模式识别 作业二

运行结果

结论

SVM

使用语言

Python

引入库

```
from libsvm.svm import *
from libsvm.svmutil import *
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

任务1

参数

默认 即`-c 1 -t 2`

实现代码

```
#-----任务1-----
train_label_1,train_pixel_1 = svm_read_problem('svmguide1.txt')
predict_label_1,predict_pixel_1 =
svm_read_problem('svmguide1_test.txt')
m1 = svm_train(train_label_1, train_pixel_1)
print("#1 result:")
p_label_1, p_acc_1, p_val_1 = svm_predict(predict_label_1,
predict_pixel_1, m1);
print(p_acc_1)
```

运行结果

```
optimization finished, #iter = 5371
nu = 0.606150
obj = -1061.528918, rho = -0.495266
nSV = 3053, nBSV = 722
Total nSV = 3053
#1 result:
Accuracy = 66.925% (2677/4000) (classification)
(66.925, 0.33075, 0.20090878844718825)
```

任务2

预处理

使用`libsvm`库中的可执行文件`svm-scale`对数据进行了规范化

参数

默认 即`-c 1 -t 2`

实现代码

```
#-----任务2-----
train_label_2,train_pixel_2 = svm_read_problem('scaledata.txt')
predict_label_2,predict_pixel_2 = svm_read_problem('scaledata_test.txt')
m2 = svm_train(train_label_2, train_pixel_2)
print("#2 result:")
p_label_2, p_acc_2, p_val_2 = svm_predict(predict_label_2, predict_pixel_2, m2);
print(p_acc_2)
```

运行结果

```
optimization finished, #iter = 496
nu = 0.202599
obj = -507.307046, rho = 2.627039
nSV = 630, nBSV = 621
Total nSV = 630
#2 result:
Accuracy = 95.6% (3824/4000) (classification)
(95.6, 0.044, 0.8332137891240148)
```

任务3

参数

线性核`-t 0`

实现代码

```
#-----任务3-----
train_label_3,train_pixel_3 = svm_read_problem('svmguide1.txt')
predict_label_3,predict_pixel_3 =
svm_read_problem('svmguide1_test.txt')
m3 = svm_train(train_label_3, train_pixel_3, '-t 0')
print("#3 result:")
p_label_3, p_acc_3, p_val_3 = svm_predict(predict_label_3,
predict_pixel_3, m3);
print(p_acc_3)
```

运行结果

```
optimization finished, #iter = 3509115
nu = 0.121917
obj = -376.234540, rho = 5.887607
nSV = 381, nBSV = 375
Total nSV = 381
#3 result:
Accuracy = 95.675% (3827/4000) (classification)
(95.675, 0.04325, 0.8345425456989267)
```

任务4

参数

`-c 1000 -t 2`

实现代码

```
#-----任务4-----
train_label_4,train_pixel_4 = svm_read_problem('svmguide1.txt')
predict_label_4,predict_pixel_4 =
svm_read_problem('svmguide1_test.txt')
m4 = svm_train(train_label_4, train_pixel_4, '-c 1000 -t 2')
print("#4 result:")
p_label_4, p_acc_4, p_val_4 = svm_predict(predict_label_4,
predict_pixel_4, m4);
print(p_acc_4)
```

运行结果

```
optimization finished, #iter = 6383
nu = 0.000721
obj = -1114.038221, rho = -0.407723
nSV = 3001, nBSV = 0
Total nSV = 3001
#4 result:
Accuracy = 70.475% (2819/4000) (classification)
(70.475, 0.29525, 0.25160063391442156)
```

任务5

预处理

使用`libsvm`内置的`tools`工具库中的脚本`easy.py`确定了`RBF核`中的超参数`-c -g`，脚本运行结果如下：

```
(base) ➜  tools python easy.py /Users/plotnickslope/Desktop/学习资料/模式识别/作业/SVM/svmguide1.txt /Users/plotnickslope/Desktop/学习资料/模式识别/作业/SVM/svmguide1_test.txt
Scaling training data...
Cross validation...
Best c=8192. 0, g=0. 03125 CV rate=96. 9569
Training...
Output model: svmguide1.txt.model
Scaling testing data...
Testing...
Accuracy= 96. 525% (3861/4000) (classification)
Output prediction: svmguide1_test.txt.predict
```

易知脚本确定的参数为`-c 8192 -g 0.03125`且对数据进行了规范化处理

参数

`-c 8192 -g 0.03125 -t 2`

实现代码

```
#-----任务5-----
train_label_5,train_pixel_5 = svm_read_problem('scaledata.txt')
predict_label_5,predict_pixel_5 = svm_read_problem('scaledata_test.txt')
m5 = svm_train(train_label_5, train_pixel_5, '-c 8192 -g 0.03125 -t 2')
print("#5 result:")
p_label_5, p_acc_5, p_val_5 = svm_predict(predict_label_5, predict_pixel_5, m5);
print(p_acc_5)
```

运行结果

```
optimization finished, #iter = 65401
nu = 0.090885
obj = -2206868.495761, rho = 102.101563
nSV = 287, nBSV = 272
Total nSV = 287
#5 result:
Accuracy = 95.8% (3832/4000) (classification)
(95.8, 0.042, 0.8423994835502108)
```

任务6

通过这组实验，我学习到 SVM 模型的性能与准确率在很大程度上取决于输入的超参数与选用的核函数。在选取合适的超参数或核函数时，即使是同一份数据也可以在分类的准确率上获得大幅提升。同时，对数据进行合适的规范化或缩放操作也能大幅提高模型的识别准确率

附加任务

数据选择

根据检验，官网的`w1a`数据中共`2477`条数据，其中负类样本`2405`条，标签为`-1`，正类样本`72`条，标签为`+1`，符合不平衡数据集要求。

参数

默认参数

为验证`-wi`参数的作用 首先用默认参数`-w1 1`运行代码，由于已知数据不平衡，考虑计算其数量较少的正类样本的预测正确率，即真阳率，实现代码如下：

实现代码

```
train_label_6,train_pixel_6 = svm_read_problem('w1a.txt')
predict_label_6,predict_pixel_6 = svm_read_problem('w1a.t')
m = svm_train(train_label_6, train_pixel_6)
print("#6 result:")
p_label_6, p_acc_6, p_val_6 = svm_predict(predict_label_6,
predict_pixel_6, m)
print(p_acc_6)
all = 0
```

```
right = 0
for i in range(len(predict_label_6)):
    if predict_label_6[i] == 1:
        all = all + 1
        if p_label_6[i] == 1:
            right = right + 1
print('TPR:')
print(right / all)
```

运行结果

```
optimization finished, #iter = 360
nu = 0.058135
obj = -140.822687, rho = 0.597212
nSV = 203, nBSV = 114
Total nSV = 203
#6 result:
Accuracy = 97.0236% (45865/47272) (classification)
(97.02360805550855, 0.11905567777965814, nan)
TPR:
0.0
```

不难看出，模型整体准确率高，但真阳率极低，因此用`-wi`参数对模型进行调整

调整参数

为体现`-wi`参数的效果 对正类样本的权重依次赋值为`20, 25, 30...45, 50`，并依次计算模型的整体准确率与真阳率，实现代码如下：

实现代码

```
train_label_6,train_pixel_6 = svm_read_problem('w1a.txt')
predict_label_6,predict_pixel_6 = svm_read_problem('w1a.t')
m = svm_train(train_label_6, train_pixel_6)
print("#6 result:")
p_label_6, p_acc_6, p_val_6 = svm_predict(predict_label_6,
predict_pixel_6, m)
print(p_acc_6)
all = 0
right = 0
for i in range(len(predict_label_6)):
```

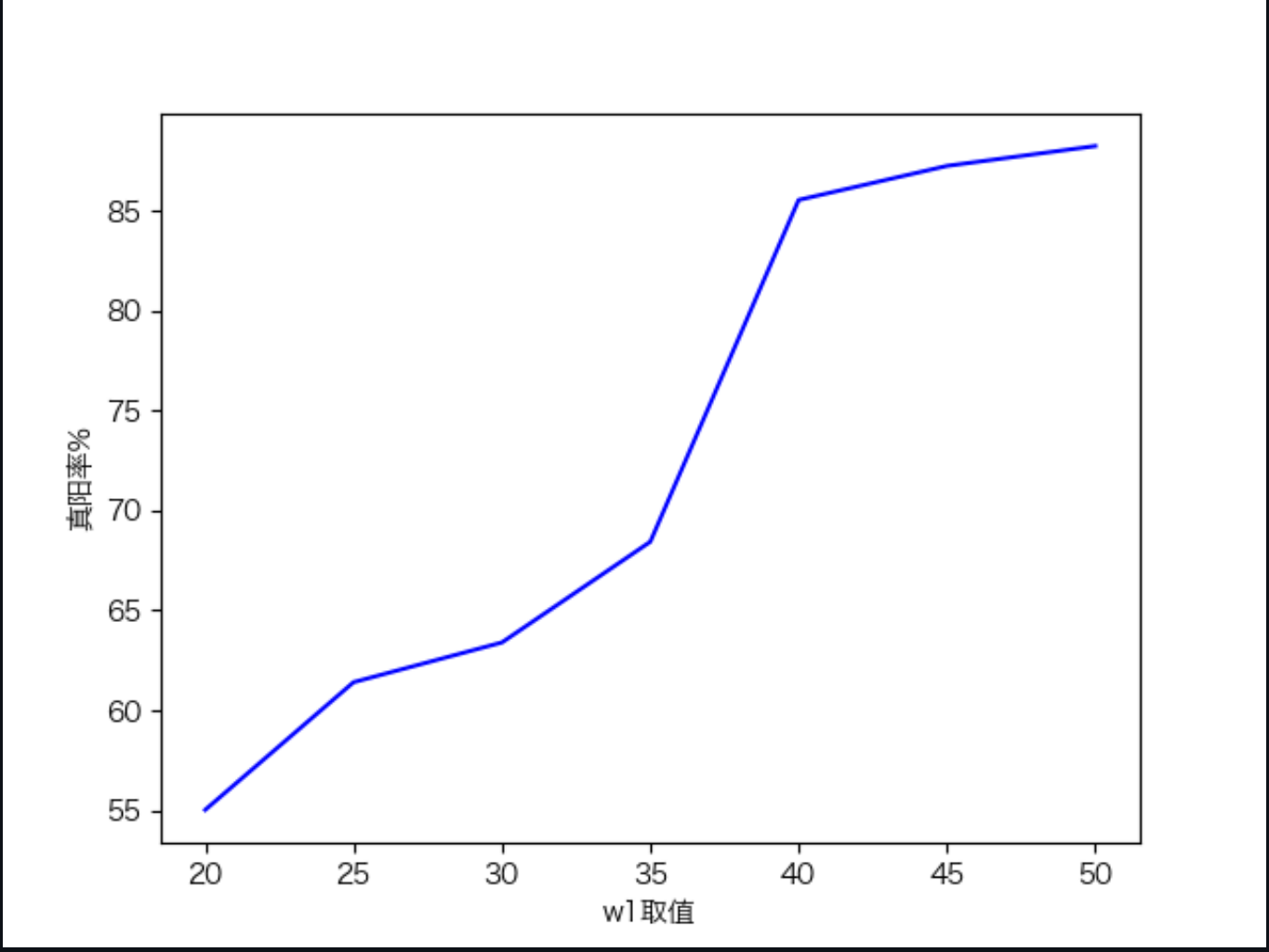
```
    if predict_label_6[i] == 1:
        all = all + 1
        if p_label_6[i] == 1:
            right = right + 1
print('TPR:')
print(right / all)

xdata = np.array([]) # w1取值
ydata = np.array([]) # 真阳率
acc = np.array([]) # 样本总数
for i in range(20, 51, 5):
    option = '-w1 ' + str(i)
    xdata = np.append(xdata, i)
    m6 = svm_train(train_label_6, train_pixel_6, option)
    p_label, p_acc, p_val = svm_predict(predict_label_6,
predict_pixel_6, m6);
    right = 0
    all = 0
    for i in range(len(predict_label_6)):
        if predict_label_6[i] == 1:
            all = all + 1
            if p_label[i] == 1:
                right = right + 1
    ydata = np.append(ydata, right / all * 100)
    acc = np.append(acc, p_acc[0])
data = pd.DataFrame([])
data['w1取值'] = xdata
data['真阳率%'] = ydata
data['准确率%'] = acc
data.to_excel('运行结果.xlsx')

plt.figure()
plt.plot(xdata, ydata, '-b')
plt.xlabel('w1取值')
plt.ylabel('真阳率%')
plt.show()
```

运行结果

结果保存至excel，展示如下：

	w1取值	真阳率%	准确率%
0	20	55.01066	96.97495
1	25	61.40725	96.63014
2	30	63.3973	96.35514
3	35	68.4435	94.41741
4	40	85.57214	82.27069
5	45	87.2779	76.54849
6	50	88.27292	74.49653

真阳率随`w1`取值变化的折线图如下：



结论

由此可见，在不平衡数据集中，通过调整`-wi`参数，人为增大较少数据量类别在模型计算中的权重，可以有效提高该类的预测准确率。