SVM Report

支持向量机

支持向量机（SVM）用来解决二分类目标中确定最大间隔超平面的问题，优化目标为

$$\min_{w,b} \frac{1}{2}\|w\|^2$$
$$s.t. y_i\left(w^{\top} x_i+b\right) \geq 1 \qquad i=1, \cdots, m$$

软间隔SVM

软间隔支持向量机是为了解决线性不可分问题而设计的，它允许向量机在一些样本上不满足约束条件，因此，优化目标变为

$$\min_{w,b} \frac{1}{2}||w||^2 + C\sum_{i=1}^{m} \ell_{0/1}(y_i(w^T x_i + b) - 1)$$

为了便于求导，将0/1损失函数替换为hinge损失函数，优化目标变为

$$\min_{w,b} \ \frac{1}{2}||w||^2 + C\sum_{i=1}^{m} \max\left(0, 1 - y_i\left(w^{\mathrm{T}} x_i + b\right)\right)$$

引入松弛变量后，用拉格朗日数乘法，可得对偶问题

$$\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \ \sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i\alpha_j y_i y_j x_i^{\mathrm{T}} x_j$$

$$\text{s.t.} \quad \sum_{i=1}^{m} \alpha_i y_i = 0 \ , 0 \leqslant \alpha_i \leqslant C \ , \quad i = 1, 2, \ldots, m$$

序列最小化SMO

SMO即为

序列最小化SMO用来求解上式中最优的 α ，不断执行下面两个步骤直至收敛

第一步：选取一对需更新的变量 α_i 和 α_j

第二步：固定 α_i 和 α_j 以外的参数, 求解对偶问题更新 α_i 和 α_j

则优化目标变为

$$g(\alpha_i, \alpha_j) = \alpha_i + \alpha_j - \frac{1}{2}\left(\alpha_i^2 K_{ii} + \alpha_j^2 K_{jj} + 2\alpha_i\alpha_j y_i y_j K_{ij} + a_i y_i \sum_{k\neq i,j} a_k y_k K_{ik} + \alpha_j y_j \sum_{k\neq i,j} a_k y_k K_{jk}\right)$$

$$\alpha_i y_i + \alpha_j y_j = -\sum_{k\neq i,j}^{m} \alpha_i y_i = \zeta$$

把α_i 用 α_j 表示，改写优化变量，求导并整理，得到

$$\alpha_j^{new} = \alpha_j^{old} + \frac{y_j(E_i - E_j)}{K}$$

再考虑限制条件 $0 \leqslant \alpha_j \leqslant C$，得到其下界和上界分别为

$$L=\begin{cases}\max\left\{0,\alpha_j^{\text{old}}-\alpha_i^{\text{old}}\right\} & y_iy_j=-1\\ \max\left\{0,\alpha_i^{\text{old}}+\alpha_j^{\text{old}}-C\right\} & y_iy_j=1\end{cases},H=\begin{cases}\min\left\{C,C+\alpha_j^{\text{old}}-\alpha_i^{\text{old}}\right\} & y_iy_j=-1\\ \min\left\{C,\alpha_j^{\text{old}}+\alpha_i^{\text{old}}\right\} & y_iy_j=1\end{cases}$$

所以

$$\alpha_i^{new}=\alpha_i^{old}+y_iy_j(\alpha_j^{old}-\alpha_j^{new})$$

最后更新b

$$\begin{aligned}b_i^{new}&=-E(x_i)-y_ik_{ii}(\alpha_i^{new}-\alpha_i^{odd})-y_jk_{ji}(\alpha_j^{new}-\alpha_j^{odd})+b^{odd}\\ b_j^{new}&=-E(x_i)-y_ik_{ii}(\alpha_i^{new}-\alpha_i^{old})-y_jk_{ji}(\alpha_j^{new}-\alpha_j^{old})+b^{odd}\\ b^{new}&=\frac{b_i^{new}+b_j^{new}}{2}\end{aligned}$$

选取变量的策略

第一个变量：选取违背KKT条件程度最大的变量

第二个变量：与第一个变量的间隔最大的变量

方法介绍

SVM1：梯度下降法

核心即为计算梯度，更新w，计算损失

计算梯度

```python
    def compute_gradient(self, X, y):

        distances = 1 - y * (np.dot(X, self.w))
        distances[distances < 0] = 0
        distances[distances > 0] = 1
        dw = self.w - self.C * (y.T @ (distances * X)).T
        return dw
```

计算损失

```python
    def compute_loss(self, X, y):

        distances = 1 - y * (np.dot(X, self.w))
        distances[distances < 0] = 0  # Max(0, distances)
        hinge_loss = self.C * np.sum(distances)
        return 0.5 * np.dot(self.w.T, self.w) + hinge_loss
```

主体循环

```
        for _ in range(self.max_iter):
            dw = self.compute_gradient(X, y)
            self.w -= self.lr * dw
            # self.b -= self.lr * db
            loss = self.compute_loss(X, y)[0][0]
            if _ > 0 and np.abs(loss - prev_loss) < self.tol:
                break
            prev_loss = loss
            loss_t.append(loss)
```

SVM2：序列最小化SMO

计算error，上式中的E_i，用于计算 α

```
    def compute_error(self, X, y, i):
        f_xi = np.dot(self.w, X[i]) + self.b
        E_i = f_xi - y[i]
        w1 = (self.w).reshape(X.shape[1], 1)
        self.errors = (X @ w1) - y
        self.errors += self.b
        return E_i
```

选择第二个变量α_j：与第一个变量α_i的间隔最大的变量

```
    def select_j(self, i, n_samples):
        gap = self.errors - self.errors[i][0]
        gap = np.abs(gap)
        # j = np.argmax(gap, axis=0)[0]
        m = np.max(gap)
        index = np.where(m == gap)
        rand = np.random.randint(0, len(index[0]))
        j = index[0][rand]
```

内层循环，用来选择并更新两个变量α_i α_j，设置上下界，以及更新b

```
            for i in range(n_samples):
                E_i = self.compute_error(X, y, i)
                if (y[i] * E_i < -self.epsilon and self.alpha[i] < self.C) or
(y[i] * E_i > self.epsilon and self.alpha[i] > 0):
                    # 这里的i和j与上面公式中的i和j反过来了，先选i再选与i误差最大的j
                    j = self.select_j(i, n_samples)
                    E_j = self.compute_error(X, y, j)

                    alpha_i_old, alpha_j_old = self.alpha[i], self.alpha[j]

                    if y[i] != y[j]:
                        L = max(0, alpha_j_old - alpha_i_old)
                        H = min(self.C, self.C + alpha_j_old - alpha_i_old)
                    else:
                        L = max(0, alpha_i_old + alpha_j_old - self.C)
                        H = min(self.C, alpha_i_old + alpha_j_old)
                    if L == H:
```

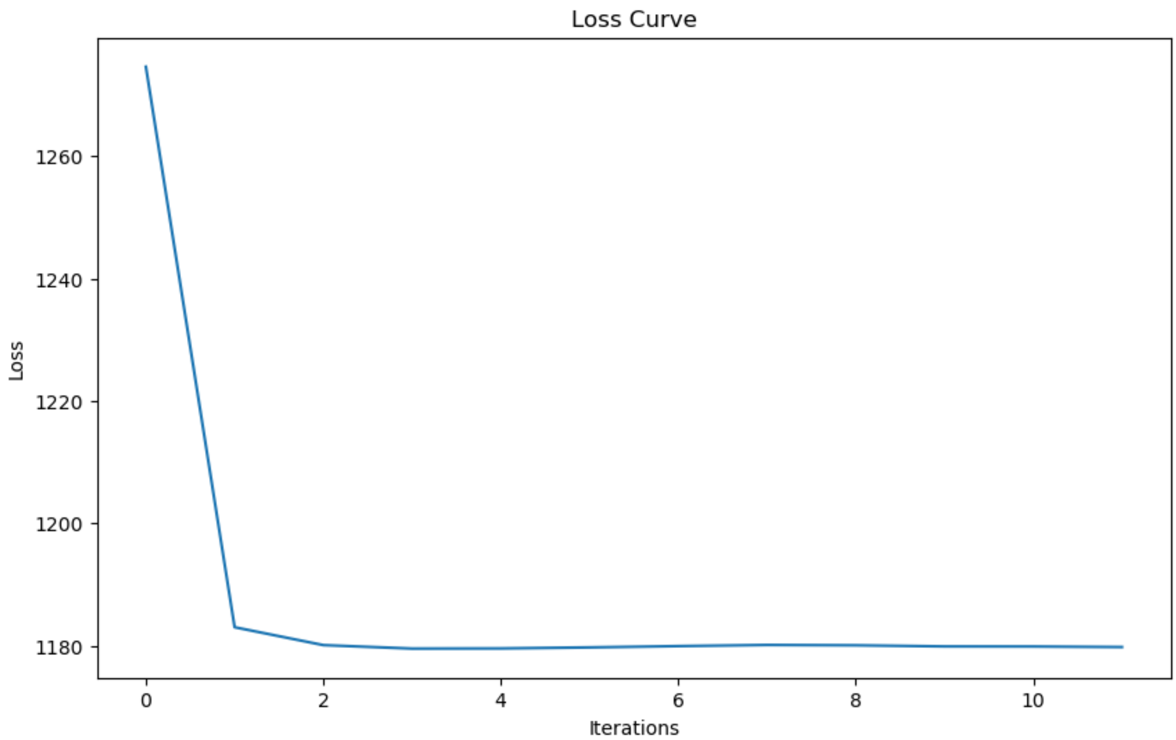
```
                        continue

                    eta = 2 * self.K[i, j] - self.K[i, i] - self.K[j, j]
                    if eta >= 0:
                        continue

                    self.alpha[j] -= y[j] * (E_i - E_j) / eta
                    self.alpha[j] = max(L, min(H, self.alpha[j]))
                    if np.abs(self.alpha[j] - alpha_j_old) < self.epsilon:
                        continue

                    self.alpha[i] += y[i] * y[j] * (alpha_j_old - self.alpha[j])

                    b1 = self.b - E_i - y[i] * (self.alpha[i] - alpha_i_old) * self.K[i, i] - y[j] * (self.alpha[j] - alpha_j_old) * self.K[i, j]
                    b2 = self.b - E_j - y[i] * (self.alpha[i] - alpha_i_old) * self.K[i, j] - y[j] * (self.alpha[j] - alpha_j_old) * self.K[j, j]

                    if 0 < self.alpha[i] < self.C:
                        self.b = b1
                    elif 0 < self.alpha[j] < self.C:
                        self.b = b2
                    else:
                        self.b = (b1 + b2) / 2
```

实验结果

SVM1：梯度下降法

20维10000样本数据集，训练集：测试集 = 3：1，主要参数lr = 0.00001, max_iter = 1000, tol = 1e-2

错标率mislabel = 0.0349， 准确率 = 0.9592，训练时间0.2s



可以看到损失函数很快就收敛

SVM2：序列最小化SMO

由于SMO算法用循环实现，且启发式算法寻找变量较为耗时，所以采用较小的样本来计算

10维1000样本数据集，训练集：测试集 = 3：1，主要参数max_iter=10000, epsilon=0.025

错标率mislabel = 0.022， 准确率 = 0.9080，训练时间 41.8s

方法比较

单次验证

样本大小	mislabel	梯度下降准确率	梯度下降用时	SMO准确率	SMO用时	sklearn准确率	sklearn用时
10 * 1000	0.022	0.9680	0.0130s	0.9080	21.59s	0.9760	0.0010s
10 * 500	0.04	0.9360	0.0070s	0.9120	10.099s	0.9280	0.0010s
10 * 200	0.05	0.9400	0.01000s	0.8800	0.0210s	0.9200	0.0s

四折交叉验证法 10维1000样本

mislabel = 0.029

模型	准确率
sklearn	0.9528
软间隔支持向量机 梯度下降法	0.9548
软间隔支持向量机 SMO	0.9455

可以看出，梯度下降法在时间上和准确率方面都优于SMO，但是与sklearn的准确率相差并不大

SMO中的α的维数跟样本数有关，在样本数过大时，需要迭代的次数过多，耗时太久，所以这里仅比较了10*1000大小的数据集