国家卓越工程师学院明月科创实验班

《概率论与数理统计》课程项目 1

姓　名：王子诺

学　号：20235459

年　级：2023 级

班　级：1 班

日　期：2025 年 6 月 25 日

重庆大学国家卓越工程师学院 2024 年制

目录

概率论与数理统计项目二报告

一、项目分析

本次项目的主要目标是实现一个基于 SVM（支持向量机）的分类问题，并将所得结果与 Logistic Regression 进行详细的数据对比分析。在构建 SVM 模型的过程中，我们需要分别实现硬间隔 SVM 和软间隔 SVM 两种不同的模型。具体来说，硬间隔 SVM 侧重于寻找一个严格的分类边界，而软间隔 SVM 则允许一定程度的误分类，以更好地处理实际数据中的噪声和异常点。

在数据准备阶段，我将采取多种途径进行数据收集，既包括从实际应用场景中获取的真实数据，也包括通过特定算法生成的模拟数据。为了确保模型的训练效果和评估准确性，数据集必须明确划分为训练集和测试集（或预测集）。每类样本的数量不得少于 250 个，这样的样本量不仅能够提供足够的数据支撑，还能确保数据集的高质量，从而为算法性能的提升奠定坚实基础。此外，我们还将按照一定比例将数据集进一步划分为训练集和验证集，以有效提升模型的泛化能力，确保其在未知数据上的表现。

在数据库建立过程中，对数据指标的选取尤为关键。我们将通过一系列科学的方法对数据指标进行细致的筛选，力求找到相关性最为合适的两个指标，以便更准确地实现分类任务。这一步骤不仅关系到模型的训练效果，更是直接影响最终分类结果的关键环节。

在实现硬间隔 SVM 建模时，我们选择使用线性决策边界。然而，这种看似简单直接的方法在实际应用中存在一定的局限性，因为它仅选取部分数据点作为支持向量，可能导致模型对异常点过于敏感，进而影响最终的分类结果。与之相比，Logistic Regression 作为一种经典的分类算法，成为了我们自然选择的对比基准，能够更全面地考察线性决策边界的适用性和有效性，为我们提供更为可靠的对比依据。

二、 数据寻找和预处理

2.1 数据库来源

为了寻找合适的 SVM 数据库，我在 kaggle 上找到了一个关于两类葡萄干的数据集 raisin_dataset，其中有 900 个关于两种土耳其葡萄干（Kecimen&Besni）的多个特征的数据集，包含了面积、主 X 轴长度，交叉面积、含量等等信息。

（葡萄干数据集）

这个数据集的来源是从两个品种中获得了 900 粒葡萄干的图像。这些图像经过了各种预处理步骤，并使用图像处理技术进行了 7 次形态特征提取操作。此外，还计算了每个特征的最小值、平均值、最大值和标准偏差统计信息。研究了两种葡萄干品种在特征上的分布，并将这些分布显示在图表上。

Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	Extent	Perimeter	Class
87524	442.2460114	253.291155	0.819738392	90546	0.758650579	1184.04	Kecimen
75166	406.690687	243.0324363	0.801805234	78789	0.68412957	1121.786	Kecimen
90856	442.2670483	266.3283177	0.798353619	93717	0.637612812	1208.575	Kecimen
45928	286.5405586	208.7600423	0.684989217	47336	0.699599385	844.162	Kecimen
79408	352.1907699	290.8275329	0.56401133	81463	0.792771926	1073.251	Kecimen
49242	318.125407	200.12212	0.777351277	51368	0.658456354	881.836	Kecimen
42492	310.1460715	176.1314494	0.823098681	43904	0.665893562	823.796	Kecimen
60952	332.4554716	235.429835	0.706057518	62329	0.74359819	933.366	Kecimen
42256	323.1896072	172.5759261	0.845498789	44743	0.698030924	849.728	Kecimen
64380	366.9648423	227.7716147	0.784055626	66125	0.664375716	981.544	Kecimen
80437	449.4545811	232.3255064	0.856042518	84460	0.674235757	1176.305	Kecimen
43725	301.3222176	186.9506295	0.784258452	45021	0.697068248	818.873	Kecimen
43441	276.6108288	201.8131355	0.683882337	45133	0.690855598	803.748	Kecimen
76792	338.8575454	291.3592017	0.510583813	78842	0.772322237	1042.77	Kecimen
74167	387.7989307	247.8581228	0.769089738	76807	0.680181585	1084.729	Kecimen
33565	261.5543311	167.7084908	0.767374275	35794	0.68155052	751.413	Kecimen
64670	403.0839752	206.4846437	0.858829168	66419	0.75677257	1028.445	Kecimen
64762	354.2939396	235.7524629	0.746473726	66713	0.694998015	981.509	Kecimen
43295	304.2844667	182.8110368	0.799406959	44714	0.713838189	814.68	Kecimen
70699	418.6985723	216.5960537	0.855799392	72363	0.728075054	1061.321	Kecimen
69726	354.1769124	252.529208	0.701160962	71849	0.734398534	1035.501	Kecimen
57346	330.4784385	222.4437485	0.739555027	59365	0.723608833	928.272	Kecimen
82028	397.1149759	268.3337727	0.737169367	84427	0.686375085	1106.355	Kecimen
61251	301.5077895	273.6599414	0.419753707	64732	0.643595671	971.769	Kecimen
96277	447.1345225	275.2161542	0.788128405	97865	0.704057157	1181.921	Kecimen
75620	368.2242844	263.4592554	0.698627251	77493	0.726277372	1059.186	Kecimen
73167	340.055218	276.0151772	0.58410581	74545	0.778736856	1010.474	Kecimen
60847	336.9238696	231.4656959	0.726660229	62492	0.69858783	964.603	Kecimen
81021	347.7500583	297.6406265	0.517134931	82552	0.757559607	1063.868	Kecimen

（数据集及特征）

而在其中，两种类型的葡萄干数据分别有 450 张，我对 excel 的表格中的数据进行了格式统一为 int64 类型，以方便后续的处理。

```
类别计数:
Class
Kecimen    450
Besni      450
Name: count, dtype: int64

唯一类别: ['Kecimen' 'Besni']
```

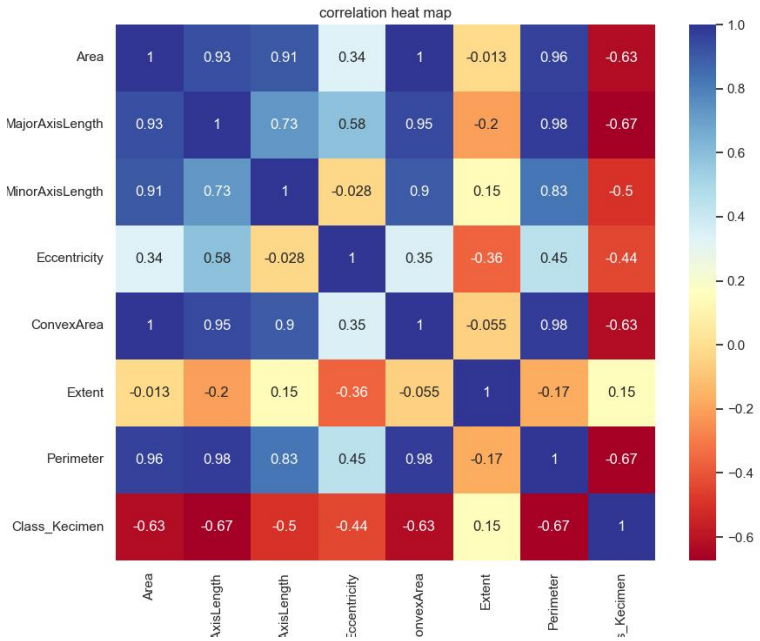
（两个类别数据的统计量）

2.2 数据库预处理

对于这个数据集中的很多数据特征，我需要找到最合适的数据特征来进进行模型的建立，因此，我定义一个名为 preprocess_data 的函数，对 DataFrame 中的 Class 列进行独热编码,将 Class_Kecimen 列提取为目标变量 target，使用相关性矩阵显示每对数值特征之间的相关性，值范围在 [-1, 1] 之间。最后再用相关性热图来可视化展示这些数据特征的相关性，选择相关性大于 0.6 的前两个特征参数来作为模型搭建的参数，最终选出来得到 Area 和 MajorAxisLength 这两个指标。

```python
def preprocess_data(df):
    """预处理数据：编码目标变量，选择特征，计算相关性"""
    # 编码分类变量
    df = pd.get_dummies(df, columns=['Class'], drop_first=True)
    target = df['Class_Kecimen']
    target = target.astype(int)
    df['Class_Kecimen'] = df['Class_Kecimen'].astype(int)

    # 计算相关性矩阵
    corr_matrix = df.select_dtypes('number').corr()

    # 可视化相关性热图
    plt.figure(figsize=(10, 8))
    sns.heatmap(corr_matrix, cmap='RdYlBu', annot=True)
    plt.title('correlation heat map')
    plt.show()

    # 选择相关性绝对值>=0.6的特征
    selected_features = corr_matrix.index[abs(corr_matrix['Class_Kecimen']) >= 0.6].tolist()
    selected_features.remove('Class_Kecimen')
    if len(selected_features) > 2:
        selected_features = selected_features[:2]

    return df, target, selected_features, corr_matrix
```

（数据处理部分代码）



（相关性热图）

2.3 数据预分类

在此之后，每次进行模型训练之前，我还要划分数据集和验证集，从 sklearn.model_selection 中导入 train_test_split 函数，引入随机数种子，按照 4:1 的比例去随机划分，从而使得模型能够具有泛化性。

2.4 开源数据链接

开源数据链接：

https://www.kaggle.com/datasets/muratkokludataset/raisin-dataset/data

三、 代码实现

3.1 Logistic Regression 方法实现

3.1.1 代码构架

```python
def train_logistic_model(X, y):
    """训练逻辑回归模型并返回预测结果"""
    # 划分数据集
    X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, test_size=0.2, random_state=42)

    # 特征标准化
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # 训练模型
    model = LogisticRegression()
    model.fit(X_train_scaled, y_train)

    # 预测并评估
    y_pred = model.predict(X_test_scaled)
    score = model.score(X_test_scaled, y_test)

    print("\n逻辑回归模型系数：", model.coef_)
    print("测试集预测结果：", y_pred)
    print("测试集准确率：", score)

    return model, X_train_scaled, X_test_scaled, y_train, y_test, y_pred
```

（Logistic 回归代码）

我在使用 logistic 回归方法进行分类时，首先划分为训练集和测试集。随后，为了提高模型的训练效率和预测准确性，必须对训练集的特征进行标准化处理。标准化过程包括计算每个特征的均值和标准差，并将所有特征值转换为均值为 0、标准差为 1 的标准化数据。这一步骤不仅有助于消除不同特征量纲的影响，还能使模型更容易收敛。标准化后的训练集和测试集特征分别存储在 X_train_scaled 和 X_test_scaled 变量中，以便后续使用。

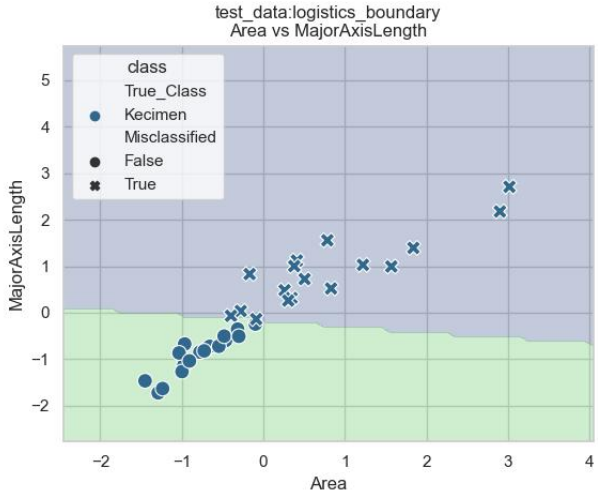
为了全面评估模型的性能，需要利用训练好的模型对测试集进行预测，并将预测结果存储在 y_pred 变量中。此外，通过计算模型的 score（通常指准确率或其他评估指标），可以量化模型在测试集上的表现，从而对模型的泛化能力有一个直观的认识。这一系列的评估步骤不仅有助于判断模型的优劣，还能为进一步的模型优化提供重要参考。

3.1.2 可视化结果显示

```python
def visualize_decision_boundary(model, X_test, y_test, y_pred, selected_features):
    """可视化决策边界和测试数据点"""
    # 如果需要，将标准化数据转换为numpy数组
    X_test_scaled_np = X_test.to_numpy() if isinstance(X_test, pd.DataFrame) else X_test

    # 创建用于可视化的DataFrame
    df_test = pd.DataFrame(X_test, columns=selected_features)
    df_test["True_Class"] = pd.Series(y_test).map({0: "Besni", 1: "Kecimen"})
    df_test["Pred_Class"] = pd.Series(y_pred).map({0: "Besni", 1: "Kecimen"})
    df_test["Misclassified"] = df_test["True_Class"] != df_test["Pred_Class"]

    # 创建网格用于决策边界
    x_min, x_max = X_test_scaled_np[:, 0].min() - 1, X_test_scaled_np[:, 0].max() + 1
    y_min, y_max = X_test_scaled_np[:, 1].min() - 1, X_test_scaled_np[:, 1].max() + 1
    xx, yy = np.meshgrid( *xi: np.arange(x_min,  *args: x_max, 0.1), np.arange(y_min,  *args: y_max, 0.1))

    # 预测网格点
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = np.where(Z == 0,  x: 0,  y: 1)
    Z = Z.reshape(xx.shape)

    # 决策边界
    # plt.contourf(xx, yy, Z, alpha=0.3, cmap="viridis", levels=[-0.5, 0.5, 1.5])
    # plt.figure(figsize=(10, 8))
    plt.contourf( *args: xx, yy, Z, alpha=0.3, cmap="viridis", levels=[-0.5, 0.5, 1.5])
    sns.scatterplot(
        data=df_test,
        x=selected_features[0],
        y=selected_features[1],
        hue="True_Class",
        style="Misclassified",
        style_order=[False, True],
        markers={False: "o", True: "X"},
        palette=palette,
        s=100,
    )
    plt.title(f"test_data:logistics_boundary\n{selected_features[0]} vs {selected_features[1]}")
    plt.legend(title="class")
    plt.show()
```

（绘制决策边界）

该代码中我标记预测错误的样本（True 表示真实类别与预测类别不一致，False 表示预测正确）。然后使用网格的方法进行决策边界，从而得到特定的数据点是否为 kecimen（其实否也就是 Besni 类的）。正确分类的点用圆形（"o"），错误分类的点用叉（"X"）。



（Logistic_决策边界结果图）

3.1.3 模型评估

最终模型的测试结果为：

```
逻辑回归模型系数： [[-0.33113448 -2.74494566]]
测试集预测结果： [0 1 1 0 1 1 1 1 1 1 0 0 0 1 0 0 0 1 1 0 1 0 1 1 0 1 0 0 0 0 1 0 0 0 0 0 1
 0 1 1 1 1 1 1 0 0 1 0 0 0 0 1 0 1 1 1 1 0 1 0 0 1 0 1 1 1 0 0 1 1 0 0 1 1
 0 0 1 0 1 0 0 1 0 1 0 0 1 0 1 0 1 1 0 0 0 0 1 1 0 0 0 1 0 0 0 1 0 1 0 1 0
 0 0 1 0 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 0 1 0 1 0 0 0 1 1 1 0 1 0 1 0 1
 0 1 1 0 1 0 1 1 1 1 1 1 1 0 1 1 0 0 1 0 0 0 1 1 1 1 1 0 0 0 0 1 1]
测试集准确率： 0.8611111111111112
```

模型系数反映了每个特征对目标变量（即分类结果）的贡献程度。第一个特征的系数为-0.33113448，负值意味着当该特征值增加时，模型预测目标类别为 1（例如“Kecimen”）的概率会降低，更倾向于预测类别 0（例如“Besni”）。第二个特征的系数为-2.74494566，较大的负值显示出该特征对分类的影响更为显著，且增加该特征值将显著降低预测类别 1 的概率。

测试集准确率为 86.11%，表示模型在测试集上的预测正确率，具有较强的分类能力，即正确分类的样本占总样本的比例。也就是说，在数据集平衡的前提下，这个准确率表明模型能够较好地区分两类数据。

3.2 硬间隔 SVM 方法实现

3.2.1 代码构架

```python
def train_svm_model(X, y):
    """训练硬间隔SVM模型并返回预测结果"""
    # 划分数据集
    X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, test_size=0.2, random_state=42)

    # 特征标准化
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # 训练硬间隔SVM模型（C设为很大值以模拟硬间隔）
    try:
        model = LinearSVC(C=1e10, max_iter=10000)
        model.fit(X_train_scaled, y_train)
    except ValueError as e:
        print("错误：数据可能不是线性可分的，无法使用硬间隔SVM。", e)
        return None, None, None, None, None, None, None

    # 预测
    y_pred = model.predict(X_test_scaled)
    score = model.score(X_test_scaled, y_test)
    print("\n硬间隔SVM回归模型系数：", model.coef_)
    print("测试集预测结果：", y_pred)
    print("测试集准确率：", score)

    return model, X_test, X_test_scaled, X_train,X_train_scaled, y_train, y_test, y_pred, scaler
```

(硬间隔 SVM 代码实现）

这里写一个函数 train_svm_model 用于训练一个硬间隔支持向量机（SVM）模型并返回预测结果。也是使用 train_test_split 将输入特征 X 和目标变量 y 划分为训练集，然后通过 StandardScaler 对训练集和测试集特征进行标准化处理，以确保特征在相同尺度上。接着，函数尝试使用 LinearSVC（C=1e10，用一个非常大的值来模拟硬间隔 SVM）在标准化后的训练数据上训练模型，若数据线性不可分则捕获异常并返回空值。训练完成后，模型在测试集上进行预测，输出模型系数、预测结果和准确率。最后，函数返回训练好的模型、原始和标准化后的测试集与训练集特征、训练集和测试集目标变量、预测结果以及标准化器对象。此函数适用于二分类任务，特别强调硬间隔 SVM 的应用，但要求数据线性可分。

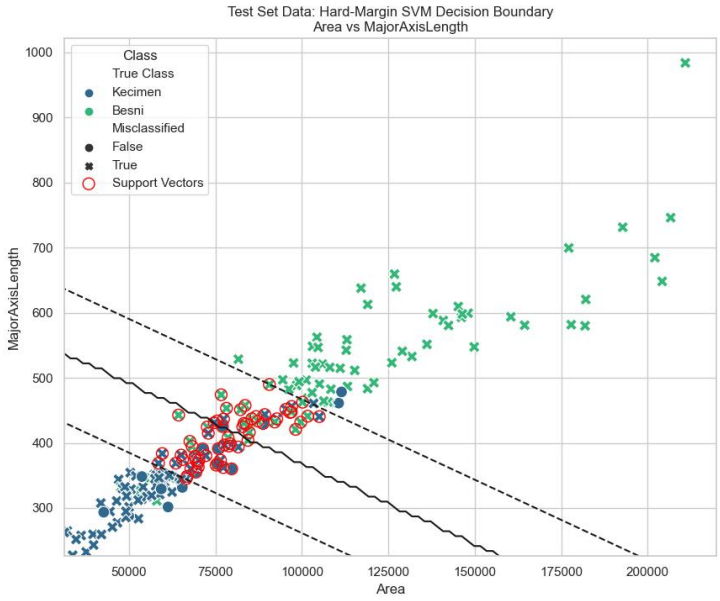
3.2.2 可视化结果显示

```python
def visualize_decision_boundary_for_svm(model, X_test, X_test_scaled, y_test, y_pred, selected_features, scaler):
    """Visualize the decision boundary (straight line) and support vectors for hard-margin SVM on the test set"""
    if model is None:
        print("Cannot visualize: Model training failed.")
        return

    # Create DataFrame for visualization (using unscaled features)
    df_test = pd.DataFrame(X_test, columns=selected_features)
    df_test["True Class"] = pd.Series(y_test).map({0: "Besni", 1: "Kecimen"})
    df_test["Predicted Class"] = pd.Series(y_pred).map({0: "Besni", 1: "Kecimen"})
    df_test["Misclassified"] = df_test["True Class"] != df_test["Predicted Class"]

    # Create grid for decision boundary (based on unscaled features)
    x_min, x_max = X_test[selected_features[0]].min() - 1, X_test[selected_features[0]].max() + 1
    y_min, y_max = X_test[selected_features[1]].min() - 1, X_test[selected_features[1]].max() + 1
    xx, yy = np.meshgrid( *xi: np.arange(x_min, *args: x_max, (x_max - x_min) / 100),
                         np.arange(y_min, *args: y_max, (y_max - y_min) / 100))

    # Scale grid points for prediction
    grid_points = np.c_[xx.ravel(), yy.ravel()]
    grid_points_scaled = scaler.transform(grid_points)
    Z = model.predict(grid_points_scaled)
    Z = Z.reshape(xx.shape)

    # Plot decision boundary and margins
    plt.figure(figsize=(10, 8))
    # Plot decision boundary (straight line)
    plt.contour( *args: xx, yy, Z, levels=[0], colors='k', linestyles=['-'])
    # Plot margin boundaries (w*x + b = ±1)
    Z_margin = model.decision_function(grid_points_scaled)
    Z_margin = Z_margin.reshape(xx.shape)
    plt.contour( *args: xx, yy, Z_margin, levels=[-1, 1], colors='k', linestyles=['--'])

    # Plot test set scatter plot
    sns.scatterplot(
        data=df_test,
        x=selected_features[0],
        y=selected_features[1],
        hue='True Class',
        style='Misclassified',
        style_order=[False, True],
        markers={False: 'o', True: 'X'},
        palette=palette,
        s=100
    )

    # Plot test set support vectors
    distances = model.decision_function(X_test_scaled)  # Modified here to decide whether to use test or training s
    support_vectors = X_test[np.abs(distances) <= 1 + 1e-6]
    plt.scatter(support_vectors[selected_features[0]],
```

(可视化硬间隔 SVM 代码）

此处使用函数 visualize_decision_boundary_for_svm 用于可视化硬间隔支持向量机（SVM）模型在测试集上的决策边界、支持向量和数据点。基于原始（未标准化）测试集特征 X_test，创建包含真实类别（"Besni" 或 "Kecimen"）、预测类别和误分类标记的 DataFrame。然后，生成一个基于两个特征(selected_features)的二维网格，标准化网格点后使用模型预测类别以绘制决策边界（直线）和间隔边界（w*x+b=±1）。再绘制测试集数据点，颜色表示真实类

别，形状区分正确和错误分类（圆形为正确，叉形为错误）。此外，函数识别测试集中的支持向量，以红色空心圆标记。最后，生成包含决策边界、间隔、数据点和支持向量的图表，直观展示 SVM 的分类效果和支持向量位置。



（硬间隔 SVM 分类结果）

3.2.3 模型评估分析

```
硬间隔SVM回归模型系数： [[-0.94369338 -1.12941151]]
测试集预测结果： [0 1 1 0 1 0 1 1 1 1 0 0 0 1 0 0 0 1 1 0 1 0 1 1 0 1 0 0 0 0 1 0 0 0 0 0 1
 0 1 1 1 1 1 1 0 0 1 0 0 0 0 1 0 1 1 1 1 0 1 0 0 1 0 1 1 1 0 0 1 1 0 0 1 1
 0 0 1 1 1 0 0 1 0 1 0 0 1 0 1 0 1 1 0 0 0 0 1 1 0 0 0 1 0 0 0 1 0 1 0 1 0
 0 0 1 0 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 0 1 0 1 0 0 0 1 1 1 0 1 0 1 0 1
 0 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 0 0 1 0 0 0 1 1 1 1 0 0 0 0 1 1]
测试集准确率： 0.8555555555555555
```

硬间隔支持向量机（SVM）模型在测试集上的准确率为 85.56%，表明其在二分类任务（中具有较好的分类性能，但略低于逻辑回归模型的 86.11%。模型系数为 [[-0.94369338 -1.12941151]]，显示两个特征对分类的贡献较为接近，第二个特征的影响略大于第一个，且负系数表明特征值增加时倾向于预测类别 0。相比逻辑回归，SVM 的准确率稍低，可能因其严格的线性分离要求未完全适应数据分布。

3.3 非线性软间隔 SVM（基于 RBF，来源论文二）

3.3.1 代码构架

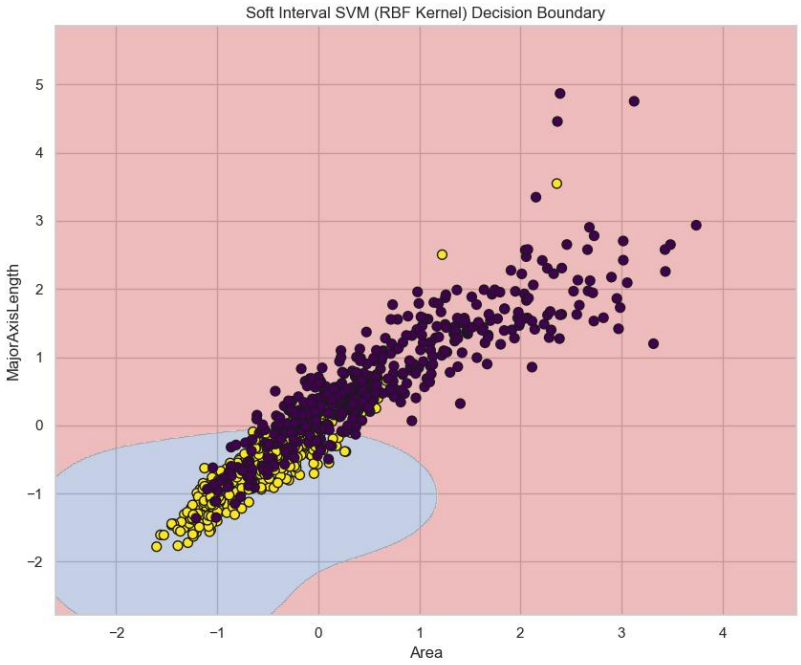
```
def train_soft_margin_svm(X, y, C=1.0, gamma="scale", test_size=0.2, random_state=42):

    # 划分训练集和测试集
    X_train, X_test, y_train, y_test = train_test_split(
        *arrays: X, y, test_size=test_size, random_state=random_state
    )

    # 标准化特征（SVM 对特征尺度敏感）
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # 训练软间隔 SVM 模型（使用 RBF 核）
    svm_model = SVC(kernel="rbf", C=C, gamma=gamma, random_state=random_state)
    svm_model.fit(X_train_scaled, y_train)

    # 预测并评估模型
    y_pred = svm_model.predict(X_test_scaled)
    accuracy = accuracy_score(y_test, y_pred)
    print("\n软间隔 SVM（RBF 核）结果：")
    print(f"测试集准确率：{accuracy:.4f}")
    print("\n分类报告：")
    print(classification_report(y_test, y_pred))
    print("\n混淆矩阵：")
    print(confusion_matrix(y_test, y_pred))

    return svm_model, scaler, accuracy
```

（基于 RBF 的软间隔 SVM 代码实现）

函数 train_soft_margin_svm 实现了一个基于径向基函数（RBF）核的软间隔支持向量机（SVM）分类器，用于处理非线性可分数据。首先通过 train_test_split 将特征矩阵 X 和目标变量 y 划分为训练集和测试集。随后标准化特征，确保特征尺度一致，因为 SVM 对尺度敏感。接着，函数初始化一个 SVC 模型，采用 RBF 核，通过参数 C（默认 1.0）控制误分类惩罚的强度，gamma（默认 "scale"）调节核函数的宽度，从而实现软间隔 SVM 的非线性分类。模型在标准化后的训练集上拟合后，对测试集进行预测，并输出准确率、分类报告（包含精确率、召回率、F1 分数）和混淆矩阵以全面评估性能。函数返回训练好的模型、标准化器和测试集准确率。这种方式通过 RBF 核将数据映射到高维空间，结合软间隔机制（允许一定误分类）实现灵活的非线性分类。

3.3.2 可视化结果显示

我使用了一个函数 plot_decision_boundary 来进行可视化基于非线性支持向量机（SVM）模型（如使用 RBF 核的软间隔 SVM）的二维特征空间决策边界。它首先检查输入特征矩阵 X 是否包含正好两个特征，接着，使用提供的 scaler 对特征数据 X 进行标准化。再基于标准化后的特征范围，生成一个步长 0.01 的二维网格，并对网格点进行模型预测，生成决策边界区域 Z。最后，绘制决策边界（使用红黄蓝颜色映射），并用 plt.scatter 绘制标准化后的数据点。展示非线性 SVM 的分类边界和数据分布，直观分析模型的分类效果。



（基于软间隔 SVM 的可视化结果）

3.3.3 模型评估

```
软间隔 SVM（RBF 核）结果:
测试集准确率: 0.8611

分类报告:
              precision    recall  f1-score   support

           0       0.86      0.85      0.85        86
           1       0.86      0.87      0.87        94

    accuracy                           0.86       180
   macro avg       0.86      0.86      0.86       180
weighted avg       0.86      0.86      0.86       180


混淆矩阵:
[[73 13]
 [12 82]]
```

（基于 RBF 核的软间隔 SVM 模型结果）

逻辑回归模型和硬间隔支持向量机（SVM）模型在测试集上的准确率分别为 86.11% 和 85.56%，表现接近，但逻辑回归略优。逻辑回归的模型系数（[-0.33113448, -2.74494566]）显示第二个特征对分类的影响远大于第一个。两者均假设数据线性可分，但逻辑回归的稍高准确率可能表明其对数据的拟合更灵活。硬间隔 SVM 的较低准确率可能受限于其严格的线性分离要求，若数据存在噪声或非线性结构，性能可能受影响。综合来看，逻辑回归在当前数据集上表现略优且计算成本较低，但若数据线性可分性较差，可考虑软间隔 SVM 或非线性核函数以提升性能。

3.4 线性软间隔 SVM（基于 liblinear，来源论文二中的 SGD）

3.4.1 代码实现

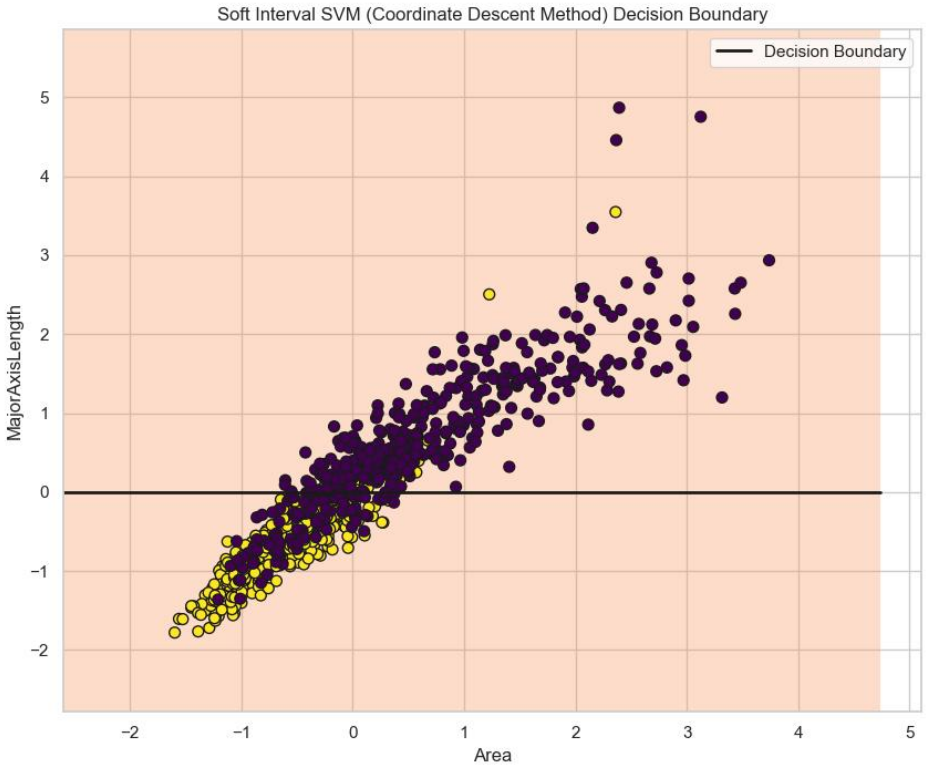
```
1个用法
def fit(self, X, y):
    """
    训练 SVM 模型，使用坐标下降法优化对偶问题。

    参数:
        X (np.ndarray): 特征矩阵，形状 (n_samples, n_features)。
        y (np.ndarray): 目标变量，值为 {-1, 1}。   你
    """
    n_samples, n_features = X.shape
    y = np.where(y == 0, -1, y: 1)  # 转换为 {-1, 1} 标签
    self.alpha = np.zeros(n_samples)  # 初始化拉格朗日乘子
    Xy = X * y[:, np.newaxis]  # 预计算 X * y，优化计算
    Q_diag = np.sum(X ** 2, axis=1)  # Q 矩阵对角线元素：x_i^T x_i

    # 坐标下降法主循环
    for iter_ in range(self.max_iter):
        max_diff = 0  # 记录 alpha 变化的最大值，用于收敛判断
        for i in range(n_samples):
            # 计算梯度 G_i = 1 - y_i * (w^T x_i)
            w_dot_x = np.dot(X[i], np.sum(Xy * self.alpha[:, np.newaxis], axis=0))
            G_i = 1 - y[i] * w_dot_x

            # 保存旧的 alpha 值
            alpha_old = self.alpha[i]

            # 更新 alpha_i
            self.alpha[i] = min(max(self.alpha[i] - G_i / (Q_diag[i] + 1e-8), 0), self.C)

            # 计算 alpha 变化量
            diff = abs(self.alpha[i] - alpha_old)
            max_diff = max(max_diff, diff)

        # 检查收敛
        if max_diff < self.tol:
            print(f"坐标下降法在第 {iter_ + 1} 次迭代收敛")
```

（基于 liblinear 的软间隔 SVM 代码实现）

此处通过设计函数 train_coordinate_descent_svm 实现了一个基于坐标下降法的软间隔 SVM 模型，通过自定义的 CoordinateDescentSVM 类优化对偶问题。CoordinateDescentSVM 类通过迭代更新拉格朗日乘子（alpha），计算权重向量 w 和偏置项 b，并识别支持向量，允许部分误分类（通过正则化参数 C 控制）。模型在标准化后的训练集上拟合后，对测试集进行预测，并输出准确率、分类报告和混淆矩阵以评估性能。返回值为训练好的模型、标准化器和准确率。之所以采用这种方法是因为可以通过坐标下降法来高效优化线性 SVM，适合线性可分或接近线性可分的数据。

3.4.2 可视化结果显示

我在可视化显示结果时使用了函数 plot_decision_boundary，用于显示基于坐标下降法训练的软间隔线性支持向量机（SVM）模型在二维特征空间中的决策边界和数据点分布。使用提供的 scaler 对特征进行标准化后，基于标准化特征的范围生成一个高分辨率（步长 0.01）的二维网格，并通过模型预测网格点类别以绘制分类区域。分类区域通过 plt.contourf 以红黄蓝颜色映射（透明度 0.3）显示，颜色由目标变量 y 决定。函数还利用模型的权重 w 和偏置 b 计算并绘制明确的决策边界直线（w1*x1 + w2*x2 + b = 0），以黑色实线表示，直观展示线性 SVM 的分类效果。



（Liblinear 算法的软间隔 SVM 的线性决策边界）

3.4.3 模型评估分析

自定义坐标下降法实现的软间隔线性 SVM 模型在测试集上的准确率仅为 52.22%，表现较差。相比逻辑回归（86.11%）、硬间隔 SVM（85.56%）和 RBF 核软间隔 SVM（86.11%），该模型性能显著较差，需调整 C 值、增加迭代次

数或考虑非线性核以改善分类效果。可能此数据集并不是特别合适该方法。

四、 文献阅读

4.1 论文一

4.1.1 论文简介

《Fast SVM classifier for large-scale classification problems》，发表在《Information Sciences》期刊（2023 年，第 642 卷，119136 号），作者为 Huajun Wang、Genghui Li 和 Zhenkun Wang。论文提出了一种新的支持向量机（SVM）模型，称为截断平方铰链损失支持向量机，旨在解决大规模分类问题中的计算复杂度和鲁棒性问题。

4.1.2 论文分析

这篇论文提出了一种新的 SVM 模型，通过引入截断平方铰链损失函数来提高模型的鲁棒性和稀疏性，同时降低计算成本。其实就是提出了一种新的损失函数，文章中写成L_{tx}，也就是截断平方铰链损失函数。

文章中是这样定义这个函数的：

$$\ell_{\mathrm{ts}}(\mathrm{r}) := \begin{cases} 1, & \mathrm{r} > 1 \\ \mathrm{r}^2, & \mathrm{r} \in [0,1] \\ 0, & \mathrm{r} < 0 \end{cases}$$

其中 ，$\mathrm{r} = 1 - \mathrm{y_i}(\langle \mathrm{w}, \mathrm{x_i} \rangle + \mathrm{b})$表示分类误差，从鲁棒性角度分析，当 r>1 时，损失固定为 1，限制了异常值对模型的过度影响，比传统铰链损失和平方铰链损失更鲁棒。

而从稀疏性的角度来分析，当 r<0 时，损失为 0，减少了支持向量的数量，从而提高了模型的稀疏性。

那么在有了新的损失函数以后，就可以提出新的 SVM 模型，也就是新的优化问题变成了：

$$\min_{\mathbf{w}\in\mathbb{R}^n, b\in\mathbb{R}} \frac{1}{2}\|\mathbf{w}\|^2 + \eta \sum_{i=1}^{m} \ell_{ts}\left(1 - y_i(\langle \mathbf{w}, \mathbf{x}_i\rangle + b)\right)$$

这就是这个文章中所提出的学习机，那么我们就有了正则化参数η，其中 x 和 y 分别是样本特征和标签。通过以上步骤，我们就完整地构建了一套有创新性的 SVM 算法。

这篇论文在创新方面，不仅在于提出了一种新的 loss 函数去构建 SVM 算法，还提出了一种新的交替方向乘子法（L_{tx}-ADMM），结合工作集策略，通过随机划分训练集为工作集和非工作集，降低计算复杂度。证明了算法的全局收敛性（公式 4.2），确保其收敛到近端驻点（即局部最优解）。算法在每次迭代中只处理部分数据，计算复杂度低，适合大规模数据集。并通过与八种其他 SVM 求解器（包括 HSVM、SSVM、PSVM 等）的比较，验证了L_{tx}-SVM 的优越性，即使用较少的支持向量，以及较短的训练时间，就能鲁棒性较好，准确率较高的结果。

4.2 论文二

4.2.1 论文简介

《SVM-SMO-SGD: A hybrid-parallel support vector machine algorithm using sequential minimal optimization with stochastic gradient descent》，发表在《Parallel Computing》期刊（2022 年，第 113 卷，102955）。作者为 Gizen Mutlu 和Çiğdem İnan Acı，提出了一种结合支持向量机（SVM）、顺序最小优化（SMO）和随机梯度下降（SGD）的混合并行算法，称为 SVM-SMO-SGD，旨在优化 SVM 在大规模数据集上的时间和资源消耗，同时保持高精度。

4.2.2 论文分析

这篇文章是为了处理当数据量极具增加时，普通 SVM 所需时间大大增加的问题，因此提出一种混合算法 SVM-SMO-SGD，通过结合 SMO（优化 SVM 权重计算）和 SGD（加速权重更新），在 CPU 和 GPU 上实现更高效的并行计算，减少时间和资源消耗，同时保持 SVM 的分类精度。在算法优化方面，主要体现在 SVM 与 SMO 的结合上。

基于传统的 SVM，其目标是找到一个最优超平面，用于最大化分类边界。在非线性情况下，SVM 通过核函数（如高斯径向基函数，RBF）将数据映射到高维空间，优化问题可以表示为以下二次规划形式：

$$\min_{\alpha_i} \varphi(\alpha) = \min_{\alpha_i} \left(\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j K(\vec{x}_i, \vec{x}_j) \alpha_i \alpha_j - \sum_{i=1}^{N} \alpha_i \right)$$

有约束条件：subject $\quad 0 \le \alpha_i \le C, \ \forall i,$
$\sum_{i=1}^{N} y_i \alpha_i = 0$

其中 C 为正则化参数，这是基于简单的 SVM 理论。而根据这样一个 RBF 的高斯核函数，我们可以进行非线性的软间隔 SVM 分类器，这也就是 3.3 中实现的一个分类情况。

而如果我们使用 SMO 来优化这个模型，让其在计算机中能够更快地进行计算，就可以通过分解 QP 问题来解决 SVM 的优化问题。每次迭代选择两个拉格朗日乘子（α_i和α_j）进行优化，而保持其他乘子不变，以满足线性等式约束 $\sum_{i=1}^{N} y_i \alpha_i = 0$。如此，我们就可以更新我们的参数：

$$\begin{aligned} \eta &= K(x_i, x_i) + K(x_j, x_j) - 2K(x_i, x_j) \\ \alpha_j^{new} &= \alpha_j + \frac{y_j(E_i - E_j)}{\eta} \\ \alpha_i^{new} &= \alpha_i + s\left(\alpha_j - \alpha_j^{new,}\right) \end{aligned}$$

这里的 E 是误差，也就是根据误差去更新拉格朗日乘子，从而达到优化的目的。

而使用随机梯度下降（SGD）通过随机选择样本更新权重向量 w ，解决 SVM 的原始形式优化问题。SVM 的损失函数在论文中表述为

$$\lambda \|w\|^2 + \max\{0, 1 - y w^{\top} \phi(x)\}$$

同时也通过 SGD 进行权重更新规则:

$$w \leftarrow w - \gamma \begin{cases} \lambda w & 若 y_i w^{\top} \phi(x_i) > 1 \\ \lambda w - y_i \phi(x_i) & 否则 \end{cases}$$

在这里γ是学习率，一个常在神经网络中出现的参数，这里决定了参数 w 受

影响的程度。SGD 通过随机选择样本进行梯度更新，在牺牲部分精度的前提下降低了计算成本，适合大规模数据集。