**中山大学计算机学院**

**人工智能**

**本科生实验报告**

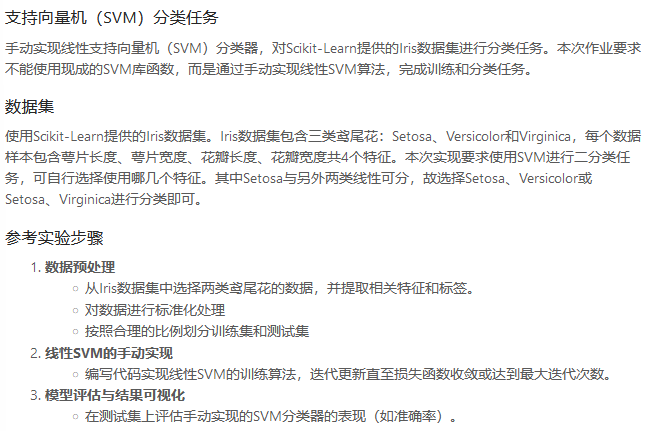
**（2022学年春季学期）**

课程名称：Artificial Intelligence

|  |  |  |  |
| --- | --- | --- | --- |
| 教学班级 | **202320346** | 专业（方向） | **计算机科学与技术** |
| 学号 | **21312450** | 姓名 | **林隽哲** |

# 实验题目

支持向量机（SVM）手动实现



# 实验内容

1. 算法原理

**原始问题（primal problem）**

原问题是一个凸二次规划问题（Quadratic Programming, QP），如下所示：

图示

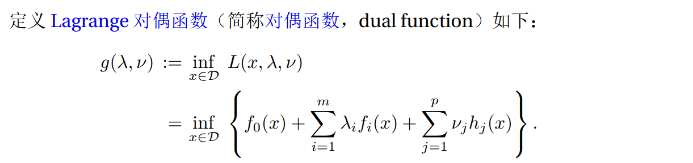
描述已自动生成

文本

中度可信度描述已自动生成

原问题可以通过应用**拉格朗日乘子法**构造**拉格朗日函数（Lagrange function）**再通过求解其**对偶问题（dual problem）**得到原始问题的最优解。

**拉格朗日（Lagrange）函数**

****

****

****

不难看出，函数 实际上给出了**原问题最优值的下界**。我们令：

手表上有字

描述已自动生成

原问题可以被转换为以下的等价问题：

文本

低可信度描述已自动生成

**Lagrange对偶问题**

**文本

描述已自动生成**

**文本

描述已自动生成**

**对偶性下的最优性条件**

**文本, 信件

描述已自动生成**

**文本

描述已自动生成**

**文本

描述已自动生成**

总结以上提到的算法原理，我们首先再次明确原问题与对偶问题：

手表上有字

中度可信度描述已自动生成

示意图

描述已自动生成

图片包含 表格

描述已自动生成

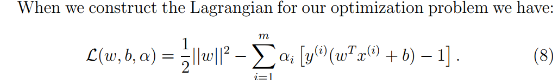
**IN OUR CASE**

**分类任务对应的原始问题：**

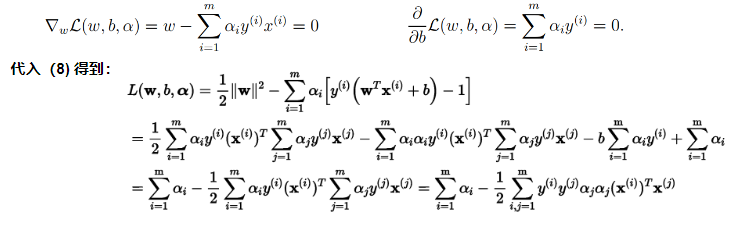
图示

描述已自动生成

**对应的拉格朗日函数：**

****

**带入KKT条件对式子进行化简：**

****

**从而我们将原问题的求解转换为对偶问题的求解：**

**文本, 示意图

描述已自动生成**

**求解对偶问题：**

我们可以通过**SMO算法、AML方法**等等求解出能使W(α)达到最大的α值，然后在求解出取得最有解时对应的w和b：

文本

中度可信度描述已自动生成

文本

低可信度描述已自动生成

文本

描述已自动生成



**软间隔问题求解**

**电脑屏幕的照片上有文字

中度可信度描述已自动生成**

**文本, 信件

描述已自动生成**

**文本

描述已自动生成**

**文本, 信件

描述已自动生成**

**文本, 信件

描述已自动生成**

**图表, 散点图

描述已自动生成**

**SMO算法（Sequential minimal optimization）**

**图形用户界面, 文本

描述已自动生成**

**文本

描述已自动生成**

**文本, 信件

描述已自动生成文本, 信件

描述已自动生成**

**图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成**

**文本, 信件

描述已自动生成**

**文本

描述已自动生成**

**文本

描述已自动生成**

1. 伪代码

图片包含 日程表

描述已自动生成

**注：在实际进行代码的编写时，我简化了变量选择的过程，具体可见以下的代码。**

1. 关键代码展示

**SVM模型（使用SMO算法进行求解）**

class SVM:

    def \_\_init\_\_(self, C=1.0, max\_iter=1000, tol=1e-3):

        """

        :param C: 惩罚系数

        :param max\_iter: 最大迭代次数

        :param tol: 容忍度

        """

        self.C = C

        self.max\_iter = max\_iter

        self.tol = tol

        """

        self.alphas: 拉格朗日乘子

        self.w: 权重向量

        self.b: 偏置项

        """

        self.alphas = None

        self.w = None

        self.b = None

    def fit(self, X, y): # SMO

        n\_samples, n\_features = X.shape

        self.alphas = np.zeros(n\_samples)

        self.w = np.zeros(n\_features)

        self.b = 0

        for \_ in range(self.max\_iter):

            alpha\_prev = np.copy(self.alphas)

            for j in range(n\_samples):

                i = np.random.randint(0, n\_samples)

                while i == j:

                    i = np.random.randint(0, n\_samples)

                xi, xj, yi, yj = X[i], X[j], y[i], y[j]

                # kii, kjj, kij = np.dot(xi, xi), np.dot(xj, xj), np.dot(xi, xj)

                kii, kjj, kij = self.\_kernel(xi, xi), self.\_kernel(xj, xj), self.\_kernel(xi, xj)

                eta = 2 \* kij - kii - kjj

                Ei = self.\_decision\_function(xi) - yi

                Ej = self.\_decision\_function(xj) - yj

                if eta == 0:

                    continue

                alpha\_j\_old = self.alphas[j]

                # 计算L和H (L <= alpha\_j\_new <= H)

                if yi != yj:

                    L = max(0, self.alphas[j] - self.alphas[i])

                    H = min(self.C, self.C + self.alphas[j] - self.alphas[i])

                else:

                    L = max(0, self.alphas[j] + self.alphas[i] - self.C)

                    H = min(self.C, self.alphas[j] + self.alphas[i])

                if L == H:

                    continue

                # 计算新的alpha值

                alpha\_j\_new = np.clip(alpha\_j\_old - yj \* (Ei - Ej) / eta, L, H)

                alpha\_i\_new = self.alphas[i] + yi \* yj \* (alpha\_j\_old - alpha\_j\_new)

                self.alphas[j] = alpha\_j\_new

                self.alphas[i] = alpha\_i\_new

            # 检查alpha是否有足够的变化

            diff = np.linalg.norm(self.alphas - alpha\_prev)

            if diff < self.tol:

                break

        # 计算权重向量w: w = Σ (alpha\_i \* y\_i \* x\_i)

        self.w = np.sum(self.alphas[:, None] \* y[:, None] \* X, axis=0)

        # 计算偏置项b: b = 1/n\_samples \* Σ (y\_i - w^T x\_i)

        self.b = np.mean([yi - np.dot(self.w, xi) for xi, yi in zip(X, y)])

    def \_decision\_function(self, x):

        # f(x) = w^T x + b

        return np.dot(x, self.w) + self.b

    def \_kernel(self, x1, x2):

        return np.dot(x1, x2)

    def predict(self, X):

        return np.sign(self.\_decision\_function(X))

当然，你也可以不直接通过KKT条件，而改用常规的梯度下降来求解该问题。你同样可以通过支持向量的条件实现最大化分类间隔的迭代，如下：

**SVM模型（使用梯度下降）**

class LinearSVM:

    def \_\_init\_\_(self, learning\_rate=0.01, lambda\_param=0.01, n\_iters=1000):

        """

        :param learning\_rate: 学习率

        :param lambda\_param: 正则化参数

        :param n\_iters: 迭代次数

        """

        self.lr = learning\_rate

        self.lambda\_param = lambda\_param

        self.n\_iters = n\_iters

        """

        self.w: 权重向量

        self.b: 偏置项

        """

        self.w = None

        self.b = None

    def fit(self, X, y): # gradient descent

        n\_samples, n\_features = X.shape

        y\_ = np.where(y <= 0, -1, 1)

        self.w = np.zeros(n\_features)

        self.b = 0

        for \_ in range(self.n\_iters):

            for idx, x\_i in enumerate(X):

                condition = y\_[idx] \* (np.dot(x\_i, self.w) - self.b) >= 1

                if condition: # 正确分类的情况

                    self.w -= self.lr \* (2 \* self.lambda\_param \* self.w)

                else:

                    self.w -= self.lr \* (2 \* self.lambda\_param \* self.w - np.dot(x\_i, y\_[idx]))

                    self.b -= self.lr \* y\_[idx]

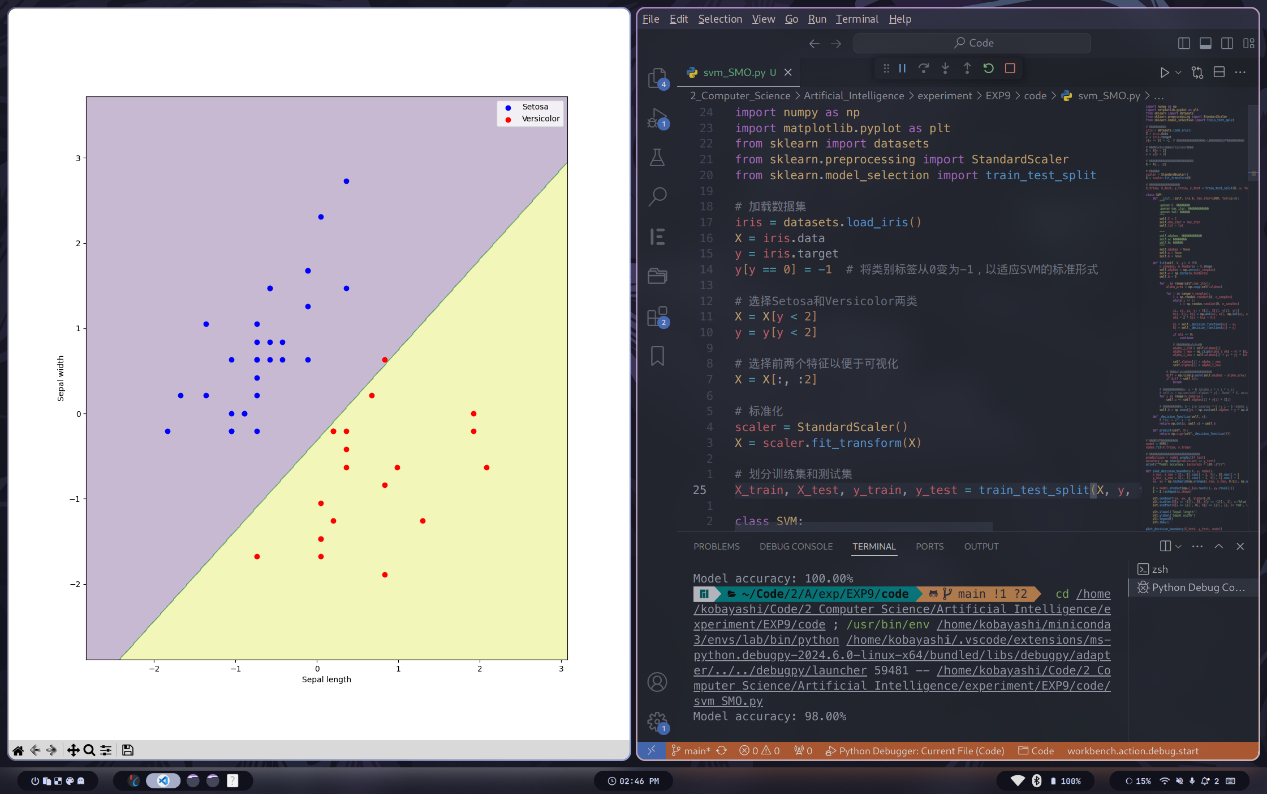
    def predict(self, X):

        linear\_output = np.dot(X, self.w) - self.b

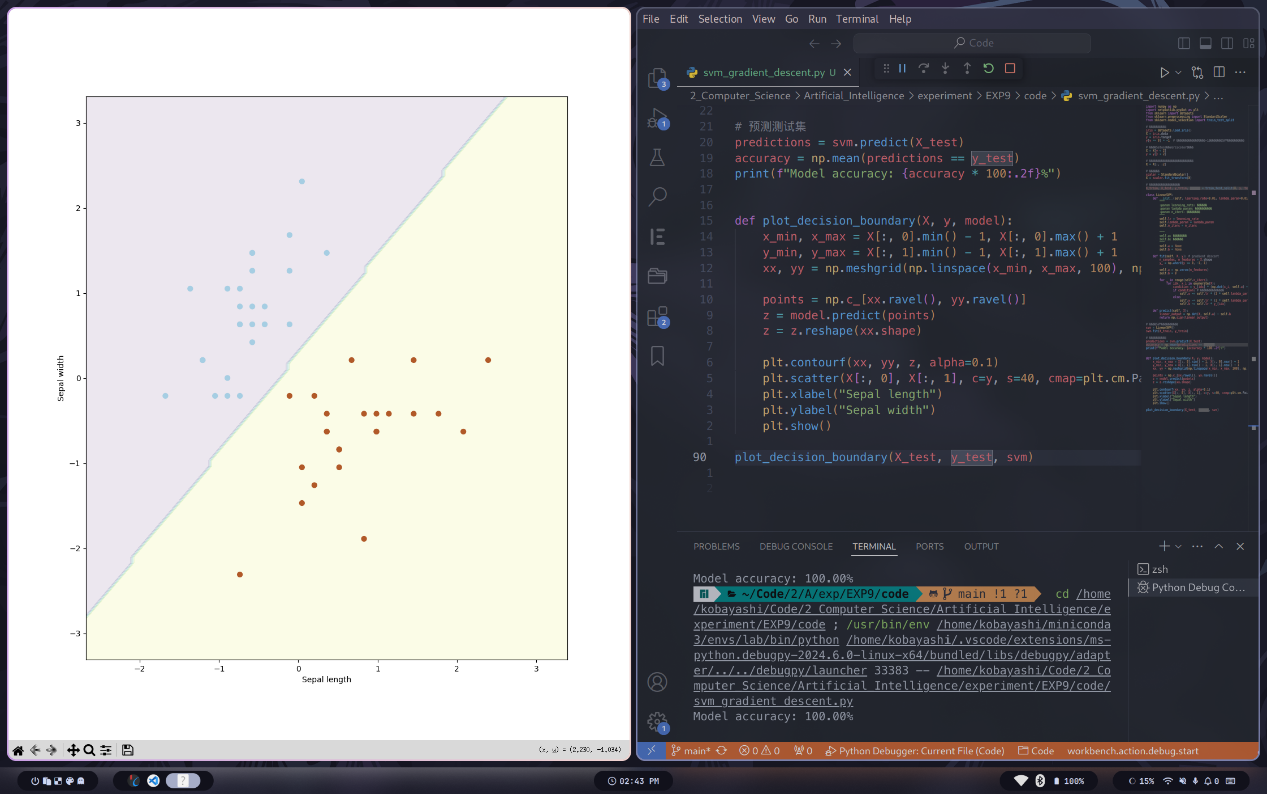
        return np.sign(linear\_output)

# 实验结果及分析

**SVM（SMO）**

****

**SVM（Gradient Descent）**

****

# 参考资料

* 理论课课件