Machine Learning Lab2

葛凇铂 PB20061376

2022.10.18

实验要求

1. 独立编写 SVM 解决二分类问题
2. 采用两种不同方法解决二次规划问题（禁止使用 sklearn 或其他机器学习库）
3. 测试代码效率并与标准库进行比较

参考曾在课外完成的 cs231n-assignment1

实验原理

SVM

线性 SVM

- 硬边距：
  给定数据和标签：X, y，硬边界 SVM 是在线性可分问题中求解最大边距超平面的算法，约束条件是样本点到决策边界的距离大于等于 1. 硬边距 SVM 可以转化为一个等价的二次凸优化问题进行求解

$$\min_{w,b} \frac{1}{2}\|w\|^2$$

$$s.t. \quad y_i(w^T X_i + b) \geq 1$$

- 软边距：
  在线性不可分问题中使用硬边距 SVM 将产生分类误差，因此可在最大化边距的基础上引入损失函数构造新的优化问题，当使用松弛变量处理损失函数时，软边距 SVM 的优化问题为

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} \xi_i$$

$$s.t. \quad y_i(w^T X_i + b) \geq 1 - L_i, L_i \geq 0$$

SMO

SMO 是一种坐标下降法，以迭代方式求解 SVM 的对偶问题，其设计是在每个迭代步选择拉格朗日乘子中的两个变量 α_i, α_j 并固定其它参数，将原优化问题化简至 1 维子可行域，此时约束条件有如下等价形式

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \Longleftrightarrow \alpha_i y_i + \alpha_j y_j = -\sum_{k\neq i,j} \alpha_k y_k = const$$

SMO 的计算框架：

1. 初始所有化拉格朗日乘子
2. 识别一个不满足 KKT 条件的乘子，并求解其二次规划问题
3. 反复执行上述步骤直到所有乘子满足 KKT 条件或参数的更新量小于设定值

SGD

从损失函数优化的角度来解决这个问题，我们构建一种 hinge 损失函数

$$L(w,b) = \sum_i \max(0, 1 - y_i(wx_i + b))$$

正确分类并且置信度 $y_i(wx_i + b)$ 大于 1 的样本点对损失函数的贡献为 0，而其他样本点会对损失函数产生影响，从而在优化损失函数的时候有针对

地调整。对于这样在开放空间上的参数解，我们通常都会增加一个正则项，以降低最终参数的复杂度。那么最终使用的损失函数

$$L(w,b) = \sum_i \max(0, 1 - y_i(wx_i + b)) + \frac{1}{C}\|w\|^2$$

实现细节

(SVM1)SVM-SGD

class SVM1

- self.w：权重矩阵
- self.loss：hinge 损失
- self.dw：权重梯度

```
    def __init__(self):
        self.w = None
        self.loss = None
        self.dw = None
```

svm_loss

计算 hinge 损失与梯度

```
    def svm_loss(self, w, x, y, reg):
        loss = 0.0

        dw = np.zeros(w.shape)
        num_train = x.shape[0]
        scores = x.dot(w)
        #print(scores[range(num_train), y[0]])
        correct_class_scores = scores[ np.arange(num_train),
            list(y)].reshape(num_train,1)
        margin = np.maximum(0, scores - correct_class_scores +1)
```

```python
        margin[range(num_train), list(y)] = 0
        data_loss = np.sum(margin)*1.0 / num_train
        reg_loss = reg * np.sum(np.square(w))
        loss = data_loss + reg_loss

        x_effect = (margin > 0).astype('float')
        x_effect[range(num_train), y] -= np.sum(x_effect, axis=1)
        dw = x.T.dot(x_effect)
        dw /= num_train
        dw += 2*reg*w

        return loss, dw
```

fit

使用 SGD 算法训练模型，考虑到数据集参数过大，可采用每次训练抽取其中一部分数据进行梯度下降的方式。

- x：训练数据参数
- y：训练数据标签
- batch_size：随机抽取的数据数量
- reg：正则化参数
- learning_rate：学习率
- num_iters：最大训练次数

```python
    def fit(self, x, y, learning_rate, reg, num_iters, batch_size, verbose):
        """
        Fit the coefficients via your methods
        """
        y = y.astype('int')
```

```
        y[:] = (y[:] - np.min(y[:])) / (np.max(y[:]) - np.min(y[:]))
        for i in range(5):
            print(y[i])
        num_classes = 2
        num_train = x.shape[0]
        if self.w is None:
            self.w = 0.001 * np.random.randn(x.shape[1], num_classes)

        loss_history = []
        for it in range(num_iters):
            x_batch = None
            y_batch = None
            batch_idx = np.random.choice(num_train, size=batch_size, replace=False)
            x_batch = x[batch_idx]
            y_batch = y[batch_idx]
            #w_batch = self.w[batch_idx]
            #self.w = self.w
            #self.w = np.array(self.w).reshape(num_train, num_classes)
            loss, dw = self.svm_loss(self.w, x_batch, y_batch, reg)

            loss_history.append(loss)
            self.w -= learning_rate * dw
            if verbose and it % 100 == 0:
                print('iteration %d / %d  loss : ' % (it, num_iters))
                print(loss)
            if loss < 0.2:
                break
        return loss_history
```

predict

使用训练得到的模型进行预测

- x：待预测数据

```
def predict(self, X):
    """
    Use the trained model to generate prediction probabilities on a neself.w
    collection of data points.
    """
    y_pred = np.zeros(X.shape)
    y_pred = np.argmax(X.dot(self.w), axis=1)
    return y_pred
```

(SVM2)SVM-SMO

class SVM2

- self.w：权重矩阵
- self.c：正则化参数
- self.alpha：拉格朗日乘子
- self.b：常数 b
- self.loss：损失

```
def __init__(self):
    self.w = None
    self.c = None
    self.alpha = None
    self.b = 0
    self.loss = []
```

clip

对 α 进行剪枝，使其值在 H,L 之间

```
def clip(self, alpha, L, H):

    if alpha < L:
        return L
    elif alpha > H:
        return H
    else:
        return alpha
```

select_j

采用随机选择的方式选择参数 α_j

```
def select_j(self, i, m):
    l = list(range(m))
    seq = l[: i] + l[i+1:]
    return random.choice(seq)
```

f

根据线性核函数计算预测值

```
def f(self, x_i, x, y):
    x_i = np.array(x_i).T
    data = np.array(x)
    ks = np.dot(data, x_i)
    wx = np.dot((self.alpha*y), ks)
    fx = wx + self.b
    return fx
```

fit

使用 SMO 算法训练模型，由于随机选择 α_j 的缘故，训练速度很慢

- x：训练数据参数
- y：训练数据标签
- C：软间隔常数
- max_iter：最大训练次数

```
    def fit(self, x, y, C, max_iter):
        x = np.array(x)
        m, n = x.shape
        y = np.array(y)
        self.alpha = np.zeros(m)
        self.b = 0
        self.c = C
        it = 0

        while it < max_iter:
            pair_changed = 0
            for i in range(m):
                a_i, x_i, y_i = self.alpha[i], x[i], y[i]
                fx_i = self.f(x_i, x, y)
                loss_i = fx_i - y_i
                j = self.select_j(i, m)
                a_j, x_j, y_j = self.alpha[j], x[j], y[j]
                fx_j = self.f(x_j, x, y)
                loss_j = fx_j - y_j
                K_ii, K_jj, K_ij = np.dot(x_i, x_i), np.dot(x_j,
                    x_j), np.dot(x_i, x_j)
                eta = K_ii + K_jj - 2*K_ij
                if eta <= 0:
                    continue
                a_i_old, a_j_old = a_i, a_j
                a_j_new = a_j_old + y_j*(loss_i - loss_j)/eta
                if y_i != y_j:
```

```
                    L = max(0, a_j_old - a_i_old)
                    H = min(self.c, self.c + a_j_old - a_i_old)
                else:
                    L = max(0, a_i_old + a_j_old - self.c)
                    H = min(self.c, a_j_old + a_i_old)
                a_j_new = self.clip(a_j_new, L, H)
                a_i_new = a_i_old + y_i*y_j*(a_j_old - a_j_new)
                if abs(a_j_new - a_j_old) < 0.00001:
                    continue
                self.alpha[i], self.alpha[j] = a_i_new, a_j_new
                b_i = -loss_i - y_i*K_ii*(a_i_new - a_i_old) - y_j*K_ij*(a_j_new - a_j_old) + self.b
                b_j = -loss_j - y_i*K_ij*(a_i_new - a_i_old) - y_j*K_jj*(a_j_new - a_j_old) + self.b
                if 0 < a_i_new < self.c:
                    self.b = b_i
                elif 0 < a_j_new < self.c:
                    self.b = b_j
                else:
                    self.b = (b_i + b_j)/2
                pair_changed += 1
                loss = 0
                #for i in range(m):
                #    loss += abs(self.f(x[i], x, y) - y[i])
                #self.loss.append(loss / m)

            if pair_changed <= 5:
                it += 1
            else:
                it = 0
            print('iteration:{}  pair_changed:{}'.format(it, pair_changed))
            self.get_w(x, y)
        return self.loss
```

predict

使用训练得到的模型进行预测

- x：待预测数据

```
    def predict(self, x):
        y_pred = np.zeros(x.shape)
        y_pred = x.dot(self.w) + self.b
        m = y_pred.shape[0]
        for i in range(m):
            if y_pred[i] >= 0:
                y_pred[i] = 1.0
            else:
                y_pred[i] = -1.0
        return y_pred
```

性能分析

SVM-SGD

参数信息

样本数量：10000，维度：20，错标率：0.0375

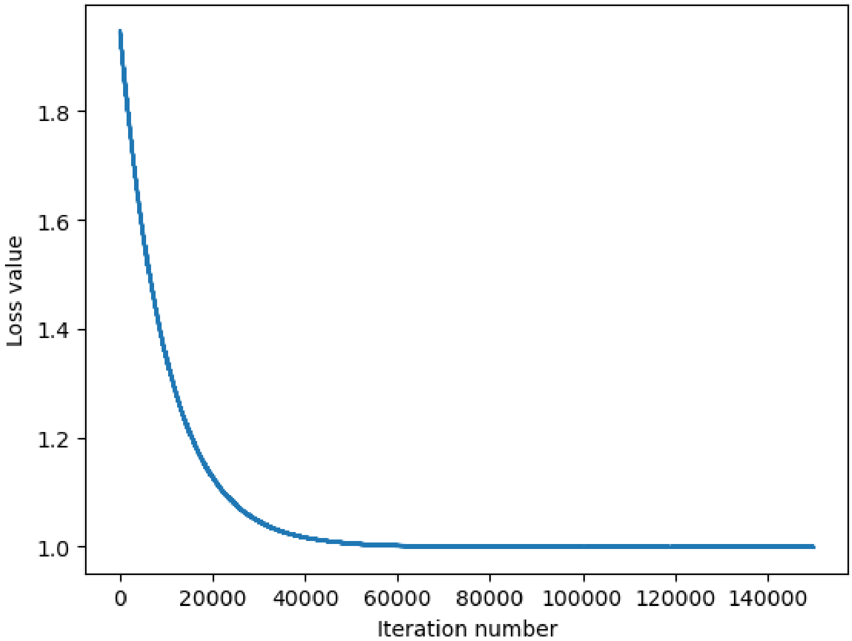
learning_rate：1e-9

reg：2.5e4

num_iters：150000

batch_size：500

loss 曲线



训练时间：**90.56s**
训练精确度：**0.945375**
预测精确度：**0.946000**

SVM-SMO

参数信息
样本数量：**500**，维度：**20**，错标率：**0.0375**
C：0.7
max_iter：40
训练时间：**667.76s**
训练精确度：**0.955000**
预测精确度：**0.930000**

参数信息
样本数量：**10000**，维度：**20**，错标率：**0.0375**

C：0.7
max_iter：40
训练时间（使用了云服务器）：12.5h
训练精确度：0.95374
预测精确度：0.94230

sklearn-SVM

参数信息
样本数量：10000，维度：20，错标率：0.0375
C：0.7
max_iter：40
训练时间：1.20s
训练精确度：0.96725
预测精确度：0.943

实验反思

采用随机选择 α_j 的方法代码过于繁琐，此外由于随机选择的原因，导致计算量过大，训练时间过长。事实上 SMO 算法应采用优化的内循环方法，结合 α_j 上次是否被替换过以及 α_j 是否为 0 考虑，由于个人能力原因未能完成内循环的 SMO 算法的实现。