1. **问题一**

## 1.1数据处理与分析

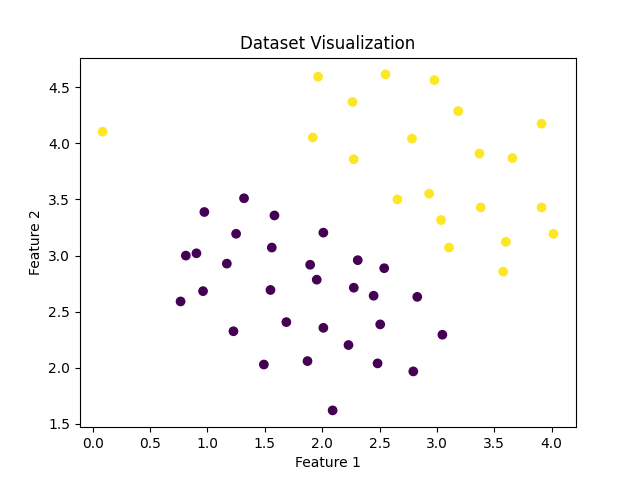
首先，我们加载data1.mat数据集，并使用matplotlib库将其可视化。通过观察数据点的分布，我们可以初步了解数据的特点。数据可视化如图1所示。

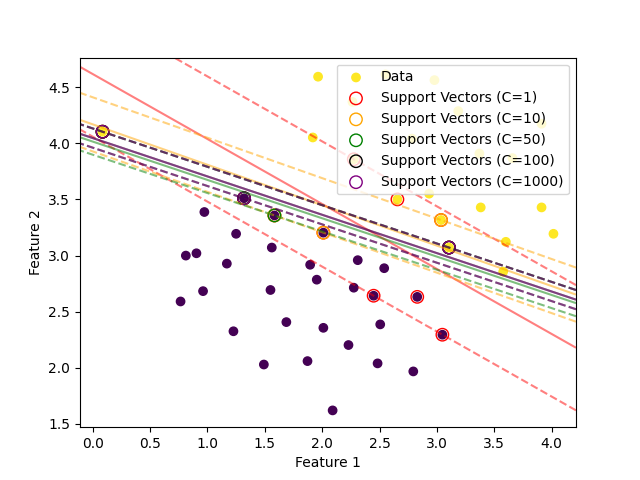
图 1 data1.mat数据可视化

从给定的图片和数据点分布来看，异常点或噪声点并不明显。图片中只有两个明显的簇集：紫色的点主要集中在x轴为1.5的区域，而黄色的点则集中在x轴为2.5的区域。左上角处的黄色点离两个簇均较远，且没有很明显属于某个簇故可能容易被错分。

## 1.2数学建模过程

**软间隔线性SVM**

在标准的线性SVM中，我们假设数据是线性可分的，即存在一个超平面能够完全将不同类别的数据分开。但在实际应用中，这种理想情况并不总是存在。为了处理这种情况，我们引入了软间隔的概念。软间隔允许一些数据点被错误分类，但希望这种错误分类的程度尽可能小。在软间隔线性SVM中，我们引入了一个参数C，用于控制对错误分类的惩罚程度。C值越大，对错误分类的惩罚越重，模型会尽可能地将所有数据点都正确分类，但可能会导致过拟合；C值越小，模型对错误分类的容忍度越高，可能会牺牲一些分类准确性来避免过拟合。

分别取C=1，10，50，100，1000对data1.mat数据集构建软间隔线性SVM模型进行分类。将所有平面画于一个图中，如图2所示，其中C=1为粉线，C=10为黄线，C=50为绿线，C=1000为紫线。

设置了5个参数但图中只有4条线，故猜测有不同参数下分类线相重合导致有线没有显示出来。为了证实该猜测，取C=1和C=100时以及C=1和C=1000时作图比较，其中C=1为粉线。

图 2 不同参数下软间隔线性SVM分类结果示意图

|  |  |
| --- | --- |
|  |  |
| C=1和C=100时 | C=1和C=1000时 |

图 3 比较图

由图3可知，C=100和C=1000时的分类效果相近。由此可知，当参数C到达一定大小之后分类效果改变不会很大，故可以在分类准确度和算法计算量大小上找到一个平衡。从图2可以看出C=50时分类效果已经比较好，为验证使用一些指标对结果进行评价。

## 1.3结果的评价与分析

为了对上述基于不同C值训练的线性SVM模型进行评价，可以采用一系列的性能评估指标，并结合可视化结果来进行综合判断。

**1、准确率（Accuracy）**：

准确率是指在所有样本中，被正确分类的比例。准确率可以衡量模型的整体性能。然而，对于不平衡的数据集，准确率可能不是一个完全可靠的指标，因为它可能受到多数类别样本的支配。

**2、精确率（Precision）：**

Precision是一种评估分类器预测结果正确性的指标，具体定义为：对于某一类，被预测为该类的样本中真正属于该类的比例。换句话说，Precision衡量的是预测为正样本的实例中真正为正样本的比例。

计算公式为：

其中，真正例（True Positives, TP）：实际为正样本，且被分类器正确判断为正样本的实例数量。假正例（False Positives, FP）：实际为负样本，但被错误判断为正样本的实例数量。

Precision的值越高，说明模型预测为正样本的实例中真正为正样本的比例越高，也就是模型预测为正样本时更加“精确”。

**3、召回率（Recall）：**

Recall也是一种评估分类器预测结果正确性的指标，具体定义为：对于某一类，真正属于该类的样本中被预测为该类的比例，即Recall衡量的是所有正样本中被正确预测的比例。

计算公式为：

其中，假反例（False Negatives, FN）：实际为正样本，但被错误判断为负样本的实例数量。

Recall的值越高，说明模型能够找出更多的正样本，也就是模型预测为正样本时更加“全面”。

**4、F1值：**

F1值是精确率和召回率的调和平均数，用于平衡两者之间的权重。F1值越高，说明模型在精确率和召回率之间取得了较好的平衡。

F1-score的值越高，说明模型在Precision和Recall上都表现良好，也就是模型在预测为正样本时既精确又全面。

对上述模型进行评价，结果如表1所示。

表 1 评价指标

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Accuracy | Precision | Recall | F1 |
| C=1 | 1.00 | 1.00 | 1.00 | 1.00 |
| C=10 | 0.94 | 1.00 | 0.86 | 0.92 |
| C=50 | 1.00 | 1.00 | 1.00 | 1.00 |
| C=100 | 1.00 | 1.00 | 1.00 | 1.00 |
| C=1000 | 1.00 | 1.00 | 1.00 | 1.00 |

C=1时评价指标看起来很漂亮但实际运行结果中可以看到C=1时分类出错还是较多，出现这种情况的原因可能为数据量较小，对训练集测试集的划分对结果影响较大。由表1可知，当C=50时模型的分类正确率较高，故综合考虑下应该取C=50比较合适。

## 1.4代码

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  from scipy.io import loadmat  from sklearn import svm  from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_curve, auc  from sklearn.model\_selection import train\_test\_split  # 加载数据  data = loadmat('data1.mat')  X = data['X']  y = data['y'].ravel() # 确保y是一维数组  # 划分数据集为训练集和测试集  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.35, random\_state=42)  # 选择一系列C值来训练SVM  Cs = [1, 10, 50, 100, 1000]  colors = ['red', 'orange', 'green', 'black', 'purple']  # 绘制原始数据  plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis', marker='o', label='Data')  plt.xlabel('Feature 1')  plt.ylabel('Feature 2')  # 遍历C值并绘制决策边界和评估模型  for C, color in zip(Cs, colors):  # 使用SVC创建SVM分类器，并设置C值  clf = svm.SVC(kernel='linear', C=C)  # 拟合模型（仅在训练集上）  clf.fit(X\_train, y\_train)  # 绘制决策函数  ax = plt.gca()  xlim = ax.get\_xlim()  ylim = ax.get\_ylim()  xx = np.linspace(xlim[0], xlim[1], 30)  yy = np.linspace(ylim[0], ylim[1], 30)  YY, XX = np.meshgrid(yy, xx)  xy = np.vstack([XX.ravel(), YY.ravel()]).T  Z = clf.decision\_function(xy).reshape(XX.shape)  # 绘制决策边界和边距  ax.contour(XX, YY, Z, colors=color, levels=[-1, 0, 1], alpha=0.5, linestyles=['--', '-', '--'])  # 标记支持向量  ax.scatter(clf.support\_vectors\_[:, 0], clf.support\_vectors\_[:, 1], s=80, facecolors='none', edgecolors=color,  label=f'Support Vectors (C={C})')  # 评估模型（在测试集上）  y\_pred = clf.predict(X\_test)  accuracy = accuracy\_score(y\_test, y\_pred)  precision = precision\_score(y\_test, y\_pred)  recall = recall\_score(y\_test, y\_pred)  f1 = f1\_score(y\_test, y\_pred)  print(f"C={C}: Accuracy={accuracy:.2f}, Precision={precision:.2f}, Recall={recall:.2f}, F1={f1:.2f}")  # 添加图例  plt.legend(loc='upper right')  plt.show() |

# 2、问题二（1）（2）

## 1.1数据处理与分析

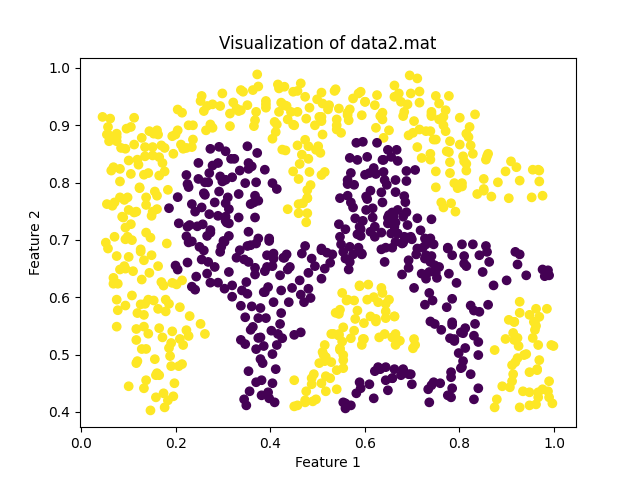
首先，我们加载data2.mat数据集，并使用matplotlib库将其可视化。通过观察数据点的分布，我们可以初步了解数据的特点。数据可视化如图4所示。

图 4 data2.mat数据可视化

从给定的图片和数据点分布来看，异常点或噪声点并不明显。两重数据点有明显的分界，边界为曲线，故考虑用基于高斯核函数的非线性SVM对数据进行分类。首先进行数据预处理，使用train\_test\_split函数将数据集划分为训练集和测试集，并使用StandardScaler对训练集和测试集的特征进行标准化处理，以确保模型训练的有效性。由于sklearn有些函数要求数据为一维，将标签数组y\_train和y\_test转换为一维数组。

## 1.2数学建模过程

高斯核函数（Radial Basis Function, RBF）也被称为径向基核函数或高斯核函数，它将输入空间的数据映射到一个新的特征空间。在这个新空间中，SVM更有可能找到能够将数据分隔开的超平面。高斯核函数的形式是：

其中，x和y是输入数据点，是高斯核的宽度参数，决定了数据点映射后的分布。在SVM的训练过程中，我们需要计算所有训练数据点之间的核函数值，形成一个核矩阵。这个矩阵的每一行和每一列都代表一个数据点，而矩阵中的元素是两个数据点之间的核函数值。

将错误项的惩罚参数C设成50，高斯核函数的系数设成scale，即为

其中，n是输入特征的数量，X.var()是输入数据X的方差。

最后将分类超平面图像输出，如图5所示。

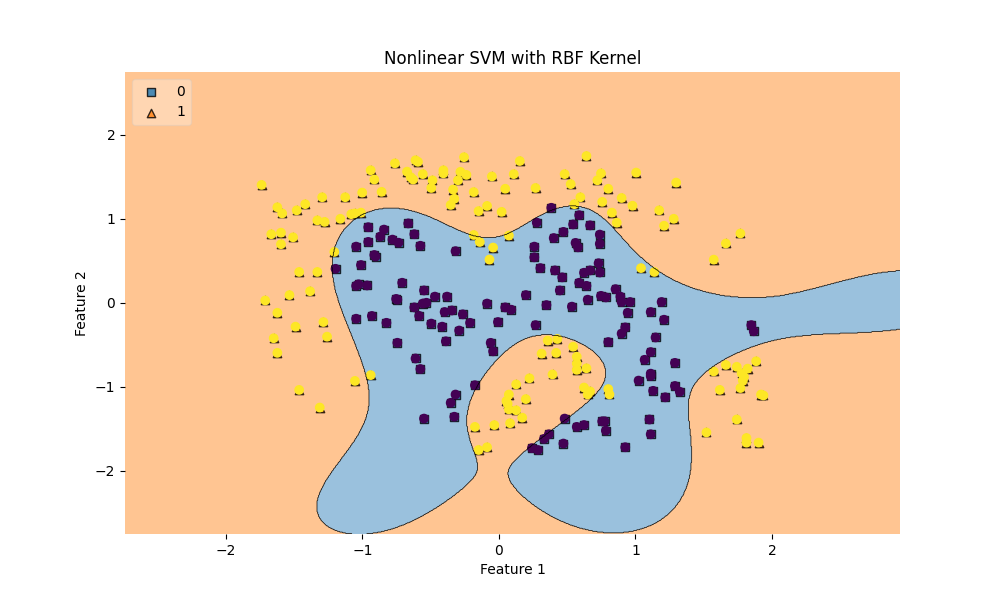


图 5 用基于高斯核函数非线性SVM分类结果示意图

从图5可见只有少数的数据点分类错误，故该模型分类效果较好。

## 1.3结果的评价与分析

为了对上述基于高斯核函数非线性SVM模型进行评价，可以采用一系列的性能评估指标，并结合可视化结果来进行综合判断。运行结果如图6所示。

首先分类器的准确度为0.9536679536679536，表示在所有样本中，被正确分类的样本所占的比例，即模型正确分类了大约95.37%的样本。

混淆矩阵中第一行第一列（119）表示实际为正类（标记为0）且被预测为正类的样本数。第一行第二列（2）表示实际为正类但被预测为负类的样本数（即误报或假阳性）。第二行第一列（10）表示实际为负类（标记为1）但被预测为正类的样本数（即漏报或假阴性）。第二行第二列（128）表示实际为负类且被预测为负类的样本数。

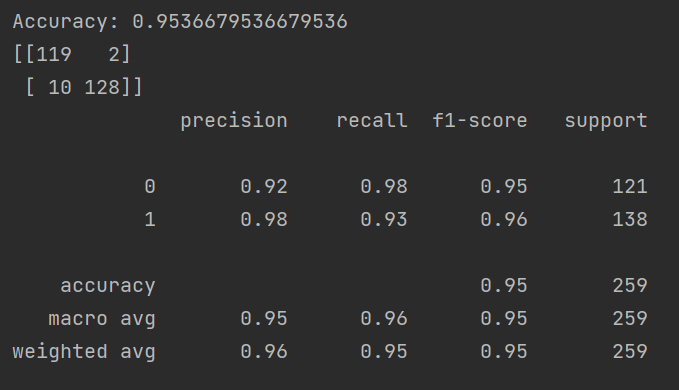
**precision（精确度）**：对于某一类，被预测为该类的样本中真正属于该类的比例。**recall（召回率）**：对于某一类，真正属于该类的样本中被预测为该类的比例。**f1-score**：precision和recall的调和平均值，用于综合评估分类器的性能。**support**：每一类中的样本数。由图结果可知模型预测为正类的样本中有92%是真正的正类；而对于所有真正的正类样本，模型找出了其中的98%。对于负类，模型预测为负类的样本中有98%是真正的负类；而对于所有真正的负类样本，模型找出了其中的93%。

图 6 对模型的评价指标

**macro avg**：不考虑各类样本数目的权重，直接计算所有类的precision、recall和f1-score的平均值。**weighted avg**：考虑各类样本数目的权重，根据各类样本数占总样本数的比例，计算precision、recall和f1-score的加权平均值。由图6结果可知，macro avg和weighted avg的precision、recall和f1-score都接近0.95，这表示模型在两类上的性能相对均衡。

综上所述，模型在分类效果比较好，准确度接近95%，且两类之间的性能差异不大。

## 1.4代码

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  from sklearn import svm  from sklearn.model\_selection import train\_test\_split  from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report  from sklearn.preprocessing import StandardScaler  from scipy.io import loadmat  from mlxtend.plotting import plot\_decision\_regions  # 加载data2.mat数据集  data = loadmat('data2.mat') # 假设data2.mat在你的工作目录下  X = data['X'] # 假设数据矩阵名为X  y = data['y'] # 假设标签矩阵名为y  # 可视化数据  if X.shape[1] == 2:  plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis', marker='o')  plt.xlabel('Feature 1')  plt.ylabel('Feature 2')  plt.title('Visualization of data2.mat')  plt.show()  else:  print("Cannot visualize data in more than 2 dimensions.")  # 划分训练集和测试集  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)  # 数据标准化  scaler = StandardScaler()  X\_train\_scaled = scaler.fit\_transform(X\_train)  X\_test\_scaled = scaler.transform(X\_test)  # 确保y\_train和y\_test都是一维数组  y\_train = y\_train.ravel()  y\_test = y\_test.ravel()  # 创建基于高斯核函数的SVM分类器  clf = svm.SVC(kernel='rbf', C=50.0, gamma='scale')  # 训练模型  clf.fit(X\_train\_scaled, y\_train)  # 预测测试集  y\_pred = clf.predict(X\_test\_scaled)  # 评估模型  accuracy = accuracy\_score(y\_test, y\_pred)  print("Accuracy:", accuracy)  print(confusion\_matrix(y\_test, y\_pred))  print(classification\_report(y\_test, y\_pred))  # 展示分类结果和决策边界（仅对二维数据有效）  if X\_train\_scaled.shape[1] == 2:  plt.figure(figsize=(10, 6))  plot\_decision\_regions(X\_test\_scaled, y\_test, clf=clf, legend=2)  plt.scatter(X\_test\_scaled[:, 0], X\_test\_scaled[:, 1], c=y\_test, cmap='viridis', marker='o')  plt.xlabel('Feature 1')  plt.ylabel('Feature 2')  plt.title('Nonlinear SVM with RBF Kernel')  plt.show()  else:  print("Cannot visualize decision boundary in more than 2 dimensions.") |

# 2、问题二（3）（4）

## 1.1数据处理与分析

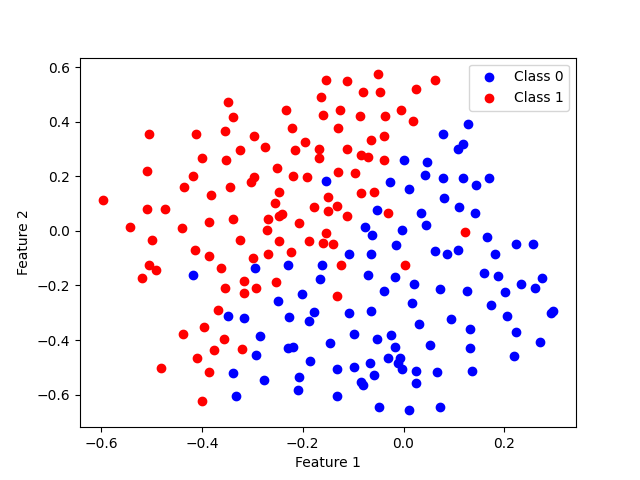
首先，我们加载data3.mat数据集，并使用matplotlib库将其可视化。通过观察数据点的分布，我们可以初步了解数据的特点。数据可视化如图7所示。

图 7 data3.mat可视化

从给定的图片和数据点分布来看，尽管异常点或噪声点并不明显，但两重数据点之间存在大致的分界，同时也有部分数据点交叉融合在边界附近，形成了曲线边界。为了准确地对数据进行分类，需要使用能够处理非线性关系的分类器，故考虑使用基于高斯核函数非线性SVM模型进行分类。

## 1.2数学建模过程

建模过程同2中建模过程。为找到最佳的参数C和σ，遍历C和σ的全部组合（C,σ∈{0.01,0.03,0.1,0.3,1,3,10,30}），找出最好的C和σ。通过一个双层循环来遍历所有的参数组合，并在每个组合下构建SVM模型。我们使用5折交叉验证来评估模型的性能，即将数据集分为5份，每次使用4份作为训练集，1份作为验证集，重复5次，并计算平均得分。在这个循环中，我们为每个(C, σ)组合训练一个SVM模型，并使用5折交叉验证来计算其平均性能得分。如果当前组合的得分高于之前记录的最佳得分，我们就更新最佳得分和对应的参数值。

经过遍历所有参数组合后，我们找到了最佳的C和σ值：C=30和σ=3。这些参数值将使得SVM模型在给定数据集上获得最佳的性能。最后，我们将这些最佳参数代入SVM模型，并使用整个数据集（或者一个独立的测试集）来训练模型，并绘制分类示意图，如图8所示。

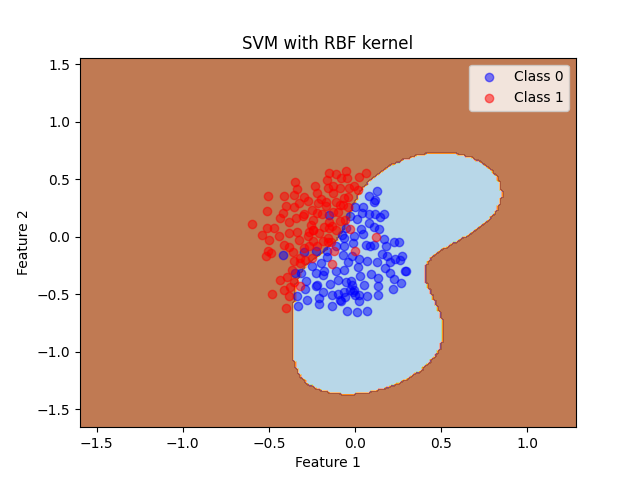
在图8中，我们可能看到的是一个全局的超平面边界视图，但由于数据点分布较为集中或者绘图范围过大，边界附近的分类细节可能不够明显。为了解决这个问题，缩小绘图范围并重新绘制图形。

图 8 最佳参数下基于高斯核函数非线性SVM分类结果示意图

图9就是缩小绘图范围重新绘制的图形。在这个新的视图中，我们可以看到数据点在图中更大，边界附近的分类情况也更加清晰。从图中可以看到只有少部分的数据点分类错误，故该模型分类结果较好。

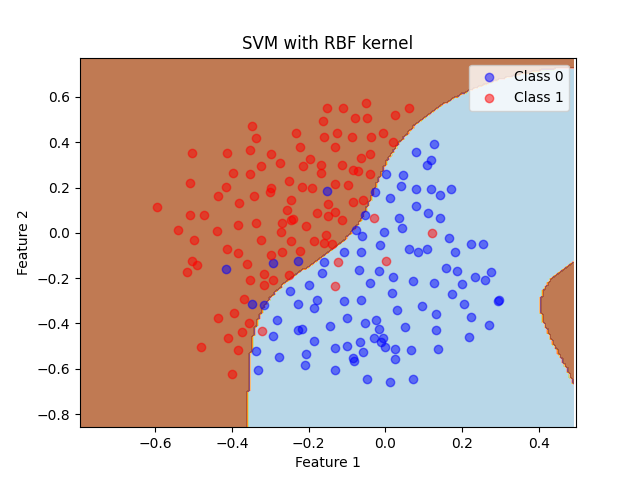


图 9 缩小绘图范围后的分类结果示意图

## 1.3结果的评价与分析

为了对上述基于高斯核函数非线性SVM模型进行评价，可以采用一系列的性能评估指标，指标与前面评估模型指标一致，并结合可视化结果来进行综合判断。

训练准确率为0.943127962085308，这表示模型在训练集上能够正确分类大约94.3%的样本。准确率是一个常用的分类性能指标，但它可能受到数据集不平衡的影响。训练精确度为0.9433962264150944，精确度衡量的是模型预测为正样本的实例中，真正为正样本的比例。它接近训练准确率，说明模型在预测为正样本的实例中，大多数都是真正的正样本。训练F1得分为0.9433962264150944。F1得分是精确度和召回率的调和平均值，用于平衡这两个指标。由于精确度和F1得分在训练集上非常接近，这进一步证实了模型在训练集上的良好性能。

验证准确率为0.95这表明模型在验证集上也能够很好地泛化，正确分类了大约95%的样本。验证准确率略高于训练准确率，这是一个好迹象，说明模型没有过拟合。验证精确度为0.9529411764705882，接近验证准确率，再次表明模型在预测为正样本的实例中表现良好。验证F1得分为0.941860465116279，尽管验证F1得分略低于训练F1得分，但总体上仍然很高，说明模型在验证集上的性能是稳定的。

模型在训练集和验证集上都表现良好，没有出现过拟合的迹象（验证准确率高于或接近训练准确率）。精确度和F1得分也都很高，表明模型在预测为正样本的实例中表现可靠。综上该模型具有较好的分类能力。

## 1.4代码

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  from scipy.io import loadmat  from sklearn import svm  from sklearn.model\_selection import cross\_val\_score  from sklearn.metrics import accuracy\_score, precision\_score, f1\_score  # 加载数据  data = loadmat('data3.mat')  X = data['X']  y = data['y'].flatten() # 将y转化为numpy数组  Xval = data['Xval']  yval = data['yval'].flatten()  # 可视化训练数据  plt.scatter(X[y == 0, 0], X[y == 0, 1], color='blue', label='Class 0')  plt.scatter(X[y == 1, 0], X[y == 1, 1], color='red', label='Class 1')  plt.xlabel('Feature 1')  plt.ylabel('Feature 2')  plt.legend()  plt.show()  Cs = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]  gammas = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30] # 注意这里的值实际上对应于1 / (2 \* sigma^2)  best\_score = -1  best\_C = None  best\_gamma = None  for C in Cs:  for gamma in gammas:  clf = svm.SVC(kernel='rbf', C=C, gamma=gamma)  # 假设我们使用5折交叉验证  scores = cross\_val\_score(clf, X, y, cv=5)  if np.mean(scores) > best\_score:  best\_score = np.mean(scores)  best\_C = C  best\_gamma = gamma  print(f"Best C: {best\_C}, Best gamma: {best\_gamma}, Best score: {best\_score}")  # 使用最佳参数训练SVM  clf = svm.SVC(kernel='rbf', C=best\_C, gamma=best\_gamma)  clf.fit(X, y)  # 预测训练集和测试集  y\_train\_pred = clf.predict(X)  y\_val\_pred = clf.predict(Xval)  # 定义绘制边界的参数范围  ax\_min, ax\_max = X[:, 0].min() - 1, X[:, 0].max() + 1  ay\_min, ay\_max = X[:, 1].min() - 1, X[:, 1].max() + 1  # 创建网格  hx = (ax\_max - ax\_min) / 200  hy = (ay\_max - ay\_min) / 200  xx, yy = np.meshgrid(np.arange(ax\_min, ax\_max, hx), np.arange(ay\_min, ay\_max, hy))  # 使用训练好的SVM模型预测网格上的点  Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()])  # 将预测结果重新组织成网格的形状  Z = Z.reshape(xx.shape)  # 绘制决策边界和样本点  plt.figure()  plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)  plt.scatter(X[y == 0, 0], X[y == 0, 1], color='blue', label='Class 0', alpha=0.5)  plt.scatter(X[y == 1, 0], X[y == 1, 1], color='red', label='Class 1', alpha=0.5)  plt.xlabel('Feature 1')  plt.ylabel('Feature 2')  plt.legend()  plt.title('SVM with RBF kernel')  plt.show()  # 缩小绘图范围，使数据点在图中看起来更大  ax\_min, ax\_max = X[:, 0].min() - 0.2, X[:, 0].max() + 0.2  ay\_min, ay\_max = X[:, 1].min() - 0.2, X[:, 1].max() + 0.2  # 创建网格  hx = (ax\_max - ax\_min) / 200  hy = (ay\_max - ay\_min) / 200  xx, yy = np.meshgrid(np.arange(ax\_min, ax\_max, hx), np.arange(ay\_min, ay\_max, hy))  # 使用训练好的SVM模型预测网格上的点  Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()])  # 将预测结果重新组织成网格的形状  Z = Z.reshape(xx.shape)  # 绘制决策边界和样本点  plt.figure()  plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)  plt.scatter(X[y == 0, 0], X[y == 0, 1], color='blue', label='Class 0', alpha=0.5)  plt.scatter(X[y == 1, 0], X[y == 1, 1], color='red', label='Class 1', alpha=0.5)  plt.xlim(ax\_min, ax\_max) # 设置x轴的范围  plt.ylim(ay\_min, ay\_max) # 设置y轴的范围  plt.xlabel('Feature 1')  plt.ylabel('Feature 2')  plt.legend()  plt.title('SVM with RBF kernel')  plt.show()  # 评估模型  # 计算训练集上的性能  train\_accuracy = accuracy\_score(y, y\_train\_pred)  train\_precision = precision\_score(y, y\_train\_pred, average='binary') # 假设是二分类问题  train\_f1 = f1\_score(y, y\_train\_pred, average='binary') # 假设是二分类问题  # 计算验证集上的性能  val\_accuracy = accuracy\_score(yval, y\_val\_pred)  val\_precision = precision\_score(yval, y\_val\_pred, average='binary') # 假设是二分类问题  val\_f1 = f1\_score(yval, y\_val\_pred, average='binary') # 假设是二分类问题  # 打印结果  print(f"Training Accuracy: {train\_accuracy}")  print(f"Training Precision: {train\_precision}")  print(f"Training F1 Score: {train\_f1}")  print(f"Validation Accuracy: {val\_accuracy}")  print(f"Validation Precision: {val\_precision}")  print(f"Validation F1 Score: {val\_f1}") |