第四次作业

姓名：杨金泰 学号 201621010609

说明：本文分类函数引用 Scikit-learn 官网的 Python 工具箱，所以所有代码都是基于该工具箱。

1. Consider a (hard margin) support vector machine and the following training data from two classes:

+1 : (2; 2) (4; 4) (4; 0)

-1 : (0; 0) (2; 0) (0; 2)

(a) Plot these six training points, and construct by inspection the weight vector for the optimal hyperplane. In your solution, specify the hyperplane in terms of ~w and b such that w1x1 + w2x2 + b = 0. Calculate what the margin is (i.e., 2 , where is the distance from the hyperplane to its closest data point), showing all of your work.

(b)What are the support vectors ? Explain why.

(a)求解，如图所示，蓝色表示-1 类，红色表示+1 类。

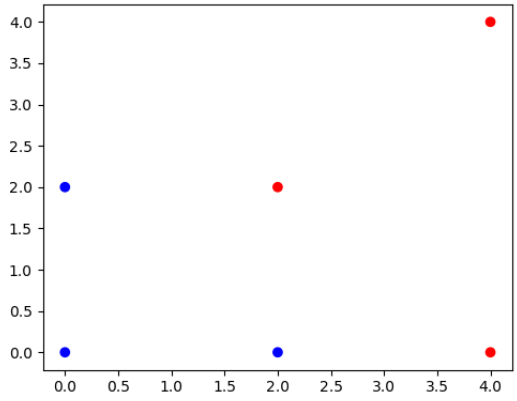

图 1 散点图

本文采用 Scikit-learn 工具箱中 SVM.SVC 构造 SVM 的分类器。因为如图显然为线性可分的，所以核函数采用线性核函数。得到如下结果

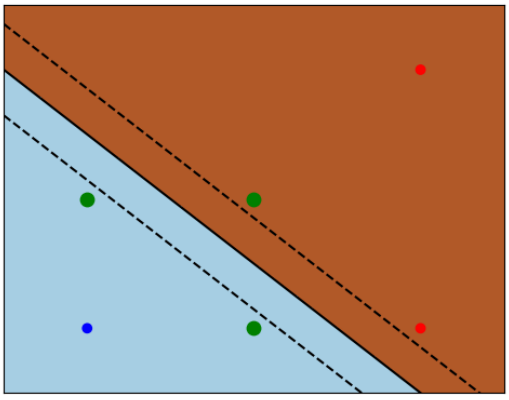

图 2 SVM 分类结果图

其中可以得到 W1=W2=1，截距 b=3.0. 如图所示，绿色点表示支持向量，实线表示分离两类的超平面（二维里面是直线）。虚线是经过支持向量的超平面。两虚线间的间隔就是 2*margin，可以得到 margin=$\frac{\sqrt{2}}{2}$.

因为该样本点显然可以线性可分，所以采用硬间隔就可以分类，本文出于学习的目的考虑了软间隔的方式，并对惩罚系数进行了调整，作为对比。结果如下图，可以看出当惩罚系数很小的时候，分类效果不够理想，这是因为“容错能力”过强，当惩罚系数增大，类效果逐渐变好。

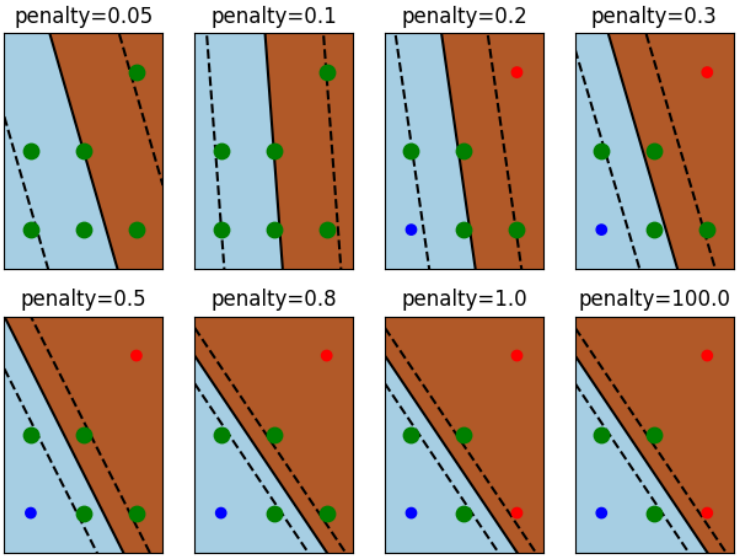

图 3 SVM 软间隔，不同惩罚系数对比

（b）因为当数据点分成若干类时，这些点离分离超平面很显然是最近的，数据分类的时候只依赖于这些点。

2. 给定两个数据集，一个为训练集（train.txt），一个为测试集（test.txt）。每个数据集都是一个 400x3 的矩阵，矩阵的每行对应于一个观察，前两列为 2 维的特征，第 3 列为类别标签，共分两类（0 类和 1 类）。

（1）Scatter plot the two data sets. Show class 0 in red and class 1 in blue.

（2）使用 Matlab 或 Python 在 Logistc Regression, Naive Bayes, MLP, SVM 四种方法中任选两种构造分类器，模型选择使用交叉验证（Cross Validation）。写出最终选用的模型参数以及在测试集上的精度，并上传 Matlab 或 Python 代码。

（1）如图所示，红色点表示 class 0，蓝色表示 class 1.

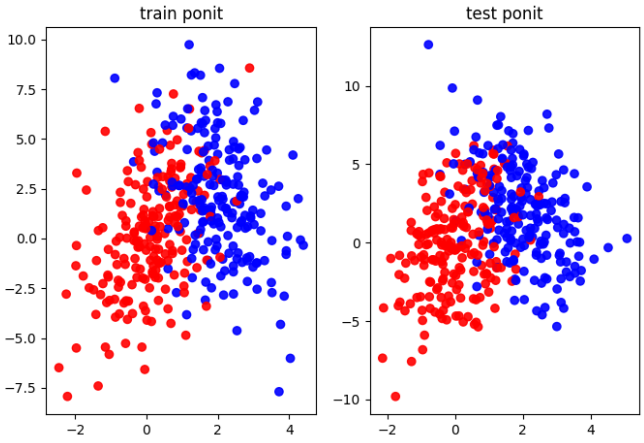

图 4 训练样本和测试样本散点图

（2）**模型横向对比：**首先本文先采用 Scikit-learn 中默认参数得到不同分类器的分类结果，如下图所示，分类结果图中，右下角数字表示该模型的预测精度。

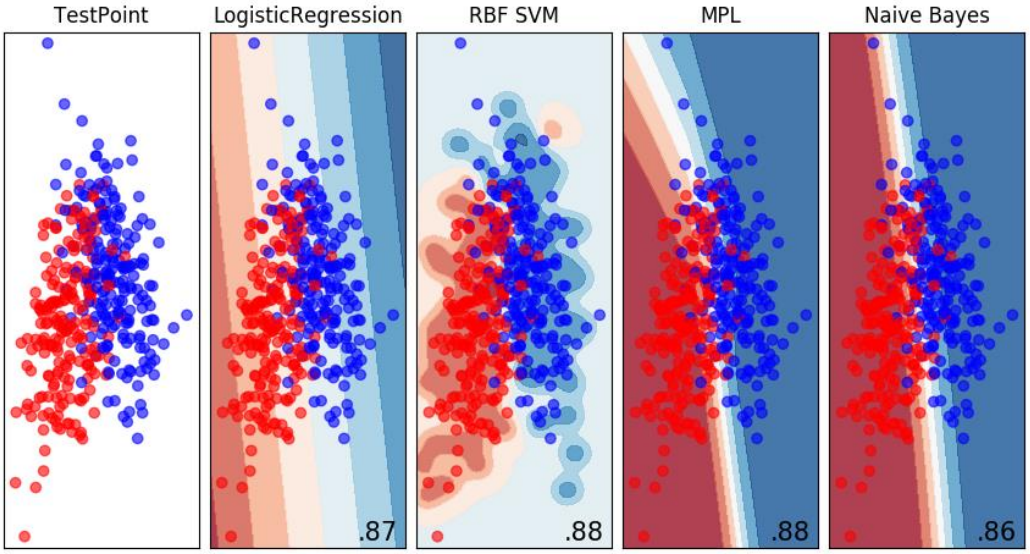

图 5 各类算法测试大致比较

其中第 2 幅图为 Logistics 回归测试结果，可以看出改模型的测试分类面为线性分类面，所以在该数据样本下精度一般，第三幅图为以指数核函数作为核的支持向量分类模型，第四幅多层感知机的分类结果，最后是朴素贝叶斯模型分类结果。

模型参数纵向对比，选取最佳参数；以该结果作为参考，本文选用 Logistics 回归模型和支持向量分类模型。

对于逻辑回归模型，分别选用测试精确率（precision）和召回率（recall）作为评价标准。当测试精确率（precision）作为参考时，得到最好的参数分别为{'penalty': 'l2', 'C': 1}，即惩罚函数采用分类平面系数的 L2 范数，惩罚系数为 C=1.。当召回率（recall）作为参考时，得到最好的参数分别为{'penalty': 'l1', 'C': 1}，即惩罚函数采用分类平面系数的 L1 范数，惩罚系数为 C=1. 参数选择范围和具体运行结果如下。

Tuning hyper-parameters for precision
Best parameters set found on development set:
{'penalty': 'l2', 'C': 1}
Grid scores on development set:
[mean(+/-std)] for parameter
0.860 (+/-0.099) for {'penalty': 'l1', 'C': 0.1}
0.860 (+/-0.080) for {'penalty': 'l1', 'C': 1}
0.857 (+/-0.074) for {'penalty': 'l1', 'C': 10}
0.857 (+/-0.074) for {'penalty': 'l1', 'C': 100}
0.857 (+/-0.074) for {'penalty': 'l1', 'C': 1000}
0.855 (+/-0.096) for {'penalty': 'l2', 'C': 0.1}
0.860 (+/-0.064) for {'penalty': 'l2', 'C': 1}
0.857 (+/-0.074) for {'penalty': 'l2', 'C': 10}
0.857 (+/-0.074) for {'penalty': 'l2', 'C': 100}
0.857 (+/-0.074) for {'penalty': 'l2', 'C': 1000}

Tuning hyper-parameters for recall
Best parameters set found on development set:
{'penalty': 'l1', 'C': 1}
Grid scores on development set:
[mean(+/-std)] for parameter
0.855 (+/-0.112) for {'penalty': 'l1', 'C': 0.1}
0.858 (+/-0.086) for {'penalty': 'l1', 'C': 1}
0.855 (+/-0.080) for {'penalty': 'l1', 'C': 10}
0.855 (+/-0.080) for {'penalty': 'l1', 'C': 100}
0.855 (+/-0.080) for {'penalty': 'l1', 'C': 1000}
0.850 (+/-0.108) for {'penalty': 'l2', 'C': 0.1}
0.858 (+/-0.072) for {'penalty': 'l2', 'C': 1}
0.855 (+/-0.080) for {'penalty': 'l2', 'C': 10}
0.855 (+/-0.080) for {'penalty': 'l2', 'C': 100}
0.855 (+/-0.080) for {'penalty': 'l2', 'C': 1000}

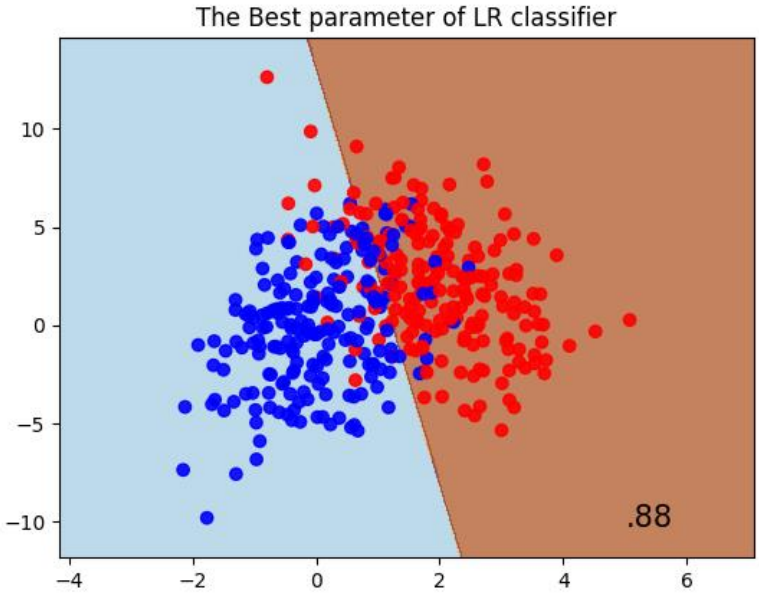

图 6　最好参数 LR 的分类结果

对于支持向量机分类模型，同样分别选用测试精确率（precision）和召回率（recall）作为评价标准。当测试精确率（precision）作为参考时，得到最好的参数分别为{'kernel': 'rbf', 'C': 1000, 'gamma': 0.01}，即 SVC 的核函数采用指数核最佳，惩罚系数为 C=1000，指数函数 decay 系数为 0.001。当召回率（recall）作为参考标准时，得到最好的参数分别为{'kernel': 'rbf', 'C': 100, 'gamma': 0.01}，即核函数还是指数核函数最佳，惩罚系数为 C=100. 指数函数 decay 系数为 0.01 参数选择范围和具体运行结果如下。

Tuning hyper-parameters for precision
Best parameters set found on development set:
{'kernel': 'rbf', 'C': 1000, 'gamma': 0.01}
Grid scores on development set:
[mean(+/-std)] for parameter
0.829 (+/-0.135) for {'kernel': 'rbf', 'C': 0.1, 'gamma': 0.01}
0.855 (+/-0.092) for {'kernel': 'rbf', 'C': 0.1, 'gamma': 0.1}
0.853 (+/-0.067) for {'kernel': 'rbf', 'C': 0.1, 'gamma': 1.0}
0.791 (+/-0.104) for {'kernel': 'rbf', 'C': 0.1, 'gamma': 10.0}
0.664 (+/-0.231) for {'kernel': 'rbf', 'C': 0.1, 'gamma': 100.0}
0.860 (+/-0.101) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.01}
0.862 (+/-0.098) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.1}
0.849 (+/-0.085) for {'kernel': 'rbf', 'C': 1, 'gamma': 1.0}
0.787 (+/-0.061) for {'kernel': 'rbf', 'C': 1, 'gamma': 10.0}
0.650 (+/-0.158) for {'kernel': 'rbf', 'C': 1, 'gamma': 100.0}
0.858 (+/-0.085) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.01}
0.851 (+/-0.075) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.1}
0.832 (+/-0.069) for {'kernel': 'rbf', 'C': 10, 'gamma': 1.0}
0.742 (+/-0.068) for {'kernel': 'rbf', 'C': 10, 'gamma': 10.0}
0.641 (+/-0.137) for {'kernel': 'rbf', 'C': 10, 'gamma': 100.0}
0.864 (+/-0.093) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.01}
0.854 (+/-0.058) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.1}
0.817 (+/-0.075) for {'kernel': 'rbf', 'C': 100, 'gamma': 1.0}
0.732 (+/-0.096) for {'kernel': 'rbf', 'C': 100, 'gamma': 10.0}
0.641 (+/-0.137) for {'kernel': 'rbf', 'C': 100, 'gamma': 100.0}
0.864 (+/-0.084) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.01}
0.849 (+/-0.066) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.1}
0.768 (+/-0.110) for {'kernel': 'rbf', 'C': 1000, 'gamma': 1.0}
0.741 (+/-0.101) for {'kernel': 'rbf', 'C': 1000, 'gamma': 10.0}
0.641 (+/-0.137) for {'kernel': 'rbf', 'C': 1000, 'gamma': 100.0}
0.856 (+/-0.073) for {'kernel': 'linear', 'C': 0.1}
0.858 (+/-0.061) for {'kernel': 'linear', 'C': 1}
0.860 (+/-0.068) for {'kernel': 'linear', 'C': 10}
0.860 (+/-0.068) for {'kernel': 'linear', 'C': 100}
0.860 (+/-0.068) for {'kernel': 'linear', 'C': 1000}

Tuning hyper-parameters for recall
Best parameters set found on development set:
{'kernel': 'rbf', 'C': 100, 'gamma': 0.01}
Grid scores on development set:
[mean(+/-std)] for parameter
0.818 (+/-0.141) for {'kernel': 'rbf', 'C': 0.1, 'gamma': 0.01}
0.850 (+/-0.100) for {'kernel': 'rbf', 'C': 0.1, 'gamma': 0.1}
0.848 (+/-0.073) for {'kernel': 'rbf', 'C': 0.1, 'gamma': 1.0}

0.777 (+/-0.122) for {'kernel': 'rbf', 'C': 0.1, 'gamma': 10.0}
0.522 (+/-0.033) for {'kernel': 'rbf', 'C': 0.1, 'gamma': 100.0}

0.858 (+/-0.106) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.01}
0.858 (+/-0.107) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.1}
0.845 (+/-0.096) for {'kernel': 'rbf', 'C': 1, 'gamma': 1.0}
0.777 (+/-0.060) for {'kernel': 'rbf', 'C': 1, 'gamma': 10.0}
0.613 (+/-0.125) for {'kernel': 'rbf', 'C': 1, 'gamma': 100.0}
0.855 (+/-0.093) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.01}
0.848 (+/-0.080) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.1}
0.828 (+/-0.075) for {'kernel': 'rbf', 'C': 10, 'gamma': 1.0}
0.735 (+/-0.064) for {'kernel': 'rbf', 'C': 10, 'gamma': 10.0}
0.608 (+/-0.103) for {'kernel': 'rbf', 'C': 10, 'gamma': 100.0}
0.860 (+/-0.100) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.01}
0.850 (+/-0.063) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.1}
0.812 (+/-0.082) for {'kernel': 'rbf', 'C': 100, 'gamma': 1.0}
0.725 (+/-0.094) for {'kernel': 'rbf', 'C': 100, 'gamma': 10.0}
0.608 (+/-0.103) for {'kernel': 'rbf', 'C': 100, 'gamma': 100.0}
0.860 (+/-0.093) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.01}
0.845 (+/-0.068) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.1}
0.762 (+/-0.111) for {'kernel': 'rbf', 'C': 1000, 'gamma': 1.0}
0.735 (+/-0.102) for {'kernel': 'rbf', 'C': 1000, 'gamma': 10.0}
0.608 (+/-0.103) for {'kernel': 'rbf', 'C': 1000, 'gamma': 100.0}
0.853 (+/-0.083) for {'kernel': 'linear', 'C': 0.1}
0.855 (+/-0.068) for {'kernel': 'linear', 'C': 1}
0.858 (+/-0.072) for {'kernel': 'linear', 'C': 10}
0.858 (+/-0.072) for {'kernel': 'linear', 'C': 100}
0.858 (+/-0.072) for {'kernel': 'linear', 'C': 1000}

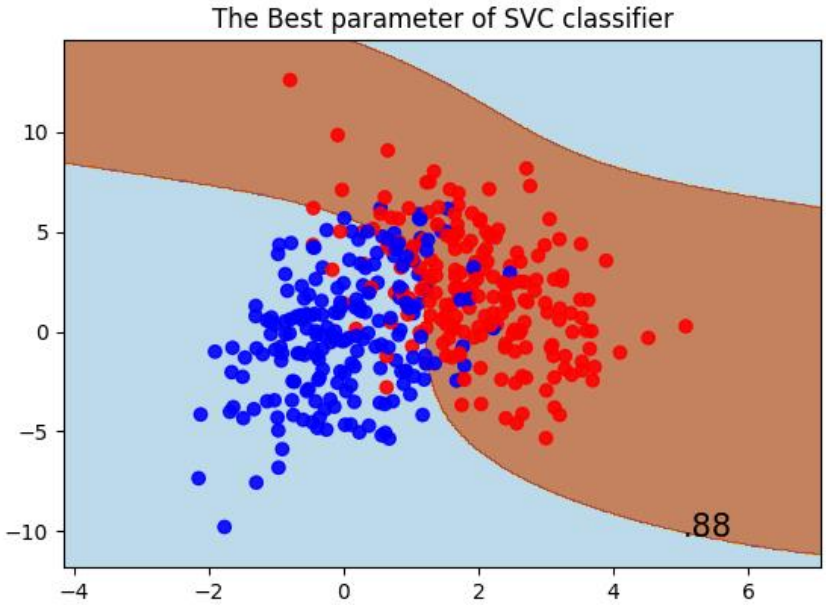

图 7 SVC 最佳参数的分类结果