机器学习实验报告

实验名称：实现支持向量机

学生：谢兴

学号：58122204

日期：2024/5/28

指导老师：刘胥影

助教：田秋雨

目录

1 任务描述

通过两种方式实现 SVM：

1. 手动实现 SMO 算法，并与直接使用传统二次规划方法进行对比。

2. 通过 scikit-learn 库实现软间隔 SVM。

并在 breast cancer 数据集上进行验证与实验。该数据集是一个二分类问题，属性均为连续属性，并已进行标准化。

2 问题复述

2.1 手动实现 SMO 算法

SVM 的对偶问题实际是一个二次规划问题，除了 SMO 算法外，传统二次规划方法也可以用于求解对偶问题。求得最优拉格朗日乘子后，超平面参数 w, b 可由以下式子得到：

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i x_i \tag{1}$$

$$b = \frac{1}{|S|} \sum_{s \in S} (\frac{1}{y_s} - \sum_{s \in S} \alpha_i y_i \mathbf{x_i^T x_s}) \tag{2}$$

请完成以下任务：

1. 不考虑软间隔情况，直接使用传统二次规划（QP）方法求解（实现）训练集上的硬间隔 SVM 对偶问题。观察并回答，这样的求解方法是否会出现问题，为什么？

2. 不限定硬间隔或是软间隔 SVM，也不限定是否为核化 SVM，根据需要选择合适的方法，手动实现 SMO 算法求解 SVM 对偶问题。注意第 3 步，KKT 条件验证步骤不能缺少。

3. 对测试数据进行预测，确保预测结果尽可能准确。

2.2 使用 **sklearn** 库简洁实现软间隔 **SVM**

2.2.1 使用 **sklearn** 库简洁实现软间隔 **SVM**。首先实现以下 **4** 个示例性的 **SVM** 模型：

1. 线性 SVM：正则化常数 C=1，核函数为线性核，

2. 线性 SVM：正则化常数 C=1000，核函数为线性核，

3. 非线性 SVM：正则化常数 C=1，核函数为多项式核，d=2，

4. 非线性 SVM：正则化常数 C=1000，核函数为多项式核，d=2，

观察并比较它们在测试集上的性能表现。

2.2.2 参数选择与参数分析

参数的选择对 SVM 的性能有很大的影响。确定正则化常数 C 与核函数及其参数的选择范围（可以在以下常用核函数中选择一种或多种），选用合适的实验评估方法（回顾第 2 章的内容，如 K 折交叉验证法等）进行参数选择，并进行参数分析实验。

1. 正则化常数 C 的选择范围可以是以下数量级，如：$\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3\}$

2. 核函数

   - 线性核：$k(x, y) = x^T y$，无参数，退化为 SVM 基本型
   - 多项式核：$k(x, y) = (x^T y + c)^d$
     - c：偏移量，通常取值 0 或 1

  - d：多项式的次数，通常取值 2, 3, 4 等较小的整数，取值为 1 时退化为线性核

- 多项式核另外一种常用的形式：$k(x, y) = (\gamma \cdot x^T y + c)^d$
  - γ：用于控制核函数的影响范围，常用值为 $1/\text{n\_features}$，n_features 是属性个数，或者也可以尝试 0.001, 0.01, 0.1, 1, 10 等
  - c：偏移量，通常取值 0 或 1
  - d：多项式的次数，通常取值 2, 3, 4 等较小的整数，取值为 1 时退化为线性核

- 高斯核/RBF 核：$k(x, y) = \exp(-\gamma \|x - y\|^2)$
  - γ：带宽参数，控制核函数的平滑程度。常用取值范围为 $\{10^{-3} - 10^3\}$。典型值为 $1/(\text{n\_features} \cdot X.\text{var}())$，其中 $X.\text{var}()$ 表示数据集 X 的方差，这可以确保 γ 的值在数据特征的数量和数据的方差范围内适当缩放。

- Laplace 核：$k(x, y) = \exp(-\gamma \|x - y\|)$
  - γ：带宽参数，控制核函数的平滑程度。常用取值范围参考高斯核的 γ 值

- Sigmoid 核函数：$k(x, y) = \tanh(\gamma \cdot x^T y + c)$
  - γ：缩放因子，常用取值范围为 $\{10^{-3} - 10^3\}$
  - c：偏移量，通常取值 0 或 1

请注意，SVM 中的参数通常需要联合选择，特别是对于核函数的参数与正则化参数 C。参数联合优化的策略有：

1. 网格搜索：网格搜索是一种穷举搜索法，在给定的参数空间中逐一尝试每种可能的参数组合，并选用合适的评估方法进行评估。

2. 随机搜索：在参数空间中对参数组合的随机采样进行评估。通常可以在较大的参数空间中找到接近最优的参数组合。步骤如下：

- 定义参数的取值范围
- 随机采样若干参数组合
- 对每个采样的参数组合进行评估
- 选择性能最优的参数组合

3. 贝叶斯优化：贝叶斯优化是一种更为高级的优化方法，通过构建一个模型来估计参数空间的性能分布，并通过最大化预期改进（Expected Improvement, EI）等准则来选择下一组参数进行评估。其步骤如下：

   - 初始采样若干参数组合并进行评估
   - 构建模型（如高斯过程回归）
   - 基于模型选择下一组参数进行评估
   - 更新模型并重复第 3 个步骤，直到达到预定的评估次数或满足停止条件。

本实验中可以选择较为简单的网格搜索策略或随机搜索策略进行调参。

参数分析实验需要报告：

1. 评估方法
2. 参数取值范围
3. 搜索策略
4. 比较分析每组参数上的性能情况
5. 最终选择的最优参数

3 实验环境

实验环境见表 1。

表 1: Experiment Environment

Items	Version
CPU	Intel Core i5-1135G7
RAM	16 GB
Python	3.11.5
scikit-learn	1.3.2
Operating system	Windows11

4 实验过程

4.1 手动实现 SMO 算法

4.1.1 传统的 QP 方法求解 SVM 对偶问题

实验结果显示，传统 QP 算法并不能直接求解 SVM 的对偶问题。这是因为 SVM 的对偶问题是一个二次规划问题，需要满足一定的条件才能使用 QP 算法进行求解。具体来说，SVM 的对偶问题需要满足以下条件：

4.1.2 手动实现 SMO 算法求解 SVM 对偶问题

在本实验中，我们手动实现了 SMO 算法来求解支持向量机（SVM）的对偶问题。代码实现的大致思路如下：

1. 定义核函数：我们使用线性核函数来计算样本之间的内积。
2. 定义求解 QP 问题的函数：使用 cvxopt 库解决二次规划问题，包括创建 QP 问题的矩阵并求解。
3. 加载训练数据并求解硬间隔 SVM：从 CSV 文件加载训练数据，调用上述函数求解 SVM 问题，得到模型参数（权重 w 和偏置 b）。

4. 定义预测函数：根据训练好的模型参数，对测试数据进行预测。

通过这些步骤，我们成功实现了 SMO 算法，并求解了 SVM 的对偶问题，得到了模型的参数（权重 w 和偏置 b）。

4.1.3 对测试数据进行预测

在完成模型训练后，我们使用训练得到的模型对测试数据进行了预测。

测试集准确率达到了 98.25%。在对比测试数据中的预测值和原始标签时，我们发现有 2 处不同。具体不同之处如下表所示：

样本索引	原始标签	预测标签
20	1	-1
77	1	-1

表 2: 原始标签与预测标签的不同之处

通过这些实验，我们验证了手动实现的 SMO 算法在解决 SVM 对偶问题上的有效性，并展示了其在实际数据集上的应用效果。

4.2 使用 sklearn 库简洁实现软间隔 SVM

4.2.1 实现 4 个示范性的 SVM 模型

在本实验中，我们使用了 sklearn 库来实现 4 个示范性的 SVM 模型，并在 breast cancer 数据集上进行训练和测试。这些模型包括：

1. 线性 SVM：正则化常数 C=1，核函数为线性核

2. 线性 SVM：正则化常数 C=1000，核函数为线性核

3. 非线性 SVM：正则化常数 C=1，核函数为多项式核，d=2

4. 非线性 SVM：正则化常数 C=1000，核函数为多项式核，d=2

以下是各个模型在测试集上的性能表现：

- 线性 **SVM (C=1)**:
  - 准确率：0.9825
- 线性 **SVM (C=1000)**:
  - 准确率：0.9561
- 非线性 **SVM (C=1, d=2)**:
  - 准确率：0.9825
- 非线性 **SVM (C=1000, d=2)**:
  - 准确率：0.9386

从实验结果可以看出：

- 线性 SVM 在 C=1 时的表现优于 C=1000 时的表现，表明适当的正则化有助于提高模型的泛化能力。
- 非线性 SVM 在 C=1 时的表现优于 C=1000 时的表现，这与线性 SVM 的情况相似，说明过大的正则化常数可能导致模型过拟合。
- 多项式核 SVM（d=2）的表现与线性 SVM 相当，表明在此数据集上，线性模型已经能够很好地分类数据，复杂的非线性核函数并未显著提升性能。

4.2.2 参数分析实验

在本实验中，我们进行了参数选择与参数分析。我们选择的参数范围如下：

- 正则化常数 C 的选择范围：$\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3\}$

- 核函数：
  - 线性核：$k(x, y) = x^T y$
  - 多项式核：$k(x, y) = (x^T y + c)^d$，其中 c 通常取值 0 或 1，d 通常取值 2, 3, 4 等较小的整数
  - 高斯核/RBF 核：$k(x, y) = \exp(-\gamma \|x - y\|^2)$，$\gamma$ 的常用取值范围为 $\{10^{-3} \square 10^3\}$
  - Laplace 核：$k(x, y) = \exp(-\gamma \|x - y\|)$，$\gamma$ 的常用取值范围与高斯核相同
  - Sigmoid 核函数：$k(x, y) = \tanh(\gamma x^T y + c)$，$\gamma$ 的常用取值范围为 $\{10^{-3} \square 10^3\}$，c 通常取值 0 或 1
- 核函数参数：
  - 多项式核：度数 d 的选择范围：$\{2, 3, 4\}$
  - 高斯核、Laplace 核、Sigmoid 核：γ 的选择范围：$\{0.001, 0.01, 0.1, 1, 10, 100\}$

我们使用网格搜索（GridSearchCV）方法对以上参数进行了交叉验证（5 折交叉验证），最终选择了最佳的参数组合。

- 最佳参数：

$$'C' : 0.001, 'degree' : 2, 'gamma' : 0.001, 'kernel' : 'linear', 'laplacian\_gamma' : 0.001$$

- 最佳交叉验证准确率：0.9780

从实验结果可以看出，线性核函数的表现最佳，同时适当的小的正则化常数 C（如 0.001）可以提高模型的泛化能力。高斯核、Laplace 核和 Sigmoid 核在本实验中未能显著提升性能，表明在此数据集上，线性核已经能够很好地分类数据。

本实验中核函数和正则化参数的选择对模型的性能有显著影响。通过网格搜索和交叉验证，我们能够有效地选择出最优参数组合，以提高模型的预测准确率。

4.2.3 对测试数据进行预测

在完成模型训练和参数选择后，我们使用最佳模型对测试数据进行了预测。具体的实验步骤和结果如下：

1. 首先，使用最佳参数配置对测试集进行了预测，得到了测试集的准确率为 0.9825。
2. 预测结果保存在 `result.csv` 文件中，文件中只包含一列数据标签分类，分别为 +1 或-1。
3. 对比了测试数据中的预测值和原始标签，结果显示有 2 处不同。

测试集准确率达到了 98.25%。在对比测试数据中的预测值和原始标签时，我们发现有 2 处不同。具体不同之处如下表所示：

样本索引	原始标签	预测标签
20	1	-1
77	1	-1

表 3: 原始标签与预测标签的不同之处

从以上结果可以看出，模型的整体准确率已经达到了较高的水平。

5 实验总结

本实验我们用三种方法实现了对 SVM 对偶形式的求解，包括传统的 QP 方法、手动实现的 SMO 算法和使用 sklearn 库的软间隔 SVM。

通过这些实验，我们验证了手动实现的 SMO 算法在解决 SVM 对偶问题上的有效性，并展示了其在实际数据集上的应用效果。同时，我们还通过 sklearn 库实现了软间隔 SVM，并进行了参数选择和分析实验，得到了最佳的参数组合。最终，我们对测试数据进行了预测，得到了较高的准确率。这些实验结果表明，SVM 在二分类问题上具有较好的性能，可以有效地应用于实际问题中。

A 手动实现 SVM 的 SMO 算法

A.1 传统 QP 算法

```
import numpy as np
import pandas as pd
from cvxopt import matrix, solvers

def kernel(x1, x2):
    return np.dot(x1, x2.T)

def svm_qp(X, y):
    m, n = X.shape
    y = y.astype(float)

    # 创建QP问题的P, q, G, h, A, b矩阵
    K = np.dot(X, X.T) * np.outer(y, y)
    P = matrix(K)
    q = matrix(-np.ones((m, 1)))
    G = matrix(-np.eye(m))
    h = matrix(np.zeros(m))
    A = matrix(y.reshape(1, -1).astype(float))
    b = matrix(np.zeros(1))

    # 求解QP问题
    sol = solvers.qp(P, q, G, h, A, b)
    alphas = np.array(sol['x']).flatten()

    # 计算权重向量w
    w = np.sum(alphas * y[:, None] * X, axis=0)

    # 计算偏置b
    support_vectors = (alphas > 1e-5)
    b = np.mean(y[support_vectors] - np.dot(X[support_vectors], w))

    return w, b, alphas

# 加载数据
X_train = np.loadtxt('breast_cancer_Xtrain.csv', delimiter=',')
y_train = np.loadtxt('breast_cancer_Ytrain.csv', delimiter=',')

```

```
# 使用训练数据集进行QP求解
w, b, alphas = svm_qp(X_train, y_train)
print("Weights:", w)
print("Bias:", b)

def predict(X, w, b):
    return np.sign(np.dot(X, w) + b)

# 加载测试数据
X_test = np.loadtxt('breast_cancer_Xtest.csv', delimiter=',')
y_test = np.loadtxt('breast_cancer_Ytest.csv', delimiter=',')

# 对测试集进行预测
predictions = predict(X_test, w, b)

# 计算准确率
accuracy = np.mean(predictions == y_test)
print("Accuracy on test data:", accuracy)
```

A.2 手动实现 SMO

```
import numpy as np

def kernel(x1, x2):
    return np.dot(x1, x2.T)

def calculate_b(X, y, alphas, b, C, tol):
    m = len(y)
    b_new = 0
    b1 = []
    b2 = []

    for i in range(m):
        y_pred = np.sum(alphas * y * kernel(X, X[i])) + b
        if y[i] * y_pred - 1 < -tol:
            b1.append(b + y[i] - y_pred)
        elif y[i] * y_pred - 1 > tol:
            b2.append(b + y[i] - y_pred)

    if len(b1) > 0:
        b_new = np.mean(b1)
```

```
    elif len(b2) > 0:
        b_new = np.mean(b2)

    return b_new

def smo_svm(X, y, C, tol, max_passes):
    m, n = X.shape
    alphas = np.zeros(m)
    b = 0
    passes = 0

    while passes < max_passes:
        alpha_pairs_changed = 0
        for i in range(m):
            E_i = np.sum(alphas * y * kernel(X, X[i])) + b - y[i]
            if (y[i] * E_i < -tol and alphas[i] < C) or (y[i] * E_i > tol
 and alphas[i] > 0):
                j = np.random.randint(0, m)
                while j == i:
                    j = np.random.randint(0, m)

                E_j = np.sum(alphas * y * kernel(X, X[j])) + b - y[j]
                alpha_i_old, alpha_j_old = alphas[i], alphas[j]

                if y[i] != y[j]:
                    L = max(0, alphas[j] - alphas[i])
                    H = min(C, C + alphas[j] - alphas[i])
                else:
                    L = max(0, alphas[i] + alphas[j] - C)
                    H = min(C, alphas[i] + alphas[j])

                if L == H:
                    continue

                eta = 2 * kernel(X[i], X[j]) - kernel(X[i], X[i]) - kernel(X
[j], X[j])
                if eta >= 0:
                    continue

                alphas[j] -= y[j] * (E_i - E_j) / eta
                alphas[j] = np.clip(alphas[j], L, H)
```

```
                if abs(alphas[j] - alpha_j_old) < 1e-5:
                    continue

                alphas[i] += y[i] * y[j] * (alpha_j_old - alphas[j])
                b1 = b - E_i - y[i] * (alphas[i] - alpha_i_old) * kernel(X[i], X[i]) - y[j] * (alphas[j] - alpha_j_old) * kernel(X[i], X[j])
                b2 = b - E_j - y[i] * (alphas[i] - alpha_i_old) * kernel(X[i], X[j]) - y[j] * (alphas[j] - alpha_j_old) * kernel(X[j], X[j])
                if 0 < alphas[i] < C:
                    b = b1
                elif 0 < alphas[j] < C:
                    b = b2
                else:
                    b = (b1 + b2) / 2

                alpha_pairs_changed += 1

        if alpha_pairs_changed == 0:
            passes += 1
        else:
            passes = 0

    return alphas, b

# Load the data
X_train = np.loadtxt('breast_cancer_Xtrain.csv', delimiter=',')
y_train = np.loadtxt('breast_cancer_Ytrain.csv', delimiter=',')

C = 1.0
tol = 1e-3
max_passes = 5

alphas, b = smo_svm(X_train, y_train, C, tol, max_passes)
print("Alphas:", alphas)
print("b:", b)


def predict(X, alphas, b, X_train, y_train):
return np.sign(np.dot((alphas * y_train).T, kernel(X_train, X)) + b)

```

```python
# Load test data
X_test = np.loadtxt('breast_cancer_Xtest.csv', delimiter=',')
y_test = np.loadtxt('breast_cancer_Ytest.csv', delimiter=',')

predictions = predict(X_test, alphas, b, X_train, y_train)

accuracy = np.mean(predictions == y_test)
print("Accuracy on test data:", accuracy)



# 预测函数
def predict(X, alphas, b, X_train, y_train):
    return np.sign(np.dot((alphas * y_train).T, kernel(X_train, X)) + b)

# 加载测试数据
X_test = np.loadtxt('breast_cancer_Xtest.csv', delimiter=',')
y_test = np.loadtxt('breast_cancer_Ytest.csv', delimiter=',')
y_test[y_test == 0] = -1  # 转换标签为 -1 和 1

# 对测试集进行预测
predictions = predict(X_test, alphas, b, X_train, y_train)

# 计算准确率
accuracy = np.mean(predictions == y_test)
print("Accuracy on test data:", accuracy)

# 将预测结果写入 result.csv 文件
np.savetxt('result.csv', predictions, delimiter=',', fmt='%d')

# 加载原始标签和预测标签
original_labels = y_test
predicted_labels = predictions

# 创建 DataFrame 来存储比较结果
comparison_df = pd.DataFrame({
    'Original': original_labels,
    'Predicted': predicted_labels
})

# 找出不同的标签
```

```
differences = comparison_df[comparison_df['Original'] != comparison_df['
    Predicted']]

# 输出不同标签的行数
print(f"Number of differences: {len(differences)}")

# 输出所有不同的标签
print("Differences between original and predicted labels:")
print(differences)
```

B 使用 sklearn 库简洁实现软间隔 SVM

```
# 1. 导入必要的库和加载数据集
import numpy as np
import pandas as pd
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV

X_train = pd.read_csv('breast_cancer_Xtrain.csv', header=None)
X_test = pd.read_csv('breast_cancer_Xtest.csv', header=None)
y_train = pd.read_csv('breast_cancer_Ytrain.csv', header=None)
y_test = pd.read_csv('breast_cancer_Ytest.csv', header=None)

y_train = y_train.values.ravel()
y_test = y_test.values.ravel()

#2、实现四个示例性的SVM模型
# 定义和训练SVM模型
models = {
    "Linear SVM (C=1)": SVC(kernel='linear', C=1),
    "Linear SVM (C=1000)": SVC(kernel='linear', C=1000),
    "Polynomial SVM (C=1, d=2)": SVC(kernel='poly', C=1, degree=2),
    "Polynomial SVM (C=1000, d=2)": SVC(kernel='poly', C=1000, degree=2)
}

# 训练并评估模型
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
```

```
    accuracy = accuracy_score(y_test, y_pred)
    print(f"{name} Accuracy: {accuracy:.4f}")


# 3. 定义拉普拉斯核函数
def laplacian_kernel(X, Y, gamma):
    K = np.exp(-gamma * np.linalg.norm(X[:, np.newaxis] - Y[np.newaxis, :], axis=2))
    return K

# 4. 定义自定义SVC类
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.utils import check_X_y, check_array
from sklearn.utils.multiclass import unique_labels

class CustomSVC(BaseEstimator, ClassifierMixin):
    def __init__(self, C=1.0, kernel='linear', degree=3, gamma='scale', coef0=0.0, laplacian_gamma=1.0):
        self.C = C
        self.kernel = kernel
        self.degree = degree
        self.gamma = gamma
        self.coef0 = coef0
        self.laplacian_gamma = laplacian_gamma
        self.svc = SVC(C=self.C, kernel=self.kernel, degree=self.degree, gamma=self.gamma, coef0=self.coef0)

    def fit(self, X, y):
        X, y = check_X_y(X, y)
        if self.kernel == 'laplacian':
            self.svc.kernel = lambda X, Y: laplacian_kernel(X, Y, self.laplacian_gamma)
        self.svc.fit(X, y)
        self.classes_ = unique_labels(y)
        return self

    def predict(self, X):
        X = check_array(X)
        return self.svc.predict(X)


```

```
66
67  # 5. 设置参数网格并进行网格搜索
68  param_grid = {
69      'C': [10**-3, 10**-2, 10**-1, 1, 10, 10**2, 10**3],
70      'kernel': ['linear', 'poly', 'rbf', 'sigmoid', 'laplacian'],
71      'degree': [2, 3, 4],
72      'gamma': [0.001, 0.01, 0.1, 1, 10, 100],
73      'laplacian_gamma': [0.001, 0.01, 0.1, 1, 10, 100]
74  }
75
76  grid_search = GridSearchCV(CustomSVC(), param_grid, cv=5, scoring='accuracy'
     )
77  grid_search.fit(X_train, y_train)
78
79  best_params = grid_search.best_params_
80  best_score = grid_search.best_score_
81  print(f"Best Parameters: {best_params}")
82  print(f"Best Cross-Validation Accuracy: {best_score:.4f}")
83
84  # 6. 在测试集上评估最优模型
85  best_model = grid_search.best_estimator_
86  y_pred_best = best_model.predict(X_test)
87  best_test_accuracy = accuracy_score(y_test, y_pred_best)
88  print(f"Best Model Test Accuracy: {best_test_accuracy:.4f}")
89
90  # 7. 对测试集中所有数据进行预测并输出结果
91  y_pred_all = best_model.predict(X_test)
92
93  # 计算所有测试数据的准确率
94  test_accuracy = accuracy_score(y_test, y_pred_all)
95  print(f"Test Accuracy: {test_accuracy:.4f}")
96
97  # 将预测结果转换为 DataFrame
98  result_df = pd.DataFrame(y_pred_all)
99
100 # 将结果保存到 result.csv 文件中，没有表头，只有一列数据标签分类
101 result_df.to_csv('result.csv', index=False, header=False)
102
103 # 8、找出不同的标签
104 import pandas as pd
105
```

```python
# 加载原始标签和预测标签
original_labels = pd.read_csv('breast_cancer_Ytest.csv', header=None).values.ravel()
predicted_labels = pd.read_csv('result.csv', header=None).values.ravel()

# 创建 DataFrame 来存储比较结果
comparison_df = pd.DataFrame({
    'Original': original_labels,
    'Predicted': predicted_labels
})

# 找出不同的标签
differences = comparison_df[comparison_df['Original'] != comparison_df['Predicted']]

# 输出不同标签的行数
print(f"Number of differences: {len(differences)}")

# 输出所有不同的标签
print("Differences between original and predicted labels:")
print(differences)
```