**一、实验目的**

1、利用SVM比较不同核函数和惩罚函数组合在鸢尾花数据分类任务中的性能表现，从而找到最优的模型。

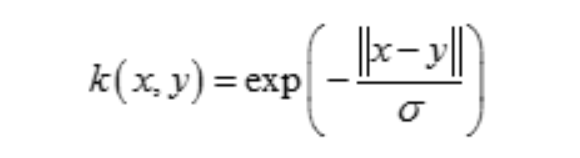
2、学习使用不同的核函数（如线性核、多项式核、高斯核等）以及不同的惩罚函数来训练SVM模型。

3、学习如何评估每种组合的模型在测试集上的分类效果，掌握计算精度、召回率、ROC曲线等性能指标。

1. **实验内容**

**2、解决思路：**

（1）、加载鸢尾花数据集并分割成训练集和测试集。

（2）、创建SVM模型，使用拉普拉斯核函数，Python的Scikit-learn库提供了一些常见的核函数，如线性核、多项式核、高斯核等，但并没有提供拉普拉斯核函数,因此要自定义拉普拉斯核函数。其中，||x-y||表示样本𝑥和𝑦之间的欧氏距离,𝜎是一个控制核宽度的参数。

（3）、定义不同的惩罚参数, 用不同的惩罚参数训练支持向量机模型，比较以得到较优的模型。

（4）、计算准确率、精度、召回率、绘制ROC曲线评估模型的好坏。

（5）、根据（4）得到惩罚参数较好的模型，绘制决策边界。

**3、算法步骤：**

1、导入所需的库和模块，如NumPy 库，从 scikit-learn 库中导入鸢尾花数据集和支持向量机等。

2、自定义了一个拉普拉斯核函数 laplacian\_kernel，用于SVM模型。

3、加载了鸢尾花数据集，只选择了前两个特征进行分类。并将数据集分为训练集和测试集

4、定义了不同惩罚参数 C\_v = [0.1, 1, 10]。

5、使用不同的惩罚参数训练了 SVM 模型，并使用准确率, 精度, 召回率，ROC曲线评估了分类效果并打印。

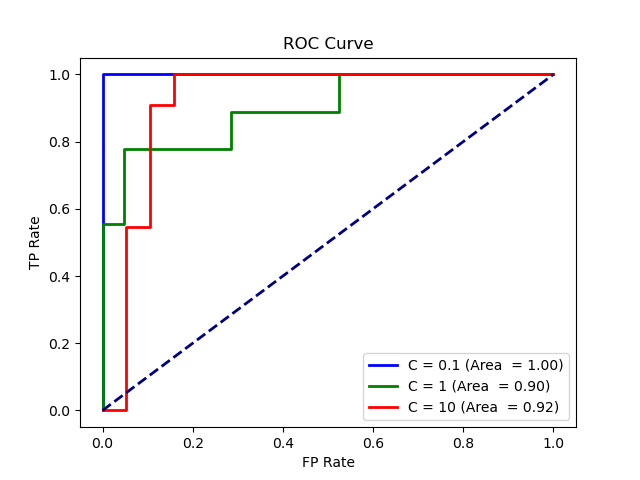
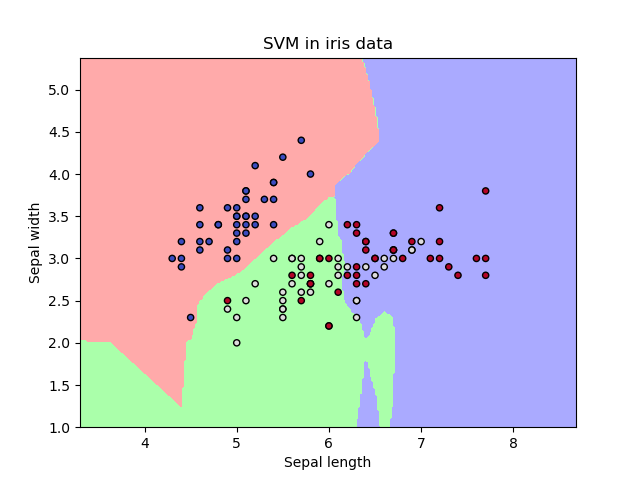
6、使用较为好的惩罚参数为 1 训练了 SVM 模型，绘制了决策边界图和ROC 曲线。

**4、结果分析：**

不同惩罚参数下的模型分类效果:

惩罚参数: 0.1, 准确率: 0.77, 精度: 0.77, 召回率: 0.77

惩罚参数: 1, 准确率: 0.90, 精度: 0.90, 召回率: 0.90

****惩罚参数: 10, 准确率: 0.80, 精度: 0.80, 召回率: 0.80

（AUC 的值越接近 1，表示模型的性能越好）

（由此可以看出惩罚函数＝0.1时，模型性能较好）

但综合考虑惩罚函数为1的时候使用拉普拉斯核函数模型性能较好，而且SVM模型使用拉普拉斯核函数时整体性能也较好。

**5、实验核心代码：**

import numpy as np  
from sklearn.datasets import load\_iris  
from sklearn.svm import SVC  
from sklearn.metrics import roc\_curve  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import label\_binarize  
from sklearn.metrics import accuracy\_score**,** precision\_score**,** recall\_score**,** roc\_auc\_score**,** confusion\_matrix  
from matplotlib.colors import ListedColormap  
import matplotlib.pyplot as plt  
  
  
# Scikit-learn库没有提供拉普拉斯核函数,所以自定义拉普拉斯核函数  
def laplacian\_kernel(X**,** Y):  
 sigma = **1** # 控制核宽度的参数  
 kernel\_matrix = np.zeros((X.shape[**0**]**,** Y.shape[**0**]))  
 for i**,** x in enumerate(X):  
 for j**,** y in enumerate(Y):  
 kernel\_matrix[i**,** j] = np.exp(-np.linalg.norm(x - y**,** ord=**1**) / sigma)  
 return kernel\_matrix  
  
# 加载鸢尾花数据集  
iris = load\_iris()  
X = iris.data[:**,** :**2**] # 只选择前两个特征  
y = iris.target  
  
# 将数据集分为训练集和测试集  
X\_train**,** X\_test**,** y\_train**,** y\_test = train\_test\_split(X**,** y**,** test\_size=**0.2,** random\_state=**42**)  
  
# 定义不同的惩罚参数  
C\_v = [**0.1, 1, 10**]  
  
# 用不同的惩罚参数训练SVM模型  
results = {}  
for C in C\_v:  
 # 创建SVM模型,不使用默认的核函数  
 svm\_model = SVC(kernel='precomputed'**,** C=C)  
 # 计算核矩阵  
 kernel\_train = laplacian\_kernel(X\_train**,** X\_train)  
 # 在训练集上训练模型  
 svm\_model.fit(kernel\_train**,** y\_train)  
 # 计算测试集的核矩阵  
 kernel\_test = laplacian\_kernel(X\_test**,** X\_train)  
 # 在测试集上进行预测  
 y\_pred = svm\_model.predict(kernel\_test)  
 # 计算准确率  
 accuracy = accuracy\_score(y\_test**,** y\_pred)  
 # 计算混淆矩阵为了计算精确率、召回率  
 conf\_matrix = confusion\_matrix(y\_test**,** y\_pred)  
  
 # 计算精度和召回率（对于多分类问题，precision和recall采用micro平均）  
 precision = precision\_score(y\_test**,** y\_pred**,** average='micro')  
 recall = recall\_score(y\_test**,** y\_pred**,** average='micro')  
  
 # 存储结果  
 results[C] = {'Accuracy': accuracy**,** 'Precision': precision**,** 'Recall': recall}  
  
# 打印结果  
print("不同惩罚参数下的模型分类效果:")  
for C**,** metrics in results.items():  
 print("惩罚参数: {}, 准确率: {:.2f}, 精度: {:.2f}, 召回率: {:.2f}".format(C**,** metrics['Accuracy']**,** metrics['Precision']**,** metrics['Recall']))  
  
# 惩罚函数为1时,模型较好  
svm\_model = SVC(kernel='precomputed'**,** C=**1**)  
kernel\_train = laplacian\_kernel(X\_train**,** X\_train)  
svm\_model.fit(kernel\_train**,** y\_train)  
  
# 绘制决策边界  
def plot\_SVM(X**,** y**,** model**,** title):  
 h = **.02** # 网格步长  
 x\_min**,** x\_max = X[:**, 0**].min() - **1,** X[:**, 0**].max() + **1** y\_min**,** y\_max = X[:**, 1**].min() - **1,** X[:**, 1**].max() + **1** xx**,** yy = np.meshgrid(np.arange(x\_min**,** x\_max**,** h)**,** np.arange(y\_min**,** y\_max**,** h))  
 Z = model.predict(laplacian\_kernel(np.c\_[xx.ravel()**,** yy.ravel()]**,** X\_train))  
 Z = Z.reshape(xx.shape)  
 cmap\_light = ListedColormap(['#FFAAAA'**,** '#AAFFAA'**,** '#AAAAFF'])  
 plt.figure()  
 plt.contourf(xx**,** yy**,** Z**,** cmap=cmap\_light)  
 plt.scatter(X[:**, 0**]**,** X[:**, 1**]**,** c=y**,** cmap=plt.cm.coolwarm**,** s=**20,** edgecolors='k')  
 plt.xlabel('Sepal length')  
 plt.ylabel('Sepal width')  
 plt.title(title)  
 plt.show()  
  
plot\_SVM(X\_train**,** y\_train**,** svm\_model**,** "SVM in iris data")  
  
# 绘制ROC曲线  
def plot\_ROC(X**,** y**,** model):  
 kernel\_test = laplacian\_kernel(X**,** X\_train)  
 y\_proba = svm\_model.decision\_function(kernel\_test)  
 y\_binary = label\_binarize(y**,** classes=[**0, 1, 2**])  
 fpr = dict()  
 tpr = dict()  
 roc\_auc = dict()  
 for i in range(**3**):  
 fpr[i]**,** tpr[i]**,** \_ = roc\_curve(y\_binary[:**,** i]**,** y\_proba[:**,** i])  
 roc\_auc[i] = roc\_auc\_score(y\_binary[:**,** i]**,** y\_proba[:**,** i])  
  
 colors = ['blue'**,** 'green'**,** 'red']  
 penalties = [**0.1, 1, 10**]  
 for i**,** color**,** penalty in zip(range(**3**)**,** colors**,** penalties):  
 plt.plot(fpr[i]**,** tpr[i]**,** color=color**,** lw=**2,** label='C = {} (Area = {:.2f})'.format(penalty**,** roc\_auc[i]))  
  
 plt.plot([**0, 1**]**,** [**0, 1**]**,** color='navy'**,** lw=**2,** linestyle='--')  
 plt.xlabel('FP Rate')  
 plt.ylabel('TP Rate')  
 plt.title('ROC Curve')  
 #放右下角  
 plt.legend(loc="lower right")  
 plt.show()  
  
  
plot\_ROC(X\_test**,** y\_test**,** svm\_model)