SVM-SMO算法

已知训练集共m个训练样本，样本特征维度为n。

1. 初始化偏置$b=0$；任意的拉格朗日乘子$\alpha_i=0$；根据$E_i=\hat{y}^{(i)}-y^{(i)}$计算初始误差；设置错误惩罚参数$C=1$；

```
# 初始化模型
def init_args(self, features, labels):
    self.m, self.n = features.shape # m - 训练样本数；n - 特征数
    self.X = features
    self.Y = labels
    self.b = 0.0
    # 将Ei保存在一个列表里——Ei为g(xi)对输入xi的预测值和yi的差
    self.alpha = torch.ones(self.m)
    self.E = torch.tensor([self._e(i) for i in range(self.m)], dtype=torch.float)
    # 错误惩罚参数
    self.C = 1.0
```

2. 进入迭代循环，共循环max_iter次：

   a. 选择违反KKT条件的α_i作为α_1^{old}，KKT条件如下：

$$\sum_{i=1}^{N}\alpha_i y_i=0,\quad 0\leqslant\alpha_i\leqslant C,\quad i=1,2,\cdots,N$$

$$y_i\cdot g(x_i)\begin{cases}\geqslant 1, & \{x_i|\alpha_i=0\}\\ =1, & \{x_i|0<\alpha_i<C\}\\ \leqslant 1, & \{x_i|\alpha_i=C\}\end{cases}$$

其中，

$$g(x_i)=\sum_{j=1}^{N}\alpha_j y_j K(x_j,x_i)+b$$

选择使得$|E_i-E_j|$最大的α_j作为α_2^{old}；若没找到，则迭代结束。

```
# g(x)预测值，输入xi（X[i]）
def _g(self, xi):
    # self.alpha * self.Y * self.kernel(self.X, xi)对应位置相乘 - (800, 1)；
    # 返回一个标量
    return (self.alpha * self.Y * self.kernel(self.X, xi)).sum() + self.b

# 核函数,多项式添加二次项即可
def kernel(self, X_data, x2, gamma=1, r=0, d=3):
    if len(X_data.shape) > 1: # X_data为m个样本的集合
        res = []
        for x1 in X_data: # 取出每一个样本与x2做内积，存入res中
            res.append(self.kernel(x1, x2).item())
        return torch.tensor(res, dtype=torch.float) # (800*1)
    else: # 对单个样本x1对x2做内积
```

```
        x1 = X_data
        if self._kernel == 'linear':
            return (x1 * x2).sum()
        elif self._kernel == 'poly':
            return (gamma * (x1 * x2).sum() + r) ** d
        return 0

# kkt条件
  def _kkt(self, i):
      y_g = self._g(self.X[i]) * self.Y[i]
      if self.alpha[i] == 0:
          return y_g >= 1
      elif 0 < self.alpha[i] < self.C:
          return y_g == 1
      else:
          return y_g <= 1

# 选择优化变量alpha_i, alpha_j；其中alpha_i为违反KKT条件的样本，alpha_j是使更新最大的变量
  def _init_alpha(self):
      # 外层循环首先遍历所有满足0<a<C的样本点，检验是否满足KKT
      index_list = [i for i in range(self.m) if 0 < self.alpha[i] < self.C]
      # 否则遍历整个训练集
      non_satisfy_list = [i for i in range(self.m) if i not in index_list]
      # extend() 函数用于在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表）
      index_list.extend(non_satisfy_list)
      for i in index_list: # i - 样本下标；index_list - 存样本下表的列表
          if self._kkt(i):
              continue
          E1 = self.E[i]
          # 如果E1是+，选择最小的；如果E1是负的，选择最大的
          if E1 >= 0:
              j = torch.argmin(self.E)
          else:
              j = torch.argmax(self.E)
          return i, j
```

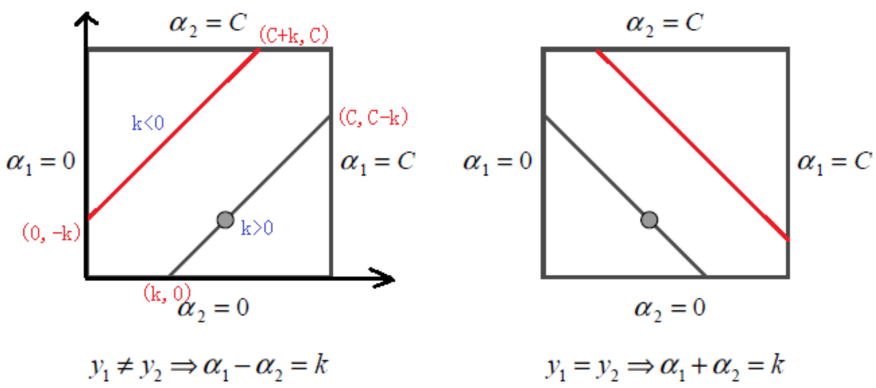
b. 计算$\eta = K_{11} + K_{22} - 2K_{12} = ||\Phi(x_1) - \Phi(x_2)||^2$；

```
# eta=K11+K22-2K12
eta = self.kernel(self.X[i1], self.X[i1]) + self.kernel(self.X[i2], self.X[i2]) - 2 * self.kernel(
    self.X[i1], self.X[i2])
if eta <= 0:
    continue
```

c. 获取E_1, E_2，根据公式$\alpha_2^{new,unc} = \alpha_2^{old} + \frac{y_2(E_1 - E_2)}{\eta}$计算未剪辑的$\alpha_2^{new,unc}$。

```
# 获取E1和E2
E1 = self.E[i1]
E2 = self.E[i2]
alpha2_new_unc = self.alpha[i2] + self.Y[i2] * (E2 - E1) / eta
```

d. 计算$\alpha_2^{new,unc}$的取值边界L和H：



$$
\begin{array}{ll}
y_1 \neq y_2 & \left\{\begin{array}{l} L=\max\left(0,\ \alpha_2^{old}-\alpha_1^{old}\right) \\ H=\min\left(C,\ C+\alpha_2^{old}-\alpha_1^{old}\right) \end{array}\right. \\
y_1 = y_2 & \left\{\begin{array}{l} L=\max\left(0,\ \alpha_2^{old}+\alpha_1^{old}-c\right) \\ H=\min\left(C,\ \alpha_2^{old}+d_1^{old}\right) \end{array}\right.
\end{array}
$$

```
# 对new_alpha2解进行裁剪
if self.Y[i1] == self.Y[i2]:
    L = max(0, self.alpha[i1] + self.alpha[i2] - self.C)
    H = min(self.C, self.alpha[i1] + self.alpha[i2])
else:
    L = max(0, self.alpha[i2] - self.alpha[i1])
    H = min(self.C, self.C + self.alpha[i2] - self.alpha[i1])
```

e. 对$\alpha_2^{new,unc}$进行剪辑得到α_2^{new}，公式如下：

$$
\alpha_2^{\text{new}} = \left\{\begin{array}{ll} H, & \alpha_2^{\text{new, unc}} > H \\ \alpha_2^{\text{new,unc}}, & L \leqslant \alpha_2^{\text{new , unc}} \leqslant H \\ L, & \alpha_2^{\text{new, unc}} < L \end{array}\right.
$$

```
alpha2_new = self._compare(alpha2_new_unc, L, H)
```

f. 根据$\alpha_2^{new}, \alpha_1^{old}, \alpha_2^{old}$计算得到$\alpha_1^{new}$，公式如下：

$$
\alpha_1^{\text{new}} = \alpha_1^{\text{old}} + y_1 y_2 \left(\alpha_2^{\text{old}} - \alpha_2^{\text{new}}\right)
$$

```
# 根据alpha2_new, alpha1_old, alpha2_old更新alpha1_new
alpha1_new = self.alpha[i1] + self.Y[i1] * self.Y[i2] * (self.alpha[i2] - alpha2_new)
```

g. 根据$\alpha_1^{new}, \alpha_2^{new}, \alpha_1^{old}, \alpha_2^{old}$和$b^{old}$计算得到$b_1^{new}, b_2^{new}$，公式如下：

$$
\begin{aligned}
b_1^{\text{new}} &= -E_1 - y_1 K_{11}\left(\alpha_1^{\text{new}} - \alpha_1^{\text{old}}\right) - y_2 K_{21}\left(\alpha_2^{\text{new}} - \alpha_2^{\text{old}}\right) + b^{\text{old}} \\
b_2^{\text{new}} &= -E_2 - y_1 K_{12}\left(\alpha_1^{\text{new}} - \alpha_1^{\text{old}}\right) - y_2 K_{22}\left(\alpha_2^{\text{new}} - \alpha_2^{\text{old}}\right) + b^{\text{old}}
\end{aligned}
$$

```
# 根据alpha1_new, alpha2_new, alpha1_old, alpha2_old和b_old 更新b1_new, b2_new
b1_new = -E1 - self.Y[i1] * self.kernel(self.X[i1], self.X[i1]) * (alpha1_new - self.alpha[i1]) - self.Y[
    i2] * self.kernel(self.X[i2], self.X[i1]) * (alpha2_new - self.alpha[i2]) + self.b
b2_new = -E2 - self.Y[i1] * self.kernel(self.X[i1], self.X[i2]) * (alpha1_new - self.alpha[i1]) - self.Y[
    i2] * self.kernel(self.X[i2], self.X[i2]) * (alpha2_new - self.alpha[i2]) + self.b
```

h. 根据α_1^{new}和α_2^{new}的取值范围以确定b^{new}，判断条件如下：

$$
b^{\text{new}} = \begin{cases} b_1^{\text{new}} & \text{if } 0 < \alpha_1^{\text{new}} < c \\ b_2^{\text{new}} & \text{if } 0 < \alpha_2^{\text{new}} < c \\ \frac{b_1^{\text{new}} + b_2^{\text{new}}}{2} & \text{otherwise} \end{cases}
$$

```
# 做最后判断
if 0 < alpha1_new < self.C:
    b_new = b1_new
elif 0 < alpha2_new < self.C:
    b_new = b2_new
else:
    # 选择中点
    b_new = (b1_new + b2_new) / 2
```

i. 更新参数

```
# 更新参数
self.alpha[i1] = alpha1_new
self.alpha[i2] = alpha2_new
self.b = b_new
self.E[i1] = self._e(i1)
self.E[i2] = self._e(i2)
```

j. 进入下一轮迭代