数据科学作业

一.　基础部分

复现课件中线性 SVM、决策树、朴素贝叶斯分类的示例，并对课件代码进行修改

（1）设定支持向量分类器的惩罚为 0.05

SVC(kernel="linear",C=0.5)其中 C 为惩罚系数，用来控制损失函数的惩罚系数，当进行更改时，使用线性支持向量机分类器的划分的标准与划分正确率会发生改变。

```
classifiers=[
    SVC(kernel="linear",C=0.05),
    DecisionTreeClassifier(random_state=0,max_depth=5),
    GaussianNB(priors=[0.4,0.6])]
```

原则上 C 可以根据需要选择所有大于 0 的数。C 越大表示整个优化过程中对于总误差的关注程度越高，对于减小误差的要求越高，甚至不惜使间隔减小。当 C 趋于无穷大时，这个问题也就是不允许出现分类误差的样本存在，那这就是一个 hard-margin SVM 问题（过拟合）。当 C 趋于 0 时，我们不再关注分类是否正确，只要求间隔越大越好，那么我们将无法得到有意义的解且算法不会收敛（欠拟合）。

过拟合（over-fitting）其实就是所建的机器学习模型或者是深度学习模型在训练样本中表现得过于优越，导致在验证数据集以及测试数据集中表现不佳。
过拟合就是学到了很多没必要的特征，
欠拟合（under-fitting）指训练样本被提取的特征比较少，导致训练出来的模型不能很好地匹配，表现得很差，甚至样本本身都无法高效的识别。

（2）对朴素贝叶斯分类器的先验概率进行设定（可随机设定）

priors 是获取各个类标记对应的先验概率，这里设置两个类的概率为 0.4 和 0.6。当进行更改时，使用朴素贝叶斯分类器的划分的标准与划分正确率会发生改变。

```
classifiers=[
    SVC(kernel="linear",C=0.05),
    DecisionTreeClassifier(random_state=0,max_depth=5),
    GaussianNB(priors=[0.4,0.6])]
```

朴素贝叶斯方法的特点是结合先验概率和后验概率，避免了只使用先验概率的主观偏见，也避免了单独使用样本信息的过拟合现象。贝叶斯定理最大的用处是在很多情况下，需要求的概率是后验概率 P(B|A)，这很难直接求解，但是他的先验概率 P(A|B)却很容易求解，这时候贝叶斯定理就在理论上支持了我们的方法。

（3）在每张结果图上展示图例

```
ax.legend(loc = 1)#添加图例
```

其中 loc=1 指明添加图例到右上角

（4）修改散点颜色为黄和绿

修改 cm_bright=ListedColormap(['#FFFF00','#00FF00']) ，其中'#FFFF00'为黄色，'#00FF00'为绿色。

```
cm_bright=ListedColormap(['#FFFF00','#00FF00'])#设置散点颜色
```

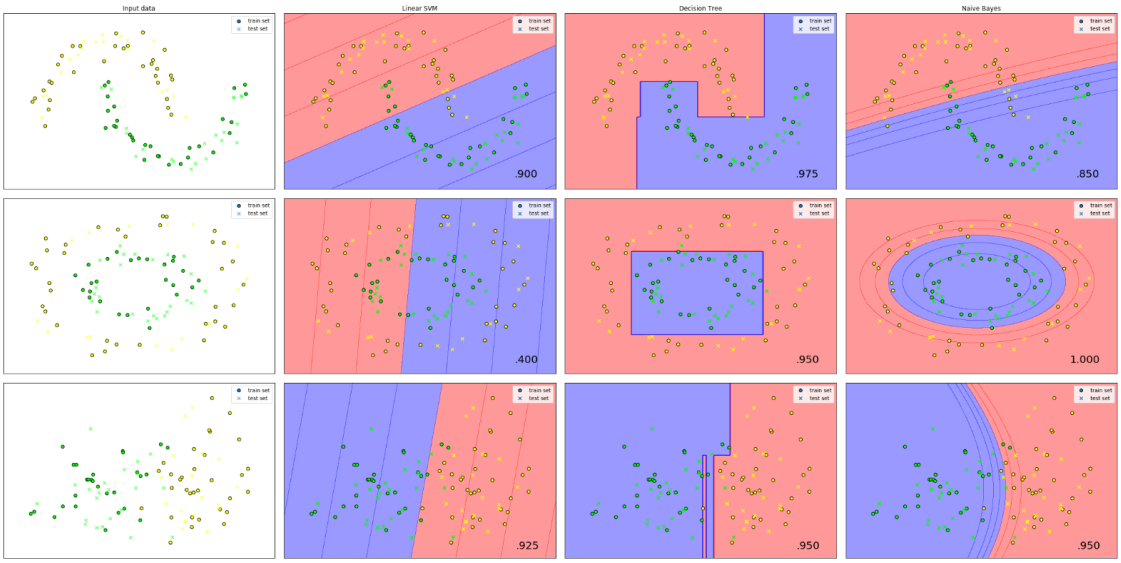
（5）测试结果的正确率保留三位小数展示

修改代码为

ax.text(xx.max()-.3,yy.min()+.3,('%.3f'%score).lstrip('0'),size=20,horizontalalignment='right')其中'%.3f'% score 为控制正确率形式

```
    #显示测试正确率
ax.text(xx.max()-.3,yy.min()+.3,('%.3f'% score).lstrip('0'),
        size=20,horizontalalignment='right')
```

修改上述代码后整体的运行结果为：



二. 拓展部分

1. 自主选取其他的数据集，采用上述三类分类器进行分类，展示分类结果

本次拓展实验选择的数据集为为 sklearn 库中的鸢尾花数据集，其中选用数据中的 sepal length in cm 和 sepal width in cm 作为特征值，同时选取类别中的 SetosaIris 和 Versicolour 作为标签，即选择前 100 个数据利用三类分类器进行分类并展示结果。

具体代码如下

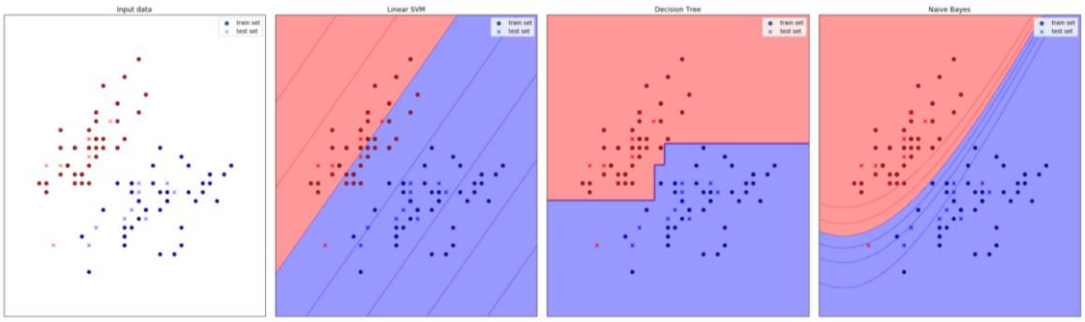
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.svm import SVC#支持向量机
from sklearn.tree import DecisionTreeClassifier#决策树分类器
from sklearn.naive_bayes import GaussianNB#先验为高斯分布的朴素贝叶斯分类器
#导入鸢尾花数据集
from sklearn.datasets import load_iris   #导入数据集iris
iris = load_iris().data  #载入数据集
#data = np.array( iris[:100,:2])#获取花卉两列数据集
#y = np.array(load_iris().target[:100])
data = iris[:100,:2]#获取花卉两列数据集
y = load_iris().target[:100]
print(data.shape,y.shape)
#设置显示结果的图的标题
names=["Linear SVM","Decision Tree","Naive Bayes"]
#设置分类器，用到线性SVM，决策树，朴素贝叶斯
classifiers=[
    SVC(kernel="linear",C=0.05),
    DecisionTreeClassifier(random_state=0,max_depth=5),
    GaussianNB(priors=[0.5,0.5])]
```

```
#设置显示结果图的大小
ii=1
figure=plt.figure(figsize=(27,8))
X_train=[]
X_test=[]
y_train =[]
y_test=[]
for i in range (0,40):
    X_train.append(data[i,:])
    X_train.append(data[i+50,:])
    y_train.append(y[i])
    y_train.append(y[i+50])
for i in range (40,50):
    X_test.append(data[i,:])
    X_test.append(data[i+50,:])
    y_test.append(y[i])
    y_test.append(y[i+50])
print(y_train)
X_train = np.array(X_train)
X_test = np.array(X_test)
y_train = np.array(y_train)
y_test = np.array(y_test)
x_min,x_max=data[:,0].min()-.5,data[:,0].max()+.5 #取第一个取值特征范围作为横轴
y_min,y_max=data[:,1].min()-.5,data[:,1].max()+.5 #取第二个取值特征范围作为纵轴
h=.02 #设置网络的步长
xx,yy=np.meshgrid(np.arange(x_min,x_max,h), #按步长在横纵轴上设置网络
                  np.arange(y_min,y_max,h))
#先展示输入数据集
cm=ListedColormap(['red','blue'])#设置分割面颜色
cm_bright=ListedColormap(['#FF0000','#0000FF'])#设置散点颜色
ax=plt.subplot(1,4,1)#划分子面 3 4 1    345   349分别为第1，5，9子图
#print(len(datasets),len(classifiers)+1,i)
if ii==1:
    ax.set_title("Input data")#初始标题
    #画训练集散点
ax.scatter(X_train[:,0],X_train[:,1],c=y_train,cmap=cm_bright,
           edgecolors='k',marker='o',label='train set')
#画测试集散点
ax.scatter(X_test[:,0],X_test[:,1],c=y_test,cmap=cm_bright,alpha=0.6,
           edgecolors='k',marker='x',label='test set')
#画坐标轴
ax.set_xlim(xx.min(),xx.max())
ax.set_ylim(yy.min(),yy.max())
ax.set_xticks(())
ax.set_yticks(())
ax.legend(loc = 1)#添加图例
ii+=1
for name,clf in list(zip(names,classifiers)):#name ,classifiers为初始定义的类别
    #print(name,classifiers)
    #zip() 函数用于将可迭代的对象作为参数，将对象中对应的元素打包成一个个元组，然后返回由这些元组组成的列表
    ax=plt.subplot(1,4,ii)
    print(X_train.shape,y_train.shape)
    clf.fit(X_train,y_train)#训练集训练分类器
    score=clf.score(X_test,y_test)#测试集测试分类器
    #hasattr用于是否包含属性的判定
    if hasattr(clf,"decision_function"):
        Z=clf.decision_function(np.c_[xx.ravel(),yy.ravel()])
    else:
        Z=clf.predict_proba(np.c_[xx.ravel(),yy.ravel()])[:,1]
    #将分类结果利用contourf（等高线）函数画出
    Z=Z.reshape(xx.shape)#形状的统一
    #print(Z.shape)
    ax.contourf(xx,yy,Z,cmap=cm,alpha=.4)#等高线的绘制
    #画训练集点
    ax.scatter(X_train[:,0],X_train[:,1],c=y_train,cmap=cm_bright,
               edgecolors='k',marker='o',label='train set')
    #画测试集点
    ax.scatter(X_test[:,0],X_test[:,1],c=y_test,cmap=cm_bright,
               edgecolors='k',marker='x',label='test set')
    #画坐标轴
    ax.set_xlim(xx.min(),xx.max())
    ax.set_ylim(yy.min(),yy.max())
    ax.set_xticks(())
    ax.set_yticks(())
    ax.legend(loc = 1)#添加图例
    if ds_cnt==0:
        #画子图标题
        ax.set_title(name)
    ii += 1

#分别对每个数据集做训练及测试

plt.tight_layout()
plt.show()
```

算法步骤即为获取数据，将数据分为训练集和检测集，利用分类器对训练集数据进行训练得到分类结果，利用检测集数据进行检验。
整体运行之后的结果如下图所示



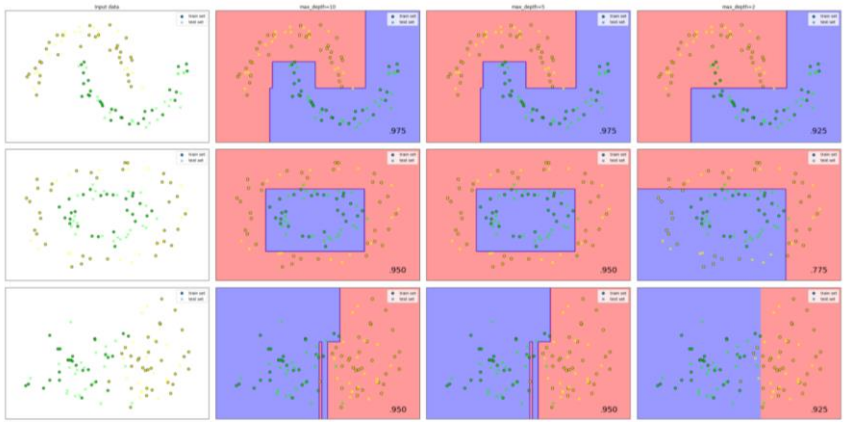
可以看到，使用三种分类器得到的分类效果都比较好，这与原样本数据本身的方差较小有关。

2. 探究分类器的参数对于分类结果的影响并进行文字分析

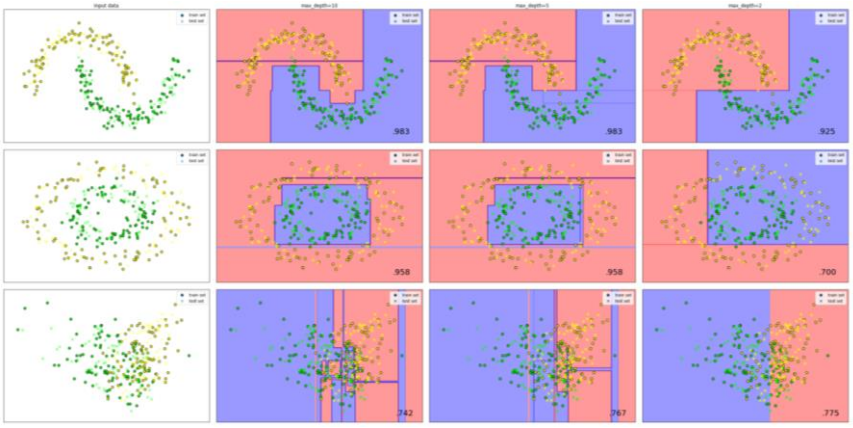
（1）DecisionTreeClassifier(max_depth=5)中 max_depth 设置对于结果的影响（如过拟合或者欠拟合）

max_depth 用于限制树的最大深度，是用的最广的参数，在高维度低样本时非常有效。决策树多生长一层，对样本的需求会增加一倍，所以限制层数能够有效限制过拟合。如果参数太小，可能会欠拟合，但是数字太大，可能会过拟合；通过准确率的显示也可以看出并非深度越深，数据集的分类结果会越准确，因此实际应用时需要根据数据集数目及类型选择恰当的决策树深度。

以下为样本集为 100 时，决策树深度分别为 10，5，2 的结果



以下为样本集为 300 时，决策树深度分别为 10，5，2 的结果



（2）朴素贝叶斯分类器的先验概率修改对于分类的影响

首先明确概念：贝叶斯决策论是概率框架下实施决策的基本方法，而贝叶斯分类器的理论框架基于贝叶斯决策论。对于分类任务，在所有相关概率都已知的情况下，贝叶斯决策论考虑如何基于这些概率和误判损失来选择最优的类别标记。

先验概率：根据以往的经验和分析得到的概率，也即根据客观事实和统计频率得出的概率；

条件概率：在某种条件下某件事发生的概率；

后验概率：事件已经发生，而要得到这件事情发生的原因是某个因素引起的可能性，即由结果推原因。

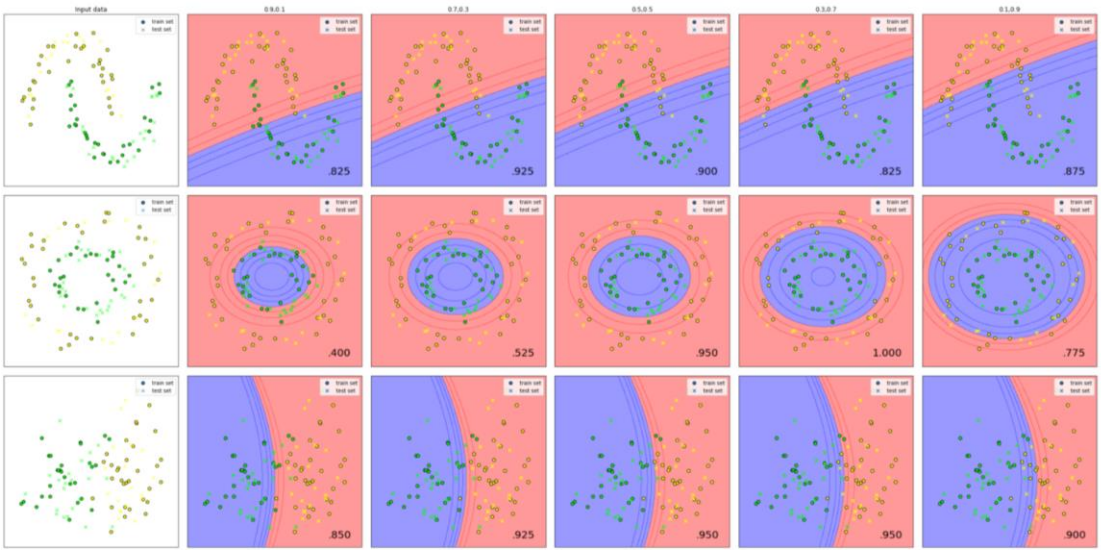
贝叶斯判定准则：决策目标是最小化总体风险，即分类器错误率，于是要对每个样本 x 选择使得后验概率 P(c|x)最大的类别标记（即使条件风险 R(c|x)最小的类别标记）

贝叶斯分类器使用生成式模型的策略（朴素贝叶斯）

生成式模型：对联合概率分布 P(x,c)先建立模型，再得到 P(c|x)

判别式模型：给定了 x，然后建立模型 P(c|x)来预测 c

实验效果：如下图为分别设置先验概率（采用 GaussianNB $P(x)=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$ ）为 "0.9,0.1","0.7,0.3","0.5,0.5","0.3,0.7","0.1,0.9"的结果



实验总结：修改不同的先验概率，划分结果与准确率都发生了改变，由于先验概率是根据以前的知识和经验得出的样本出现的概率，这对应于我们划分训练集时，对应标签的样本数据量占总的样本量的概率，因此将先验概率设定为一致时，得到的分类结果此时最好。

（3）支持向量分类器不同核函数对于结果的影响

官网给出四种内置的核函数

线性：$\langle x, x' \rangle$.

多项式：$(\gamma\langle x, x' \rangle + r)^d$. d是关键词 degree，代表多项式的次数， r指定

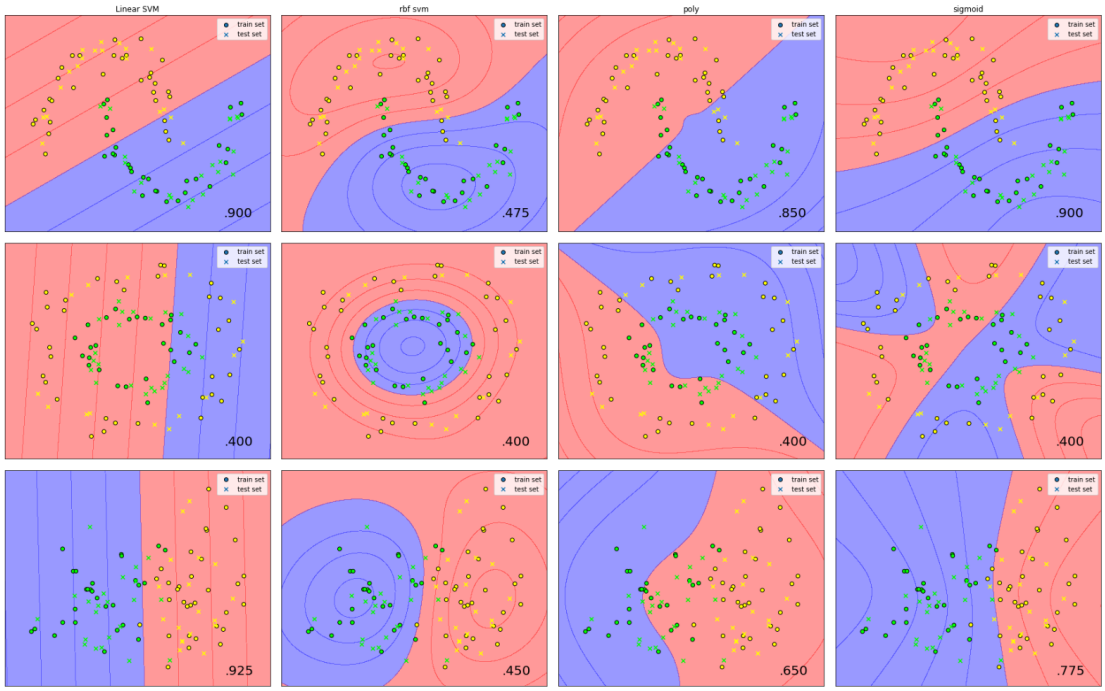
coef0，代表多项式的偏置。

rbf:径向基核函数 $\exp(-\gamma\|x-x'\|^2)$. γ是关键词 gamma，指支持向量影响区域半径的倒数，必须大于 0，gamma 越大，支持向量影响区域越小，决策边界倾向于只包含支持向量，模型复杂度高，gamma 越小，支持向量影响区域越大，决策边界倾向于光滑，模型复杂度低，容易欠拟合；

sigmoid $(\tanh(\gamma\langle x,x'\rangle+r))$，其中 r指定 coef0。

通过参考文章我们可以得知 1.线性核简单，可以求解较快一个 QP 问题，可解释性强：可以轻易知道哪些 feature 比较重要但是只能解决线性可分问题。2.多项式核依靠升维使得原本线性不可分的数据线性可分；解决了非线性问题，可以通过主观设置幂数来实现总结的预判，缺点是对于大数量级的幂数，不太适用 3.高斯核可以映射到无限维，因此决策边界更为多样，且只有一个参数，相比多项式核容易选择，缺点是可解释性差，计算速度比较慢并且容易过拟合 4.采用 Sigmoid 函数作为核函数时，支持向量机实现的就是一种多层感知器神经网络。

以下为对数据分别采用"Linear SVM","rbf svm","poly","sigmoid"四种核函数的分类效果，可以得出结论：对于不同的数据集，需要采取不同的核函数，否则分类效果将不会很好。



（4）其他分类方法的效果的对比分析（K 近邻，随机森林等）

K 近邻算法简单，通过计算欧式距离最近的 K 个点来确定分类结果，但是对于大型数据集来说，由于运算量较大几乎不可实现。

随机森林算法是常用的集成学习算法，泛用性较好。集成算法用一些相对较弱的分类器，独立地就同一批的样本进行训练，然后把各自的学习结果按某种方式整合起来，进行预测。它适合数据集的各个子样本上的多个决策树分类器，并使用平均值来提高预测精度和控制过度拟合。

以下为采用 K 近邻（选取 k=5）算法和随机森林算法得到的分类结果