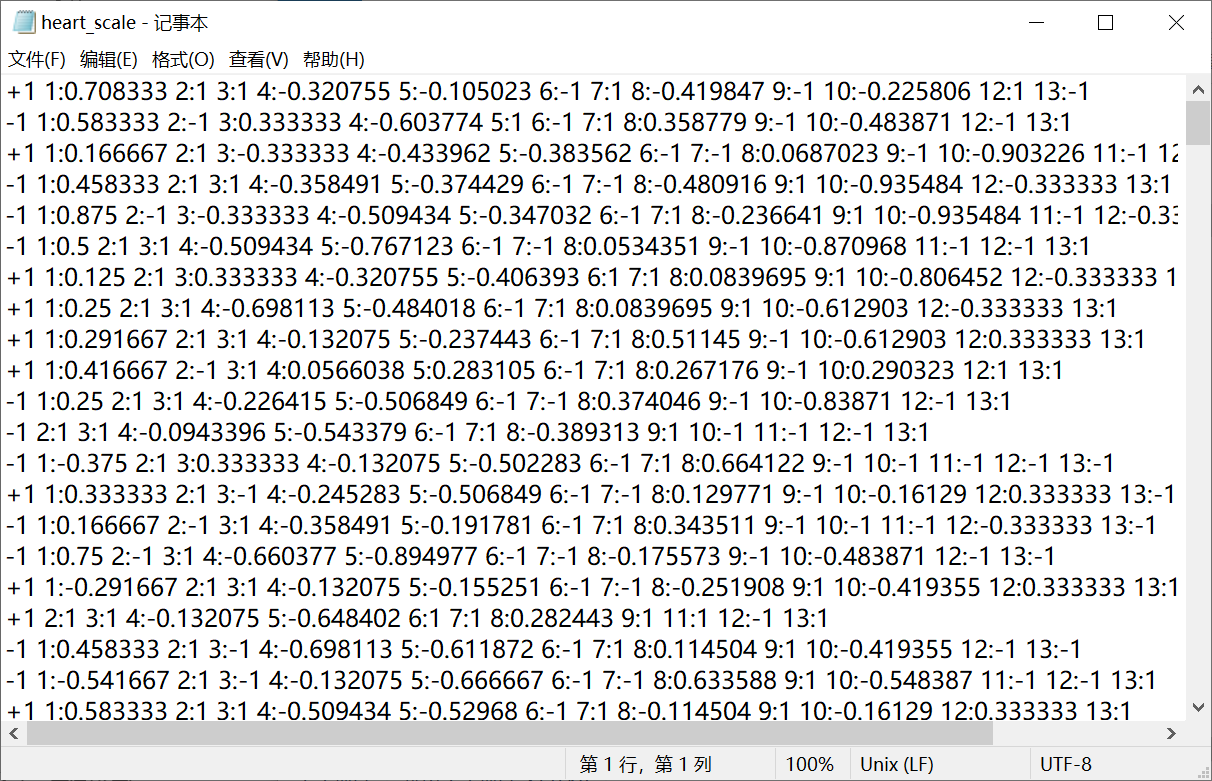
数据集截图:

heart\_scale是一个心脏有关的数据集，model\_file存放训练模型，result是学习结果也就是预测的类别的集合，svm\_test\_data是测试数据集。

svm是分类算法，svm\_train是训练器，svm\_test是测试器。



**heart\_scale**



实验代码：

**svm\_train.py**

import numpy as np

import svm

def load\_data\_libsvm(data\_file):

'''导入训练数据

input: data\_file(string):训练数据所在文件

output: data(mat):训练样本的特征

label(mat):训练样本的标签

'''

data = []

label = []

f = open(data\_file)

for line in f.readlines():

lines = line.strip().split(' ')

# 提取得出label

label.append(float(lines[0]))

# 提取出特征，并将其放入到矩阵中

index = 0

tmp = []

for i in range(1, len(lines)):

li = lines[i].strip().split(":")

if int(li[0]) - 1 == index:

tmp.append(float(li[1]))

else:

while(int(li[0]) - 1 > index):

tmp.append(0)

index += 1

tmp.append(float(li[1]))

index += 1

while len(tmp) < 13:

tmp.append(0)

data.append(tmp)

f.close()

return np.mat(data), np.mat(label).T

if \_\_name\_\_ == "\_\_main\_\_":

# 1、导入训练数据

print("------------ 1、load data --------------")

dataSet, labels = load\_data\_libsvm("heart\_scale")

# 2、训练SVM模型

print("------------ 2、training ---------------")

C = 0.6

toler = 0.001

maxIter = 500

svm\_model = svm.SVM\_training(dataSet, labels, C, toler, maxIter)

# 3、计算训练的准确性

print("------------ 3、cal accuracy --------------")

accuracy = svm.cal\_accuracy(svm\_model, dataSet, labels)

print("The training accuracy is: %.3f%%" % (accuracy \* 100))

# 4、保存最终的SVM模型

print("------------ 4、save model ----------------")

svm.save\_svm\_model(svm\_model, "model\_file")

**svm\_test.py**

# coding:UTF-8

import numpy as np

import pickle

from svm import svm\_predict

def load\_test\_data(test\_file):

'''导入测试数据

input: test\_file(string):测试数据

output: data(mat):测试样本的特征

'''

data = []

f = open(test\_file)

for line in f.readlines():

lines = line.strip().split(' ')

# 处理测试样本中的特征

index = 0

tmp = []

for i in range(0, len(lines)):

li = lines[i].strip().split(":")

if int(li[0]) - 1 == index:

tmp.append(float(li[1]))

else:

while(int(li[0]) - 1 > index):

tmp.append(0)

index += 1

tmp.append(float(li[1]))

index += 1

while len(tmp) < 13:

tmp.append(0)

data.append(tmp)

f.close()

return np.mat(data)

def load\_svm\_model(svm\_model\_file):

'''导入SVM模型

input: svm\_model\_file(string):SVM模型保存的文件

output: svm\_model:SVM模型

'''

with open(svm\_model\_file, 'rb') as f:

svm\_model = pickle.load(f)

return svm\_model

def get\_prediction(test\_data, svm):

'''对样本进行预测

input: test\_data(mat):测试数据

svm:SVM模型

output: prediction(list):预测所属的类别

'''

m = np.shape(test\_data)[0]

prediction = []

for i in range(m):

# 对每一个样本得到预测值

predict = svm\_predict(svm, test\_data[i, :])

# 得到最终的预测类别

prediction.append(str(np.sign(predict)[0, 0]))

return prediction

def save\_prediction(result\_file, prediction):

'''保存预测的结果

input: result\_file(string):结果保存的文件

prediction(list):预测的结果

'''

f = open(result\_file, 'w')

f.write(" ".join(prediction))

f.close()

if \_\_name\_\_ == "\_\_main\_\_":

# 1、导入测试数据

print( "--------- 1.load data ---------")

test\_data = load\_test\_data("svm\_test\_data")

# 2、导入SVM模型

print("--------- 2.load model ----------")

svm\_model = load\_svm\_model("model\_file")

# 3、得到预测值

print("--------- 3.get prediction ---------")

prediction = get\_prediction(test\_data, svm\_model)

# 4、保存最终的预测值

print("--------- 4.save result ----------")

save\_prediction("result", prediction)

**svm.py**

#coding:UTF-8

import numpy as np

import pickle

class SVM:

def \_\_init\_\_(self, dataSet, labels, C, toler, kernel\_option):

self.train\_x = dataSet # 训练特征

self.train\_y = labels # 训练标签

self.C = C # 惩罚参数

self.toler = toler # 迭代的终止条件之一

self.n\_samples = np.shape(dataSet)[0] # 训练样本的个数

self.alphas = np.mat(np.zeros((self.n\_samples, 1))) # 拉格朗日乘子

self.b = 0

self.error\_tmp = np.mat(np.zeros((self.n\_samples, 2))) # 保存E的缓存

self.kernel\_opt = kernel\_option # 选用的核函数及其参数

self.kernel\_mat = calc\_kernel(self.train\_x, self.kernel\_opt) # 核函数的输出

def cal\_kernel\_value(train\_x, train\_x\_i, kernel\_option):

'''样本之间的核函数的值

input: train\_x(mat):训练样本

train\_x\_i(mat):第i个训练样本

kernel\_option(tuple):核函数的类型以及参数

output: kernel\_value(mat):样本之间的核函数的值

'''

kernel\_type = kernel\_option[0] # 核函数的类型，分为rbf和其他

m = np.shape(train\_x)[0] # 样本的个数

kernel\_value = np.mat(np.zeros((m, 1)))

if kernel\_type == 'rbf': # rbf核函数

sigma = kernel\_option[1]

if sigma == 0:

sigma = 1.0

for i in range(m):

diff = train\_x[i, :] - train\_x\_i

kernel\_value[i] = np.exp(diff \* diff.T / (-2.0 \* sigma\*\*2))

else: # 不使用核函数

kernel\_value = train\_x \* train\_x\_i.T

return kernel\_value

def calc\_kernel(train\_x, kernel\_option):

'''计算核函数矩阵

input: train\_x(mat):训练样本的特征值

kernel\_option(tuple):核函数的类型以及参数

output: kernel\_matrix(mat):样本的核函数的值

'''

m = np.shape(train\_x)[0] # 样本的个数

kernel\_matrix = np.mat(np.zeros((m, m))) # 初始化样本之间的核函数值

for i in range(m):

kernel\_matrix[:, i] = cal\_kernel\_value(train\_x, train\_x[i, :], kernel\_option)

return kernel\_matrix

def cal\_error(svm, alpha\_k):

'''误差值的计算

input: svm:SVM模型

alpha\_k(int):选择出的变量

output: error\_k(float):误差值

'''

output\_k = float(np.multiply(svm.alphas, svm.train\_y).T \* svm.kernel\_mat[:, alpha\_k] + svm.b)

error\_k = output\_k - float(svm.train\_y[alpha\_k])

return error\_k

def update\_error\_tmp(svm, alpha\_k):

'''重新计算误差值

input: svm:SVM模型

alpha\_k(int):选择出的变量

output: 对应误差值

'''

error = cal\_error(svm, alpha\_k)

svm.error\_tmp[alpha\_k] = [1, error]

def select\_second\_sample\_j(svm, alpha\_i, error\_i):

'''选择第二个样本

input: svm:SVM模型

alpha\_i(int):选择出的第一个变量

error\_i(float):E\_i

output: alpha\_j(int):选择出的第二个变量

error\_j(float):E\_j

'''

# 标记为已被优化

svm.error\_tmp[alpha\_i] = [1, error\_i]

candidateAlphaList = np.nonzero(svm.error\_tmp[:, 0].A)[0]

maxStep = 0

alpha\_j = 0

error\_j = 0

if len(candidateAlphaList) > 1:

for alpha\_k in candidateAlphaList:

if alpha\_k == alpha\_i:

continue

error\_k = cal\_error(svm, alpha\_k)

if abs(error\_k - error\_i) > maxStep:

maxStep = abs(error\_k - error\_i)

alpha\_j = alpha\_k

error\_j = error\_k

else: # 随机选择

alpha\_j = alpha\_i

while alpha\_j == alpha\_i:

alpha\_j = int(np.random.uniform(0, svm.n\_samples))

error\_j = cal\_error(svm, alpha\_j)

return alpha\_j, error\_j

def choose\_and\_update(svm, alpha\_i):

'''判断和选择两个alpha进行更新

input: svm:SVM模型

alpha\_i(int):选择出的第一个变量

'''

error\_i = cal\_error(svm, alpha\_i) # 计算第一个样本的E\_i

# 判断选择出的第一个变量是否违反了KKT条件

if (svm.train\_y[alpha\_i] \* error\_i < -svm.toler) and (svm.alphas[alpha\_i] < svm.C) or\

(svm.train\_y[alpha\_i] \* error\_i > svm.toler) and (svm.alphas[alpha\_i] > 0):

# 1、选择第二个变量

alpha\_j, error\_j = select\_second\_sample\_j(svm, alpha\_i, error\_i)

alpha\_i\_old = svm.alphas[alpha\_i].copy()

alpha\_j\_old = svm.alphas[alpha\_j].copy()

# 2、计算上下界

if svm.train\_y[alpha\_i] != svm.train\_y[alpha\_j]:

L = max(0, svm.alphas[alpha\_j] - svm.alphas[alpha\_i])

H = min(svm.C, svm.C + svm.alphas[alpha\_j] - svm.alphas[alpha\_i])

else:

L = max(0, svm.alphas[alpha\_j] + svm.alphas[alpha\_i] - svm.C)

H = min(svm.C, svm.alphas[alpha\_j] + svm.alphas[alpha\_i])

if L == H:

return 0

# 3、计算eta

eta = 2.0 \* svm.kernel\_mat[alpha\_i, alpha\_j] - svm.kernel\_mat[alpha\_i, alpha\_i] \

- svm.kernel\_mat[alpha\_j, alpha\_j]

if eta >= 0:

return 0

# 4、更新alpha\_j

svm.alphas[alpha\_j] -= svm.train\_y[alpha\_j] \* (error\_i - error\_j) / eta

# 5、确定最终的alpha\_j

if svm.alphas[alpha\_j] > H:

svm.alphas[alpha\_j] = H

if svm.alphas[alpha\_j] < L:

svm.alphas[alpha\_j] = L

# 6、判断是否结束

if abs(alpha\_j\_old - svm.alphas[alpha\_j]) < 0.00001:

update\_error\_tmp(svm, alpha\_j)

return 0

# 7、更新alpha\_i

svm.alphas[alpha\_i] += svm.train\_y[alpha\_i] \* svm.train\_y[alpha\_j] \

\* (alpha\_j\_old - svm.alphas[alpha\_j])

# 8、更新b

b1 = svm.b - error\_i - svm.train\_y[alpha\_i] \* (svm.alphas[alpha\_i] - alpha\_i\_old) \

\* svm.kernel\_mat[alpha\_i, alpha\_i] \

- svm.train\_y[alpha\_j] \* (svm.alphas[alpha\_j] - alpha\_j\_old) \

\* svm.kernel\_mat[alpha\_i, alpha\_j]

b2 = svm.b - error\_j - svm.train\_y[alpha\_i] \* (svm.alphas[alpha\_i] - alpha\_i\_old) \

\* svm.kernel\_mat[alpha\_i, alpha\_j] \

- svm.train\_y[alpha\_j] \* (svm.alphas[alpha\_j] - alpha\_j\_old) \

\* svm.kernel\_mat[alpha\_j, alpha\_j]

if (0 < svm.alphas[alpha\_i]) and (svm.alphas[alpha\_i] < svm.C):

svm.b = b1

elif (0 < svm.alphas[alpha\_j]) and (svm.alphas[alpha\_j] < svm.C):

svm.b = b2

else:

svm.b = (b1 + b2) / 2.0

# 9、更新error

update\_error\_tmp(svm, alpha\_j)

update\_error\_tmp(svm, alpha\_i)

return 1

else:

return 0

def SVM\_training(train\_x, train\_y, C, toler, max\_iter, kernel\_option = ('rbf', 0.431029)):

'''SVM的训练

input: train\_x(mat):训练数据的特征

train\_y(mat):训练数据的标签

C(float):惩罚系数

toler(float):迭代的终止条件之一

max\_iter(int):最大迭代次数

kerner\_option(tuple):核函数的类型及其参数

output: svm模型

'''

# 1、初始化SVM分类器

svm = SVM(train\_x, train\_y, C, toler, kernel\_option)

# 2、开始训练

entireSet = True

alpha\_pairs\_changed = 0

iteration = 0

while (iteration < max\_iter) and ((alpha\_pairs\_changed > 0) or entireSet):

print("\t iterration: ", iteration)

alpha\_pairs\_changed = 0

if entireSet:

# 对所有的样本

for x in range(svm.n\_samples):

alpha\_pairs\_changed += choose\_and\_update(svm, x)

iteration += 1

else:

# 非边界样本

bound\_samples = []

for i in range(svm.n\_samples):

if svm.alphas[i,0] > 0 and svm.alphas[i,0] < svm.C:

bound\_samples.append(i)

for x in bound\_samples:

alpha\_pairs\_changed += choose\_and\_update(svm, x)

iteration += 1

# 在所有样本和非边界样本之间交替

if entireSet:

entireSet = False

elif alpha\_pairs\_changed == 0:

entireSet = True

return svm

def svm\_predict(svm, test\_sample\_x):

'''利用SVM模型对每一个样本进行预测

input: svm:SVM模型

test\_sample\_x(mat):样本

output: predict(float):对样本的预测

'''

# 1、计算核函数矩阵

kernel\_value = cal\_kernel\_value(svm.train\_x, test\_sample\_x, svm.kernel\_opt)

# 2、计算预测值

predict = kernel\_value.T \* np.multiply(svm.train\_y, svm.alphas) + svm.b

return predict

def cal\_accuracy(svm, test\_x, test\_y):

'''计算预测的准确性

input: svm:SVM模型

test\_x(mat):测试的特征

test\_y(mat):测试的标签

output: accuracy(float):预测的准确性

'''

n\_samples = np.shape(test\_x)[0] # 样本的个数

correct = 0.0

for i in range(n\_samples):

# 对每一个样本得到预测值

predict=svm\_predict(svm, test\_x[i, :])

# 判断每一个样本的预测值与真实值是否一致

if np.sign(predict) == np.sign(test\_y[i]):

correct += 1

accuracy = correct / n\_samples

return accuracy

def save\_svm\_model(svm\_model, model\_file):

'''保存SVM模型

input: svm\_model:SVM模型

model\_file(string):SVM模型需要保存到的文件

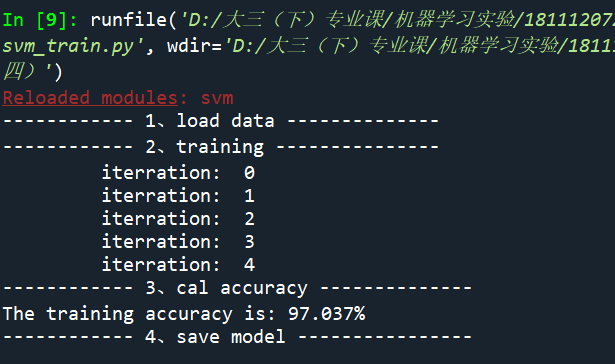
'''

with open(model\_file, 'wb') as f:

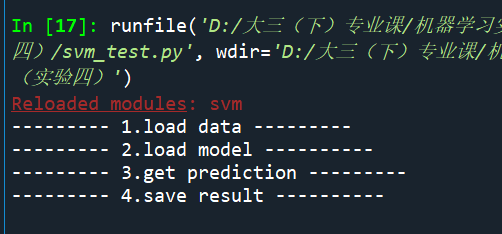
pickle.dump(svm\_model, f)

实验运行截图：

运行svm\_train.py



运行svm\_test.py



预测的类别的集合

