中山大学计算机学院人工智能本科生实验报告

课程：Artificial Intelligence

姓名： 学号：

一、实验题目

手动实现支持向量机。

二、实验内容

1.算法原理

在本次实验中，简便起见，选用鸢尾花的萼片长度与萼片宽度作为特征。这样分类超平面就是一条直线，只需要三个参数即可确定。

在感知机算法中已经分析过关于直线与其相关性质。设分类直线表示为$w_0+w_1x_1+w_2x_2=0$。那么数据点与分类直线的带方向的距离显然就是：

$$d_i=\frac{w_0+w_1x_{i1}+w_2x_{i2}}{\sqrt{w_1^2+w_2^2}}$$

接下来分类讨论。对于支持向量来说，其到分类直线的距离恰好是d，而其余的点则大于d。即：

$$\begin{cases}\dfrac{w_0+w_1x_{i1}+w_2x_{i2}}{\sqrt{w_1^2+w_2^2}}\geq d \quad , \quad y_i=1\\ \dfrac{w_0+w_1x_{i1}+w_2x_{i2}}{\sqrt{w_1^2+w_2^2}}\leq -d \quad , \quad y_i=-1\end{cases}$$

这显然等价于一个方程：

$$\frac{y_i(w_0+w_1x_{i1}+w_2x_{i2})}{\sqrt{w_1^2+w_2^2}}\geq d$$

也等价于：

$$\frac{y_i(w_0+w_1x_{i1}+w_2x_{i2})}{d\sqrt{w_1^2+w_2^2}}\geq 1$$

一般来说，总令$d\sqrt{w_1^2+w_2^2}=1$，使得算法最后得到的参数总是统一的——因为参数同时乘除一个非零常数不会改变直线的位置。也就是说问题现在转化为：

$$y_i(w_0+w_1x_{i1}+w_2x_{i2})\geq 1$$

上式在取等于1时，描述了刚好实现二分类的两个分类直线(因为y_i有正负之分)：也就是说，强行令支持向量恰好符合$y_i(w_0+w_1x_{i1}+w_2x_{i2})=1$。于是重新计算支持向量到分类直线的距离：

$$d=\left|\frac{w_0+w_1x_{i1}+w_2x_{i2}}{\sqrt{w_1^2+w_2^2}}\right|=\frac{1}{\sqrt{w_1^2+w_2^2}}$$

而支持向量机的分类直线的目标就是使d最大，而这等价于令下式最小：

$$L(w)=w_1^2+w_2^2$$

考虑到约束条件，即对于所有点下式成立：

$$y_i(w_0+w_1x_{i1}+w_2x_{i2})\geq 1$$

等价于：

$$1-y_i(w_0+w_1x_{i1}+w_2x_{i2})\leq 0$$

等价于对任意点(x_{i1},x_{i1})总存在一个参数α_i^2(平方是为了保证其非负)使得：

$$1-y_i(w_0+w_1x_{i1}+w_2x_{i2})+\alpha_i^2=0$$

这样即可构造拉格朗日函数：

$$L(w;\lambda;\alpha)=w_1^2+w_2^2+\sum_{i=1}^{n}\lambda_i(1-y_i(w_0+w_1x_{i1}+w_2x_{i2})+\alpha_i^2)$$

在这里要求$\lambda_i\geq 0$，问题转化为求上式的最小值。然而根据条件存在$\lambda_i\alpha_i^2\geq 0$，而$\alpha_i^2$不由$\lambda_i$决定。既然是要求拉格朗日函数的最小值，上式可以直接剔除掉α_i^2这个参数以减少变量：

$$L(w;\lambda)=w_1^2+w_2^2+\sum_{i=1}^{n}\lambda_i(1-y_i(w_0+w_1x_{i1}+w_2x_{i2}))$$

现在的任务是求使$L(w;\lambda)$最小的参数$w=[w_0,w_1,w_2]$，记为$min_wL(w;\lambda)$。而根据上述提到的条件，求和号内必然是负数，因此不服从约束的w参数取值可能会使拉格朗日函数更小，所以对于参数$\lambda=[\lambda_1,...\lambda_n]$来说实际是要使拉格朗日函数最大。即问题的全貌是：

$$min_wmax_\lambda L(w;\lambda)$$

由于此问题满足KKT条件，上式亦等价于其对偶问题：

$$max_\lambda min_wL(w;\lambda)$$

现在来求$L(w;\lambda)$的偏导以寻找极值条件：

$$\begin{cases} \dfrac{\partial L}{\partial w_0} = -\sum \lambda_i y_i = 0 \\ \dfrac{\partial L}{\partial w_1} = 2w_1 - \sum \lambda_i x_{i1} y_i = 0 \\ \dfrac{\partial L}{\partial w_2} = 2w_2 - \sum \lambda_i x_{i2} y_i = 0 \\ \dfrac{\partial L}{\partial \lambda_i} = 1 - y_i(w_0 + w_1 x_{i1} + w_2 x_{i2}) = 0 \end{cases}$$

故：

$$\begin{cases} \sum \lambda_i y_i = 0 \\ w_1 = \dfrac{1}{2} \sum \lambda_i x_{i1} y_i \\ w_2 = \dfrac{1}{2} \sum \lambda_i x_{i2} y_i \end{cases}$$

据此，重写拉格朗日函数：

$$\begin{aligned} L(w;\lambda) &= w_1^2 + w_2^2 + \sum \lambda_i - w_0 \sum \lambda_i y_i - w_1 \sum \lambda_i x_{i1} y_i - w_2 \sum \lambda_i x_{i2} y_i \\ &= \frac{1}{4}\left(\sum \lambda_i x_{i1} y_i\right)^2 + \frac{1}{4}\left(\sum \lambda_i x_{i2} y_i\right)^2 + \sum \lambda_i - \frac{1}{2}\left(\sum \lambda_i x_{i1} y_i\right)^2 - \frac{1}{2}\left(\sum \lambda_i x_{i2} y_i\right)^2 \\ &= \sum \lambda_i - \frac{1}{4}\left(\sum \lambda_i x_{i1} y_i\right)^2 - \frac{1}{4}\left(\sum \lambda_i x_{i2} y_i\right)^2 \\ &= \sum_{i=1}^{n} \lambda_i - \frac{1}{4} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j (x_{i1} x_{j1} + x_{i2} x_{j2}) \\ &= L(\lambda) \end{aligned}$$

也就是说，现在拉格朗日函数中只剩下$\lambda = [\lambda_1, ...\lambda_n]$的参数了！而我们的目标则是$max\ L(\lambda)$，同时不要漏了约束条件$\sum \lambda_i y_i = 0$。

接下来利用SMO来实现最优参数的搜寻。假设选取的两个优化变量是λ_k, λ_l。那么首先将$L(\lambda) = L(\lambda_k, \lambda_l)$展开为只包含这两个变量的函数，其中包含各自的二次项和一次项、还有两者的交叉项(注意到$y_i^2 = 1$)：

$$
\begin{aligned}
L(\lambda_k, \lambda_l) = (\lambda_k + \lambda_l) - \frac{1}{4}\lambda_k^2(x_{k1}^2 + x_{k2}^2) - \frac{1}{4}\lambda_l^2(x_{l1}^2 + x_{l2}^2) \\
- \frac{1}{4}\lambda_k\lambda_l y_k y_l(x_{k1}x_{l1} + x_{k2}x_{l2}) \\
- \frac{1}{4}\lambda_k y_k \sum_{i=1, i\neq k}^{n} \lambda_i y_i(x_{k1}x_{i1} + x_{k2}x_{i2}) \\
- \frac{1}{4}\lambda_k y_l \sum_{i=1, i\neq l}^{n} \lambda_i y_i(x_{l1}x_{i1} + x_{l2}x_{i2})
\end{aligned}
$$

求关于λ_k的偏导并令其为0：

$$
\begin{aligned}
4\frac{\partial L}{\partial \lambda_k} = 4 - 2\lambda_k(x_{k1}^2 + x_{k2}^2) - \lambda_l y_k y_l(x_{k1}x_{l1} + x_{k2}x_{l2}) \\
- y_k \sum_{i=1, i\neq k}^{n} \lambda_i y_i(x_{k1}x_{i1} + x_{k2}x_{i2}) = 0
\end{aligned}
$$

记$NK = \sum_{i=1, i\neq k}^{n} \lambda_i y_i(x_{k1}x_{i1} + x_{k2}x_{i2})$，这样即可解得：

$$
\lambda_k^* = \frac{4 - y_k \cdot NK - \lambda_l y_k y_l(x_{k1}x_{l1} + x_{k2}x_{l2})}{2(x_{k1}^2 + x_{k2}^2)}
$$

也就是在每一迭代步中将λ_k更新为右边的值。但由于$\sum \lambda_i y_i = 0$，所以还需要更新λ_l。这意味着：

$$
\lambda_k y_k + \lambda_l y_l = \lambda_k^* y_k + \lambda_l^* y_l
$$

故：

$$
\begin{aligned}
\lambda_l^* &= \frac{\lambda_k y_k + \lambda_l y_l - \lambda_k^* y_k}{y_l} \\
&= (\lambda_k y_k + \lambda_l y_l - \lambda_k^* y_k) y_l \\
&= (\lambda_k - \lambda_k^*) y_k y_l + \lambda_l
\end{aligned}
$$

在收敛之后、或是迭代次数之后，可以从参数λ中恢复分类直线的系数：

$$
\begin{cases}
w_1 = \frac{1}{2}\sum \lambda_i x_{i1} y_i \\
w_2 = \frac{1}{2}\sum \lambda_i x_{i2} y_i
\end{cases}
$$

为了恢复w_0，需要利用支持向量有关的结论，即支持向量满足：

$$
y_i(w_0 + w_1 x_{i1} + w_2 x_{i2}) = 1
$$

故$w_0 = y_i - w_1 x_{i1} - w_2 x_{i2}$。那么如何判定支持向量？显然由于对非支持向量有$1 - y_i(w_0 + w_1 x_{i1} + w_2 x_{i2}) \le 0$，所以要使拉格朗日函数取最大值，应使非支持向量的λ取值为0。也就是说，支持向量的λ取值不为0。只需在λ参数列表中寻找不为零的值，再利用相关参数求值即可。

2.伪代码

首先给出类结构：

```
class SupportVectorMachine:
    def __init__(self) -> None:

    def __init_multiplier__(self) -> np.ndarray:
        """初始化拉格朗日乘子"""

    def __Constraint__(self, array: np.ndarray) -> float:
        """计算乘子约束条件"""

    def SeqMinOptimize(self, index1: int, index2: int) -> None:
        """更新其中两个拉格朗日乘子"""

    def Weight(self) -> "tuple[float, float, float]":
        """返回分类直线的系数"""

    def Accuracy(self, weight: tuple[float, float, float]) -> float:
        """返回分类准确率"""

    def Draw(self) -> None:
        """绘制结果"""

    def Solve(self) -> "tuple[float, float, float]":
        """对外的解决方法"""
```

各方法调用层级如下：

$$\begin{cases} init \rightarrow init\_multiplier \rightarrow Constraint \\ Solve \begin{cases} SeqMinOptimize \\ Draw \\ Accuracy \rightarrow Weight \end{cases} \end{cases}$$

其中$init, init_multiplier, Constraint$都是类内私有方法，为初始化数据集服务。因为拉格朗日乘子$\lambda_{0\sim n}$不能在初始时就取0，尽管这是最简单的令乘子符合约束条件$\sum \lambda_i y_i = 0$的取值方法。因此将乘子取任意值之后，计算$\sum \lambda_i y_i$，之后在其中任意选一更新其值，使得所有乘子符合约束。例如选择λ_k，那么应有：

$$\sum_{i \neq k} \lambda_i y_i + \lambda_k y_k = 0$$

所以:

$$\lambda_k = -y_k \sum_{i \neq k} \lambda_i y_i$$

但是需要注意$\forall \lambda_i \geq 0$的约束条件，也就是说要更改的λ_k也不能任意选择，而需要根据$\sum_{i \neq k} \lambda_i y_i$的符号来选择。

```
def __init_multiplier__(self) -> np.ndarray:
    """初始化拉格朗日乘子"""
    将n个乘子随机取任意非0值
    计算__Constraint__
    更新其中一个乘子以符合约束


def __Constraint__(self, array: np.ndarray) -> float:
    """计算乘子约束条件"""
    计算lambda[i] * y[i]之和
```

然后是从乘子返回直线系数的方法，这个只需要按照上述算法原理中所说的公式实现即可:

```
def Weight(self) -> "tuple[float, float, float]":
    """返回分类直线的系数"""
    for (问题中数据):
        w1 = w1 + 乘子[i] * x1[i] * y[i]
        w2 = w2 + 乘子[i] * x2[i] * y[i]
    w1, w2 = w1 / 2, w2 / 2
    寻找乘子值不为0的下标index
    w0 = y[index] - w1 * x1[index] - w2 * x2[index]
    return [w0, w1, w2]
```

要计算准确率，只需要将数据点代入直线方程，然后判断所得值符号即可:

```
def Accuracy(self, weight: tuple[float, float, float]) -> float:
    """返回分类准确率"""
    count = 0
    for (问题中数据):
        value = w0 + w1 * x1[i] + w2 * x2[i]
        if (value < 0) and (y[i] == -1): count++
        elif (value > 0) and (y[i] == 1): count++
    return (count / 数据容量)
```

接下来就是支持向量机的重点SMO算法。这里也只需要按照算法原理中所求公式实现即可:

```
def SeqMinOptimize(self, index1: int, index2: int) -> None:
    """更新其中两个拉格朗日乘子"""
    for i in (数据容量, 并且i != index1): # 计算NK
        temp = x1[index1] * x1[i] + x2[index1] * x2[i]
        sum = sum + temp * y[i] * 乘子[i]
    计算其他部分参数
    计算乘子[index1]*
    判断乘子[index1]*是否符合约束范围:
        若不符合，则更新为允许的最大值或最小值
    计算乘子[index2]*
    更新乘子[index1], 乘子[index2]
```

在对外的解决方法中，只需要在迭代步内不断调用SMO即可:

```
def Solve(self) -> "tuple[float, float, float]":
    """对外的解决方法"""
    for (迭代步):
        选择要更新的两个乘子(的下标)
        SMO(两个乘子的下标)
    Accuracy()
    return Weight()
```

3.关键代码展示

这里展示程序的关键，即SMO算法:

```python
def SeqMinOptimize(self, index1: int, index2: int) -> None:
    """更新其中两个拉格朗日乘子"""
    if (index1 == index2): return
    sum = 0.0
    for i in range(0, self.scale):
        if (i == index1): continue
        temp = self.iris_length[index1] * self.iris_length[i] + self.iris_width[index1] * self.iris_w
        sum = sum + self.multiplier[i] * self.iris_target[i] * temp
    temp = self.iris_length[index1] * self.iris_length[index2] + self.iris_width[index1] * self.iris_
    ykyl = self.iris_target[index1] * self.iris_target[index2]
    temp = temp * ykyl * self.multiplier[index2]
    new_multi_i = (4 - self.iris_target[index1] * sum - temp) / (2 * (self.iris_length[index1] * self
    if (ykyl < 0):
        if (new_multi_i < max(0, self.multiplier[index2] - self.multiplier[index1])):
            new_multi_i = max(0, self.multiplier[index2] - self.multiplier[index1])
    else:
        if (new_multi_i < 0): new_multi_i = 0
        elif (new_multi_i > self.multiplier[index2] + self.multiplier[index1]):
            new_multi_i = self.multiplier[index2] + self.multiplier[index1]
    new_multi_j = (self.multiplier[index1] - new_multi_i) * ykyl + self.multiplier[index2]
    self.multiplier[index1], self.multiplier[index2] = new_multi_i, new_multi_j
```

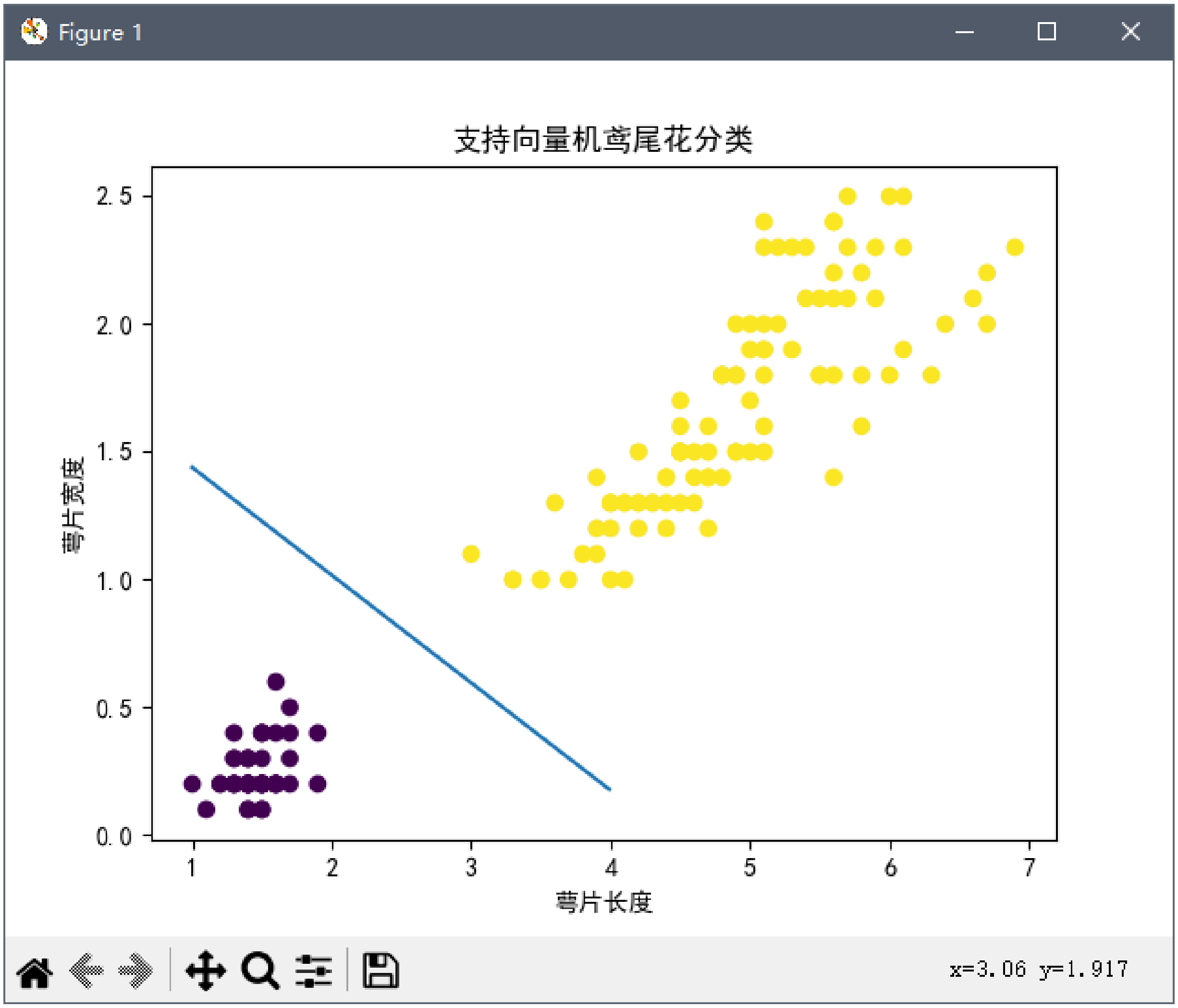
4.创新优化

无。

三、实验结果分析

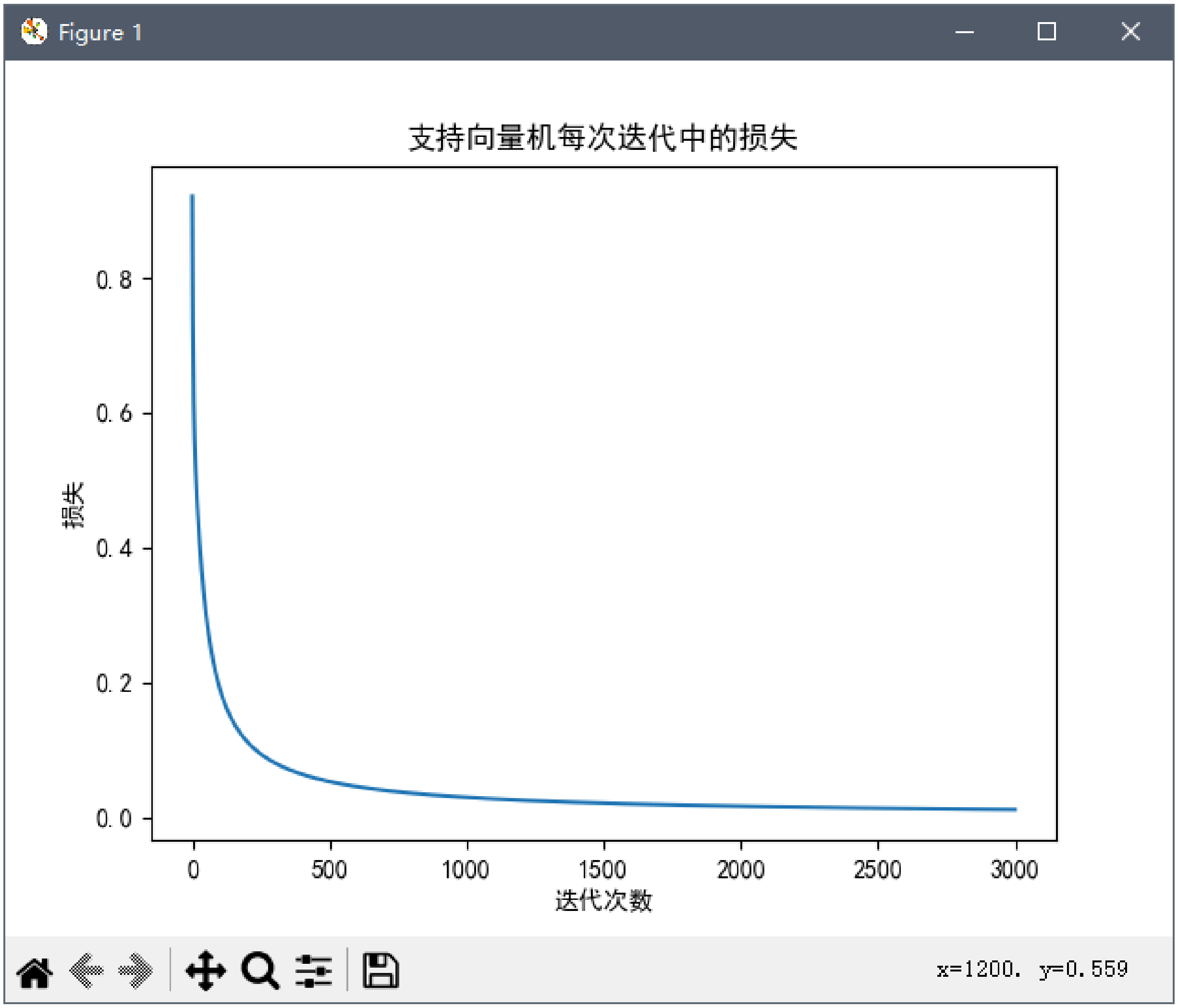
1.实验结果展示

分类直线如图所示：



2.评测指标展示分析

损失函数如下所示:



四、思考题

无。

五、参考资料

【机器学习】支持向量机 SVM（非常详细） - 阿泽的文章 - 知乎
https://zhuanlan.zhihu.com/p/77750026