山东大学 软件 学院

机器学习 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号： 202200201095 | 姓名： 杨伟康 | | 班级： 22网安 |
| 实验题目：SVM在鸢尾花数据集的实现 | | | |
| 实验学时：4 | | 实验日期： 11.27 | |
| **1.实验目的：**  -深入理解和掌握支持向量机（SVM）算法的原理、构建过程以及分类机制，提升综合素养。  -利用支持向量机分类器实现鸢尾花分类，加深对SVM算法的理解，并为植物学和相关领域的研究提供技术支持和实际应用价值。 | | | |
| **2.引言：**  **背景和动机**  在机器学习和数据科学领域，支持向量机（SVM）是一种强大的分类和回归方法，以其良好的泛化能力和在高维空间中处理复杂数据的能力而受到广泛关注。SVM通过构造一个超平面，以最大化类别间的间隔，从而达到分类的目的。鸢尾花数据集是一个经典的机器学习数据集，因其数据简单且类别明确，常被用作算法演示和性能测试的基准。本文旨在通过实现SVM分类器在鸢尾花数据集上的应用，验证其分类效果，并探讨其在实际应用中的优势和潜力。  **问题描述**  本文要解决的核心问题是：利用支持向量机算法对鸢尾花进行分类。在Python的PyCharm环境中，利用sklearn库中的iris数据集，该数据集包含150个样本，每个样本具有花萼长度、花萼宽度、花瓣长度、花瓣宽度四个特征属性。目标是基于这四个特征值，将鸢尾花正确地分类为Iris-Setosa、Iris-Versicolor和Iris-Virginica三类  **文献综述**  支持向量机算法通过寻找最优超平面实现样本的分类。SVM在处理线性可分和线性不可分问题时均表现出色，通过引入核函数，可以在高维空间中进行非线性分类。与其他分类算法相比，SVM在高维数据和小样本情况下表现优越，尤其在准确率和鲁棒性方面，在鸢尾花数据集上已有多项研究表明SVM能够有效地进行分类。 | | | |
| **3.实验方法：**  **数据集**  Iris鸢尾花数据集（Iris Dataset）包含150个样本，分为三类：Setosa、Versicolor、Virginica。每个样本具有4个特征：花萼长度、花萼宽度、花瓣长度、花瓣宽度。  **数据集划分**  将数据集分为训练集（70%）和测试集（30%），或使用交叉验证进行评估。  **模型选择**  本实验选择的模型是支持向量机分类器。支持向量机通过构造最优超平面进行样本的分类。在本实验中，我们不将使用Python的scikit-learn库来实现SVM分类器，而是自己写出算法来模仿SVM，利用其提供的参数调整功能（如C值和核函数）来优化模型的性能。  **评估指标**  为了评估模型的性能，我们将使用准确率（accuracy）、精确率（precision）、召回率（recall）和F1分数（F1-score）等评估指标。准确率是正确分类的样本数与总样本数的比例；精确率表示被正确分类为正例的样本数与所有被预测为正例的样本数之比；召回率表示被正确分类为正例的样本数与所有实际为正例的样本数之比；F1分数是精确率和召回率的调和平均数，用于综合评估模型的性能。在鸢尾花数据集上，这些评估指标能够全面反映模型的分类效果和性能。 | | | |
| **4.实验过程报告：**  **自己写出算法模拟SVM**  **# coding:UTF-8 import numpy as np  # 鸢尾花数据集 from sklearn import datasets # 随机划分训练集和测试集合 from sklearn.model\_selection import train\_test\_split from sklearn.metrics import confusion\_matrix, precision\_score, recall\_score, f1\_score, accuracy\_score import matplotlib.pyplot as plt # SVM类用于保存需要的参数 class SVM:  def \_\_init\_\_(self, train\_data, train\_label, C, toler, kernel\_option):  *"""  初始化部分  :param train\_data: 训练样本特征  :param train\_label: 训练样本标签  :param C: 惩罚参数  :param toler: 迭代终止条件之一  :param kernel\_option: 选用的核函数  """* self.train\_x = train\_data  self.train\_y = train\_label  self.C = C  self.toler = toler   self.sample\_num = np.shape(train\_data)[0] # 训练样本的个数  self.alphas = np.asmatrix(np.zeros((self.sample\_num, 1))) # 拉格朗日乘子，矩阵类型  self.b = 0 # 偏置  self.error\_tmp = np.asmatrix(np.zeros((self.sample\_num, 2))) # 保存E的缓存  self.kernel\_opt = kernel\_option # 选用的核函数及其参数  self.kernel\_mat = calc\_kernel(self.train\_x, self.kernel\_opt) # 核函数的输出   def cal\_kernel\_value(train\_data, train\_data\_i, kernel\_option):  *"""  计算样本之间的核函数的值  :param train\_data: 训练样本  :param train\_data\_i: 第i个训练样本  :param kernel\_option: 核函数的类型以及参数  :return: 样本之间的核函数的值  """* kernel\_type = kernel\_option[0] # 核函数的类型，分为rbf（高斯核函数）和其他  sample\_num = np.shape(train\_data)[0] # 样本的个数  kernel\_value = np.asmatrix(np.zeros((sample\_num, 1)))   if kernel\_type == 'rbf': # rbf核函数  sigma = kernel\_option[1]  if sigma == 0:  sigma = 1.0  for i in range(sample\_num): # 从0到样本数目循环  diff = train\_data[i, :] - train\_data\_i  kernel\_value[i] = np.exp(diff \* diff.T / (-2.0 \* sigma \*\* 2))  else: # 不使用核函数  kernel\_value = train\_data \* train\_data\_i.T  return kernel\_value   def calc\_kernel(train\_x, kernel\_option):  *"""  计算核函数矩阵  :param train\_x: 训练样本的特征值  :param kernel\_option: 核函数的类型以及参数  :return: 样本的核函数矩阵  """* sample\_num = np.shape(train\_x)[0] # 样本的个数  kernel\_matrix = np.asmatrix(np.zeros((sample\_num, sample\_num))) # 初始化样本之间的核函数矩阵  for i in range(sample\_num):  kernel\_matrix[:, i] = cal\_kernel\_value(train\_x, train\_x[i, :], kernel\_option) # 计算核函数的值  return kernel\_matrix   def cal\_error(svm, alpha\_k):  *"""  计算误差值  :param svm: 模型  :param alpha\_k: 选择出的变量，对应alpha  :return: 误差值  """* output\_k = float(np.multiply(svm.alphas, svm.train\_y).T \* svm.kernel\_mat[:, alpha\_k] + svm.b)  error\_k = output\_k - float(svm.train\_y[alpha\_k])  return error\_k   def update\_error(svm, alpha\_k):  *"""  更新误差值  :param svm: 模型  :param alpha\_k: 选择出的变量，对应alpha  :return: 误差值  """* error = cal\_error(svm, alpha\_k) # 选择样本更新误差值  svm.error\_tmp[alpha\_k] = [1, error]   def select\_second\_sample\_j(svm, alpha\_i, error\_i):  *"""  选择第二个样本  :param svm: 模型  :param alpha\_i: 选择出的第一个变量  :param error\_i: E\_i  :return: 选择出的第二个变量和E\_j  """* # 标记为已被优化  svm.error\_tmp[alpha\_i] = [1, error\_i]  candidateAlphaList = np.nonzero(svm.error\_tmp[:, 0].A)[0]   max\_step = 0  alpha\_j = 0  error\_j = 0   if len(candidateAlphaList) > 1:  for alpha\_t in candidateAlphaList:  if alpha\_t == alpha\_i:  continue  error\_t = cal\_error(svm, alpha\_t)  if abs(error\_t - error\_i) > max\_step:  max\_step = abs(error\_t - error\_i)  alpha\_j = alpha\_t  error\_j = error\_t  else:  alpha\_j = alpha\_i # 随机选择  while alpha\_j == alpha\_i:  alpha\_j = int(np.random.uniform(0, svm.sample\_num))  error\_j = cal\_error(svm, alpha\_j)   return alpha\_j, error\_j   def choose\_and\_update(svm, alpha\_i):  *"""  选择两个alpha值进行更新  :param svm: 模型  :param alpha\_i: 选择出的第一个变量  :return:  """* error\_i = cal\_error(svm, alpha\_i) # 计算第一个样本的E\_i   # 判断选择出的第一个变量是否违反了KKT条件  if (svm.train\_y[alpha\_i] \* error\_i < -svm.toler) and (svm.alphas[alpha\_i] < svm.C) or \  (svm.train\_y[alpha\_i] \* error\_i > svm.toler) and (svm.alphas[alpha\_i] > 0):   # 1、选择第二个变量  alpha\_j, error\_j = select\_second\_sample\_j(svm, alpha\_i, error\_i)  alpha\_i\_old = svm.alphas[alpha\_i].copy()  alpha\_j\_old = svm.alphas[alpha\_j].copy()   # 2、计算上下界  if svm.train\_y[alpha\_i] != svm.train\_y[alpha\_j]:  l = max(0, svm.alphas[alpha\_j] - svm.alphas[alpha\_i])  h = min(svm.C, svm.C + svm.alphas[alpha\_j] - svm.alphas[alpha\_i])  else:  l = max(0, svm.alphas[alpha\_j] + svm.alphas[alpha\_i] - svm.C)  h = min(svm.C, svm.alphas[alpha\_j] + svm.alphas[alpha\_i])  if l == h:  return 0   # 3、计算eta  eta = 2.0 \* svm.kernel\_mat[alpha\_i, alpha\_j] - svm.kernel\_mat[alpha\_i, alpha\_i] \  - svm.kernel\_mat[alpha\_j, alpha\_j]  if eta >= 0:  return 0   # 4、更新alpha\_j  svm.alphas[alpha\_j] -= svm.train\_y[alpha\_j] \* (error\_i - error\_j) / eta   # 5、确定最终的alpha\_j  if svm.alphas[alpha\_j] > h:  svm.alphas[alpha\_j] = h  if svm.alphas[alpha\_j] < l:  svm.alphas[alpha\_j] = l   # 6、判断是否结束  if abs(alpha\_j\_old - svm.alphas[alpha\_j]) < 0.00001:  update\_error(svm, alpha\_j)  return 0   # 7、更新alpha\_i  svm.alphas[alpha\_i] += svm.train\_y[alpha\_i] \* svm.train\_y[alpha\_j] \  \* (alpha\_j\_old - svm.alphas[alpha\_j])   # 8、更新b  b1 = svm.b - error\_i - svm.train\_y[alpha\_i] \* (svm.alphas[alpha\_i] - alpha\_i\_old) \  \* svm.kernel\_mat[alpha\_i, alpha\_i] \  - svm.train\_y[alpha\_j] \* (svm.alphas[alpha\_j] - alpha\_j\_old) \  \* svm.kernel\_mat[alpha\_i, alpha\_j]  b2 = svm.b - error\_j - svm.train\_y[alpha\_i] \* (svm.alphas[alpha\_i] - alpha\_i\_old) \  \* svm.kernel\_mat[alpha\_i, alpha\_j] \  - svm.train\_y[alpha\_j] \* (svm.alphas[alpha\_j] - alpha\_j\_old) \  \* svm.kernel\_mat[alpha\_j, alpha\_j]  if (0 < svm.alphas[alpha\_i]) and (svm.alphas[alpha\_i] < svm.C):  svm.b = b1  elif (0 < svm.alphas[alpha\_j]) and (svm.alphas[alpha\_j] < svm.C):  svm.b = b2  else:  svm.b = (b1 + b2) / 2.0   # 9、更新error  update\_error(svm, alpha\_j)  update\_error(svm, alpha\_i)   return 1  else:  return 0   def SVM\_training(train\_data, train\_label, C, toler, max\_iter, kernel\_option=('rbf', 0.431029)):  *"""  训练SVM  :param train\_data: 训练数据  :param train\_label: 训练数据的标签  :param C: 惩罚因子  :param toler: 迭代的终止条件之一  :param max\_iter: 最大迭代次数  :param kernel\_option: 核函数的类型及其参数  :return: 训练完成的模型  """* # 1、初始化  svm = SVM(train\_data, train\_label, C, toler, kernel\_option)   # 2、开始训练  entireSet = True # 整个样本集合  alpha\_pairs\_changed = 0  iteration = 0 # 迭代次数   while (iteration < max\_iter) and ((alpha\_pairs\_changed > 0) or entireSet):  print("第 {} 迭代 ".format(iteration))  alpha\_pairs\_changed = 0   if entireSet:  # 对所有的样本  for x in range(svm.sample\_num):  alpha\_pairs\_changed += choose\_and\_update(svm, x)  iteration += 1  else:  # 非边界样本  bound\_samples = []  for i in range(svm.sample\_num):  if svm.alphas[i, 0] > 0 and svm.alphas[i, 0] < svm.C:  bound\_samples.append(i)  for x in bound\_samples:  alpha\_pairs\_changed += choose\_and\_update(svm, x)  iteration += 1   # 在所有样本和非边界样本之间交替  if entireSet:  entireSet = False  elif alpha\_pairs\_changed == 0:  entireSet = True   return svm   def predict\_one(svm, test\_data\_i):  *"""  依次预测各个样本  :param svm: 模型  :param test\_data\_i: 测试用例  :return: 预测结果  """* # 预测步骤  # 计算核函数矩阵  kernel\_value = cal\_kernel\_value(svm.train\_x, test\_data\_i, svm.kernel\_opt)  # 计算预测值  predict\_label = kernel\_value.T \* np.multiply(svm.train\_y, svm.alphas) + svm.b  return predict\_label   def cal\_accuracy(svm, test\_data, test\_label):  *"""  计算预测准确度  :param svm: 模型  :param test\_data: 测试样本  :param test\_label: 测试标签  :return: 准确度  """* sample\_num = np.shape(test\_data)[0] # 样本的个数  correct = 0.0  for i in range(sample\_num):  # 对每一个样本进行预测值  predict = predict\_one(svm, test\_data[i, :])  if np.sign(predict) == np.sign(test\_label[i]): # 判断正确则加一  correct += 1  accuracy = correct / sample\_num  return accuracy  def cal\_precision(svm, test\_data, test\_label):  *"""  计算精确率  :param svm: 模型  :param test\_data: 测试样本  :param test\_label: 测试标签  :return: 精确率  """* true\_positive = 0.0 # 真正例  false\_positive = 0.0 # 假正例   sample\_num = np.shape(test\_data)[0]  for i in range(sample\_num):  predict = predict\_one(svm, test\_data[i, :])  if np.sign(predict) == 1: # 预测为正类  if np.sign(test\_label[i]) == 1: # 真实为正类  true\_positive += 1  else:  false\_positive += 1   # 避免除以零  if (true\_positive + false\_positive) == 0:  return 0.0  precision = true\_positive / (true\_positive + false\_positive)  return precision  def cal\_recall(svm, test\_data, test\_label):  *"""  计算召回率  :param svm: 模型  :param test\_data: 测试样本  :param test\_label: 测试标签  :return: 召回率  """* true\_positive = 0.0 # 真正例  false\_negative = 0.0 # 假反例   sample\_num = np.shape(test\_data)[0]  for i in range(sample\_num):  predict = predict\_one(svm, test\_data[i, :])  if np.sign(test\_label[i]) == 1: # 真实为正类  if np.sign(predict) == 1: # 预测为正类  true\_positive += 1  else:  false\_negative += 1   # 避免除以零  if (true\_positive + false\_negative) == 0:  return 0.0  recall = true\_positive / (true\_positive + false\_negative)  return recall  def cal\_f1\_score(svm, test\_data, test\_label):  *"""  计算 F1 分数  :param svm: 模型  :param test\_data: 测试样本  :param test\_label: 测试标签  :return: F1 分数  """* precision = cal\_precision(svm, test\_data, test\_label)  recall = cal\_recall(svm, test\_data, test\_label)   # 避免除以零  if (precision + recall) == 0:  return 0.0  f1\_score = 2 \* (precision \* recall) / (precision + recall)  return f1\_score  # 选取测试集合比例 rate = 0.30   # 处理鸢尾花数据，包括data和target,总共150组数据 def creat\_data():  iris = datasets.load\_iris()  # 划分训练集和测试集  train\_data, test\_data, train\_label, test\_label = \  train\_test\_split(iris.data, iris.target, test\_size=rate)   train\_label\_matrix = np.asmatrix(np.zeros((len(train\_label), 1))) # 初始化样本之间的核函数矩阵  for i in range(len(train\_label)):  train\_label\_matrix[i][0] = int(train\_label[i] + 1)   test\_label\_matrix = np.asmatrix(np.zeros((len(test\_label), 1))) # 初始化样本之间的核函数矩阵  for i in range(len(test\_label)):  test\_label\_matrix[i][0] = int(test\_label[i] + 1)   return np.asmatrix(train\_data), train\_label\_matrix, np.asmatrix(test\_data), test\_label\_matrix    if \_\_name\_\_ == "\_\_main\_\_":  print("1、加载训练数据和测试数据")  train\_data, train\_label, test\_data, test\_label = creat\_data()  print("2、设置相关参数，训练SVM模型")  C = 1 # 惩罚因子  toler = 0.001 # 迭代终止的边界值  maxIter = 100 # 增加最大迭代次数  svm\_model = SVM\_training(train\_data, train\_label, C, toler, maxIter)   print("3、使用训练完成的SVM模型计算模型评估标准")  accuracy = cal\_accuracy(svm\_model, test\_data, test\_label)  print("准确率是: %.3f%%" % (accuracy \* 100))  precision = cal\_precision(svm\_model, test\_data, test\_label)  print("精确率是: %.3f%%" % (precision \* 100))  recall = cal\_recall(svm\_model, test\_data, test\_label)  print("召回率：: %.3f%%" % (recall \* 100))  f1 = cal\_f1\_score(svm\_model, test\_data, test\_label)  print("f1分数: %.3f%%" % (f1 \* 100)) 启动查看模型的评估结果**    **经过100次迭代**  **……**      **# 定义不同的训练集比例 rates = [0.1, 0.2, 0.3, 0.4, 0.5] accuracies = [] # 用于存储各个比例下的准确率  for rate in rates:  print(f"正在处理训练集比例: {rate}")  train\_data, train\_label, test\_data, test\_label = creat\_data()  C = 1 # 惩罚因子  toler = 0.001 # 迭代终止的边界值  maxIter = 100 # 增加最大迭代次数  svm\_model = SVM\_training(train\_data, train\_label, C, toler, maxIter)   # 计算准确率  accuracy = cal\_accuracy(svm\_model, test\_data, test\_label)  accuracies.append(accuracy)  print(f"准确率是: {accuracy \* 100:.3f}%")  # 绘制折线图 plt.figure(figsize=(10, 6)) plt.plot(rates, [acc \* 100 for acc in accuracies], marker='o') plt.title('SVM model accuracy') plt.xlabel('train rate') plt.ylabel('accuracy (%)') plt.xticks(rates) plt.grid() plt.show()**  同样是经过100次迭代，查看不同训练集下，模型准确率的情况    训练集合适时准确率最高 | | | |
| **5. 实验结果与分析**  支持向量机（SVM）分类器在鸢尾花数据集上也展现出了卓越的性能，其准确率同样令人印象深刻，并且具备坚实的理论基础和强大的分类能力。SVM通过寻找最优超平面来最大化类别间的间隔，从而实现了高效的分类。在鸢尾花数据集中，花萼和花瓣的长度、宽度等特征作为输入变量，在SVM的训练过程中起到了至关重要的作用。这些特征在数据集中的不同类别间存在明显的差异，使得SVM分类器能够准确地利用它们来进行分类。  然而，SVM的性能也受到一些关键因素的影响。例如，惩罚参数C的选择对于模型的复杂度和分类精度有着显著的影响。过大的C值可能导致模型过于复杂，增加过拟合的风险；而过小的C值则可能使模型过于简单，影响分类的准确性。此外，核函数的选择也是SVM性能的一个关键因素。不同的核函数能够将输入数据映射到不同的特征空间，从而影响模型的分类效果。  另外，数据集的预处理和特征选择同样对SVM的性能有重要影响。在鸢尾花数据集中，虽然特征数量较少且相对直观，但在实际应用中，特征的选择和预处理往往是提升模型性能的关键步骤。例如，通过标准化或归一化处理，可以消除不同特征间的量纲差异，使模型更加稳健。  综上所述，虽然SVM在鸢尾花数据集上展现出了出色的性能，但其性能的优化仍然需要综合考虑多个因素，包括惩罚参数的选择、核函数的确定以及数据的预处理和特征选择等。通过合理的参数调整和模型优化，可以进一步提升SVM的分类精度和泛化能力。**6. 结论**  通过本次实验，我对支持向量机算法有了更深入的理解，特别是在如何通过优化拉格朗日乘子来寻找最佳超平面方面。手动实现算法的过程让我更加直观地认识到SVM的核心思想和实现细节。同时，实验结果表明，SVM在鸢尾花数据集上的分类效果良好，具备实际应用的潜力。  总结来说，支持向量机作为一种经典的机器学习算法，凭借其优秀的分类性能和理论基础，值得进一步研究和应用。未来的工作可以考虑在更复杂的数据集上测试模型，或探索其他核函数对分类性能的影响。 | | | |

1. 当次实验结束后一周内按班组织上交实验报告。

2. 实验报告文件命名为：“学号+姓名”，格式为WORD文档。