

模式识别 作业二

模式识别 作业二

SVM

使用语言

引入库

任务1

参数

实现代码

运行结果

任务2

预处理

参数

实现代码

运行结果

任务3

参数

实现代码

运行结果

任务4

参数

实现代码

运行结果

任务5

预处理

参数

实现代码

运行结果

任务6

附加任务

数据选择

参数

默认参数

实现代码

运行结果

调整参数

实现代码

运行结果

结论

KDE

使用语言

引入库

任务a

实现代码

任务b&任务c

预处理

实现代码

导出图像

结果分析

任务d

实现代码

运行结果

结果分析

SVM

使用语言

Python

引入库

```
1 from libsvm.svm import *
2 from libsvm.svmutil import *
3 import numpy as np
4 import pandas as pd
5 import matplotlib.pyplot as plt
```

任务1

参数

默认即 `-c 1 -t 2`

实现代码

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

运行结果

```
1 optimization finished, #iter = 5371
2 nu = 0.606150
3 obj = -1061.528918, rho = -0.495266
4 nSV = 3053, nBSV = 722
5 Total nSV = 3053
6 #1 result:
7 Accuracy = 66.925% (2677/4000) (classification)
8 (66.925, 0.33075, 0.2009087884471825)
```

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

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

运行结果

```
1 optimization finished, #iter = 496
2 nu = 0.202599
3 obj = -507.307046, rho = 2.627039
4 nSV = 630, nBSV = 621
5 Total nSV = 630
6 #2 result:
7 Accuracy = 95.6% (3824/4000) (classification)
8 (95.6, 0.044, 0.8332137891240148)
```

任务3

参数

线性核 `-t 0`

实现代码

```

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

```

运行结果

```

1 optimization finished, #iter = 3509115
2 nu = 0.121917
3 obj = -376.234540, rho = 5.887607
4 nSV = 381, nBSV = 375
5 Total nSV = 381
6 #3 result:
7 Accuracy = 95.675% (3827/4000) (classification)
8 (95.675, 0.04325, 0.8345425456989267)

```

任务4

参数

```
-c 1000 -t 2
```

实现代码

```

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

```

运行结果

```
1 optimization finished, #iter = 6383
2 nu = 0.000721
3 obj = -1114.038221, rho = -0.407723
4 nSV = 3001, nBSV = 0
5 Total nSV = 3001
6 #4 result:
7 Accuracy = 70.475% (2819/4000) (classification)
8 (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：

```
1 (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
2 Scaling training data...
3 Cross validation...
4 Best c=8192. 0, g=0. 03125 CV rate=96. 9569
5 Training...
6 Output model: svmguide1.txt.model
7 Scaling testing data...
8 Testing...
9 Accuracy= 96. 525% (3861/4000) (classification)
10 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

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

运行结果

```
1 optimization finished, #iter = 65401
2 nu = 0.090885
3 obj = -2206868.495761, rho = 102.101563
4 nSV = 287, nBSV = 272
5 Total nSV = 287
6 #5 result:
7 Accuracy = 95.8% (3832/4000) (classification)
8 (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

```
1 train_label_6,train_pixel_6 = svm_read_problem('w1a.txt')
2 predict_label_6,predict_pixel_6 = svm_read_problem('w1a.t')
3 m = svm_train(train_label_6, train_pixel_6)
4 print("#6 result:")
5 p_label_6, p_acc_6, p_val_6 = svm_predict(predict_label_6,
6 predict_pixel_6, m)
6 print(p_acc_6)
7 all = 0
8 right = 0
9 for i in range(len(predict_label_6)):
10     if predict_label_6[i] == 1:
11         all = all + 1
12         if p_label_6[i] == 1:
13             right = right + 1
14 print('TPR:')
15 print(right / all)
```

运行结果

```
1 optimization finished, #iter = 360
2 nu = 0.058135
3 obj = -140.822687, rho = 0.597212
4 nSV = 203, nBSV = 114
5 Total nSV = 203
6 #6 result:
7 Accuracy = 97.0236% (45865/47272) (classification)
8 (97.02360805550855, 0.11905567777965814, nan)
9 TPR:
10 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

```
1 train_label_6,train_pixel_6 = svm_read_problem('w1a.txt')
2 predict_label_6,predict_pixel_6 = svm_read_problem('w1a.t')
3 m = svm_train(train_label_6, train_pixel_6)
4 print("#6 result:")
5 p_label_6, p_acc_6, p_val_6 = svm_predict(predict_label_6,
6 predict_pixel_6, m)
6 print(p_acc_6)
7 all = 0
8 right = 0
9 for i in range(len(predict_label_6)):
10     if predict_label_6[i] == 1:
11         all = all + 1
12         if p_label_6[i] == 1:
13             right = right + 1
14 print('TPR:')
15 print(right / all)
16
17 xdata = np.array([]) # w1取值
18 ydata = np.array([]) # 真阳率
19 acc = np.array([]) # 样本总数
20 for i in range(20, 51, 5):
21     option = '-w1 ' + str(i)
22     xdata = np.append(xdata, i)
23     m6 = svm_train(train_label_6, train_pixel_6, option)
24     p_label, p_acc, p_val = svm_predict(predict_label_6,
25 predict_pixel_6, m6);
26     right = 0
27     all = 0
28     for i in range(len(predict_label_6)):
29         if predict_label_6[i] == 1:
30             all = all + 1
31             if p_label[i] == 1:
32                 right = right + 1
33     ydata = np.append(ydata, right / all * 100)
34     acc = np.append(acc, p_acc[0])
35 data = pd.DataFrame([])
36 data['w1取值'] = xdata
```

```

36 data['真阳率%'] = ydata
37 data['准确率%'] = acc
38 data.to_excel('运行结果.xlsx')
39
40 plt.figure()
41 plt.plot(xdata, ydata, '-b')
42 plt.xlabel('w1取值')
43 plt.ylabel('真阳率%')
44 plt.show()

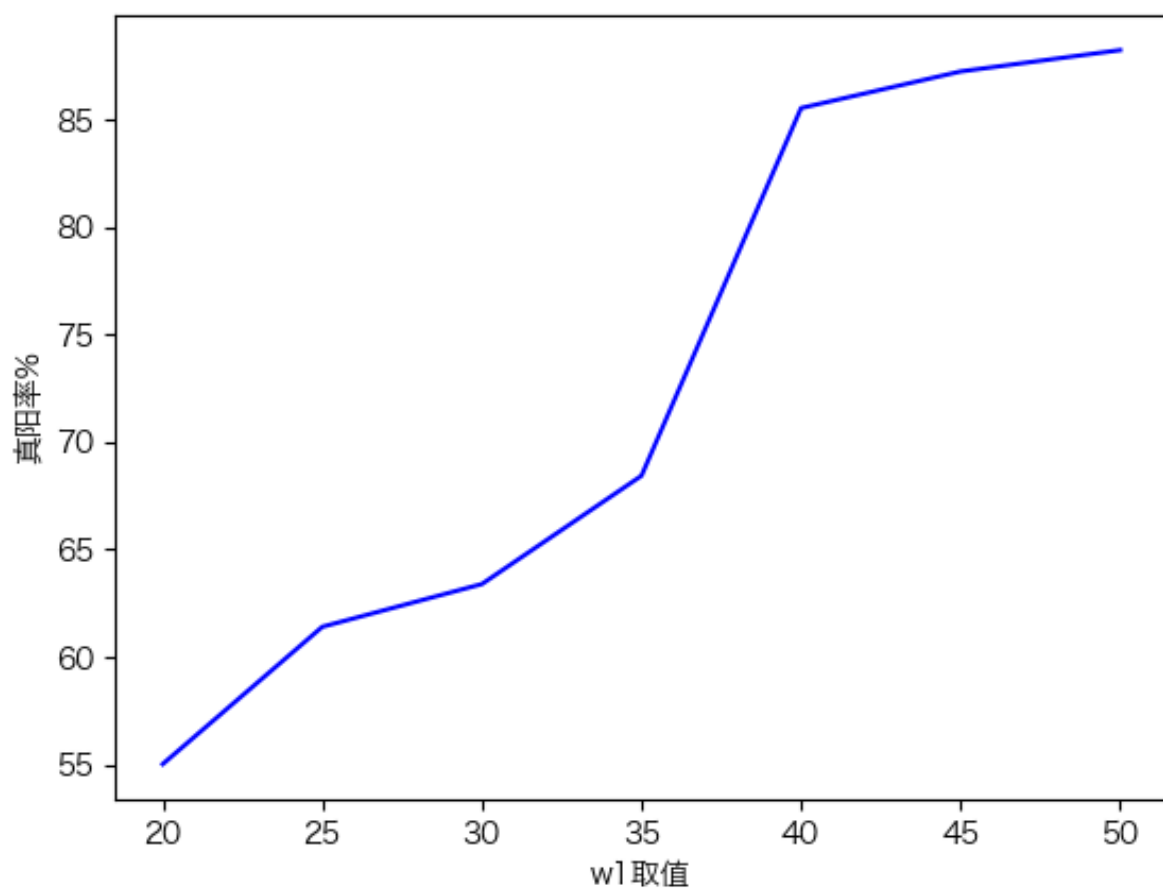
```

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。

KDE

使用语言

Python Matlab

引入库

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 from scipy import stats as st
5 from sklearn.neighbors import KernelDensity
6 from sklearn.model_selection import GridSearchCV
7 from sklearn.model_selection import LeaveOneOut
8 import math
```

任务a

实现代码

```
1 #-----任务a-----
2 sigma = 0.5
3 mu = 2
4
5 xdata = st.lognorm.rvs(s = sigma, scale = math.exp(mu), size =
    1000)
6
7 data = pd.DataFrame([])
8 data['x'] = xdata
9
10 data.to_excel('data.xlsx')
```

样本导出至 data.xlsx

任务b&任务c

预处理

Python 下没有较为便捷的可以根据数据量自动选择 KDE 带宽的函数，此处预先用 Matlab 中的 ksdensity 计算了自动选择出的带宽：

```
1 [num_e3, txt_e3, raw_e3] = xlsread('data.xlsx', 'B2:B1001');
2 [f_e3, xi_e3, bw_e3] = ksdensity(num_e3);
3 disp(bw_e3);
```

求解出的推荐带宽为1.0223

实现代码

```
1  #-----任务b-----
2  plt.figure()
3  # X: 符合对数正态分布的数据
4  xdata = pd.read_excel('data.xlsx')
5  xdata = np.array(xdata['x'])
6  xdata = np.sort(xdata)
7  X = xdata[:, np.newaxis]
8
9  # 创建一个[1,52]范围内包含1200个数据的等差数列X_plot
10 X_plot = np.linspace(1, 52, 1200)[: , np.newaxis]
11 # 计算mu = 2 sigma = 0.5的对数正态分布的真实概率密度
12 sigma = 0.5
13 mu = 2
14 true_dens = st.lognorm.pdf(X_plot[:, 0], s = sigma, scale =
    math.exp(mu))
15 plt.plot(X_plot[:, 0], true_dens, 'b-', label='true') # 绘制真实数
    据
16
17 # 求出最佳带宽 求解出的最佳带宽为:0.6428073117284322
18 bandwidths = 10 ** np.linspace(-1, 1, 100)
19 grid = GridSearchCV(KernelDensity(kernel='gaussian'),
    {'bandwidth': bandwidths}, cv=LeaveOneOut())
20 grid.fit(X)
21 best_KDEbandwidth = grid.best_params_['bandwidth']
22 print(best_KDEbandwidth)
23
24 #-----任务c-----
25 # 通过matlab的ksdensity函数求解出推荐的带宽为1.0223
26 # 依次选取带宽
27 bands = [0.6428073117284322, 1.0223, 0.2, 5]
28 colors = ["r", 'y', "g", "darkorange"]
29 for i in range(len(bands)):
30     # 用X数据训练模型
31     kde = KernelDensity(kernel='gaussian',
        bandwidth=bands[i]).fit(X)
32     # 在X_plot数据上测试
33     log_dens = kde.score_samples(X_plot)
34     # 画图
35     lab = 'bandwidth=' + str(bands[i])
36     if i == 0:
```

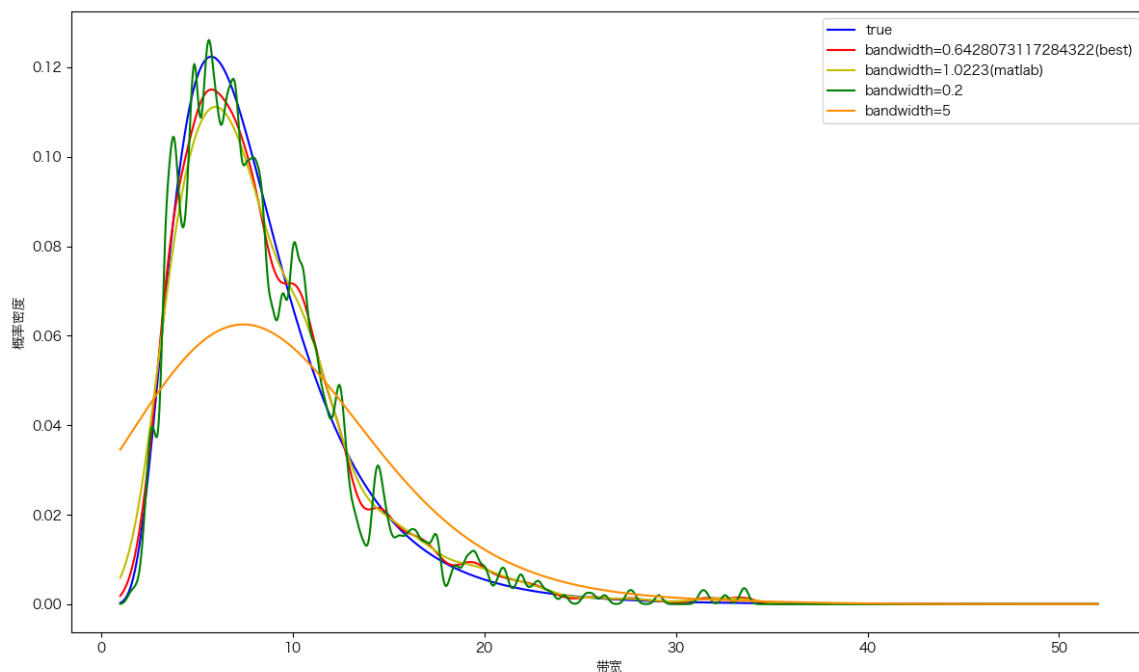
```

37     lab = lab + '(best)'
38     elif i == 1:
39         lab = lab + '(matlab)'
40     plt.plot(X_plot[:, 0], np.exp(log_dens), color=colors[i],
41              label=lab) # kde数据
42
43 plt.xlabel('带宽')
44 plt.ylabel('概率密度')
45 plt.legend()
46 plt.show()

```

为了进行对比，此处用 Python 中的 GridSearchCV 函数对带宽进行了交叉验证调优，求解出最优带宽为0.6428073117284322

导出图像



如图 对比了带宽分别取最优带宽、Matlab 推荐带宽、0.2、5时 KDE 预测的概率密度函数

结果分析

略

任务d

实现代码

```
1 [num_e3, txt_e3, raw_e3] = xlsread('data.xlsx', 'B2:B1001');
2 [num_e4, txt_e4, raw_e4] = xlsread('data_1e4.xlsx', 'B2:B10001');
3 [num_e5, txt_e5, raw_e5] = xlsread('data_1e5.xlsx',
   'B2:B100001');
4
5 [f_e3, xi_e3, bw_e3] = ksdensity(num_e3);
6 [f_e4, xi_e4, bw_e4] = ksdensity(num_e4);
7 [f_e5, xi_e5, bw_e5] = ksdensity(num_e5);
8
9 disp(bw_e3);
10 disp(bw_e4);
11 disp(bw_e5);
```

使用 `Matlab` 求解出了样本量分别为 `1000`、`10000`、`100000` 时 `ksdensity` 函数的推荐带宽

运行结果

```
>> main
样本量: 1000
    1.0223

样本量: 10000
    0.6047

样本量: 100000
    0.3744
```

不难看出，随着样本量逐渐增大，自动选择的带宽有逐渐减小的趋势

结果分析

略