机器学习大作业：从 0 实现 SVM 及其核函数的探究

521030910376 应之未

本次大作业的代码可以在本链接中找到。

1. 问题背景及任务描述

SVM 是最经典的一类机器学习的分类算法，其基本想法是通过给数据集划分最大化 Margin 的超平面从而实现分类目的，在此基础上衍生出了 Soft-SVM 等诸多针对更复杂的分类问题的进阶算法。本次大作业要求不使用现有的机器学习库函数，从 0 实现 SVM 的训练与测试，并在 MNIST，CIFAR10 数据集进行实验（且不允许使用神经网络对图片特征进行预处理）。

本实验报告主要包含以下内容：

- SVM 原理
- SMO 算法
- SVM 解决多分类问题
- 代码实现及测试
- 改进过程及方式
- 总结

2. SVM 原理

我们首先假设高维空间中的样本点可以被一个线性超平面划分成两类，不妨假设在此超平面上面的样本点标签为 1，在此超平面下面的样本点标签为 -1。这个超平面可以由平面的法向量 $\boldsymbol{w}$ 和位移量 $\boldsymbol{b}$ 来唯一表示，

$$\boldsymbol{w}^T\boldsymbol{x}+b=0$$

两个和此超平面平行的边缘分界面（离此超平面最近的点位于这两个边缘分界面上）可以表示成，

$$|\boldsymbol{w}^T\boldsymbol{x}+b|=d$$

由齐次性，我们可以不妨设 $d=1$，即两个超平面可以表示为，

$$\boldsymbol{w}^T\boldsymbol{x}+b=\pm 1$$

事实上，我们可以很容易地求得在两个边缘分界面上的任意一个点 p 到此超平面的距离 d，

$$d=\frac{|\boldsymbol{w}^T\boldsymbol{p}+b|}{||\boldsymbol{w}||}=\frac{1}{||\boldsymbol{w}||}$$

对于点 x_i，我们不妨设它的标签为 y_i。我们想要最大化 Margin$=2d$，可以表示为，

$$\max \frac{2}{||\boldsymbol{w}||} \quad \text{s.t. } \forall i. y_i(\boldsymbol{w}^T\boldsymbol{x}_i+b)\geq 1$$

可以将其转化为，

$$\min \frac{||\boldsymbol{w}||^2}{2} \quad \text{s.t. } \forall i. y_i(\boldsymbol{w}^T\boldsymbol{x}_i+b)\geq 1$$

该问题是一个凸优化问题，其 Lagrangian 为，

$$L(\boldsymbol{w},b,\boldsymbol{\alpha})=\frac{||\boldsymbol{w}||^2}{2}+\sum_i \alpha_i y_i(1-\boldsymbol{w}^T\boldsymbol{x}_i-b)$$

对 $L(\boldsymbol{w},b,\boldsymbol{\alpha})$ 求梯度并令梯度为 0，可得，

$$\boldsymbol{w}=\sum_i \alpha_i y_i \boldsymbol{x}_i$$

$$\sum_i \alpha_i y_i=0$$

于是，$L(\boldsymbol{w},b,\boldsymbol{\alpha})$ 可以重新写为，

$$L(\boldsymbol{w},b,\boldsymbol{\alpha})=-\frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j y_i y_j \boldsymbol{x}_i\cdot\boldsymbol{x}_j+\sum_i \alpha_i=L(\boldsymbol{\alpha})$$

原优化问题可以转化为，

$$\max L(\boldsymbol{\alpha}) \quad \text{s.t. } \forall i.\alpha_i\geq 0,\ \sum_i \alpha_i y_i=0$$

但实际上，样本点不一定线性可分。这时候，原始样本空间可能并不存在一个能正确划分两类样本的超平面，此时我们需要一个非线性变换 Φ，将数据从原先的特征空间映射到一个新的空间中，使得在这个新空间下我们能找到线性的超平面对样本点进行划分。此时，决策分界面可以由下式表示，

$$\boldsymbol{w}^T\Phi(\boldsymbol{x})+b=0$$

经过与线性可分时完全相同的数学推导后，最大化 Margin 可以转化为，

$$\max\ -\frac{1}{2}\sum_i\sum_j\alpha_i\alpha_j y_i y_j\Phi(\boldsymbol{x}_i)\cdot\Phi(\boldsymbol{x}_j)+\sum_i\alpha_i$$
$$\text{s.t. }\forall i.\alpha_i\geq 0,\ \sum_i\alpha_i y_i=0$$

于是我们就引出了另一个很重要的概念，核函数。假设有这样的核函数 $K(u,v)=\Phi(u)\cdot\Phi(v)$，即便我们不知道 u 和 v 具体的变换后的结果，但是我们实际上并不关心这个，我们只需要得到变换后结果的内积。

目前有一些经典且常用的核函数：

- 线性核函数：$K(u,v)=u\cdot v$
- 多项式核函数：$K(u,v)=(u\cdot v+\sigma)^d$
- 高斯核函数（RBF）：$K(u,v)=e^{-\frac{||u-v||^2}{2\sigma^2}}$
- 拉普拉斯核函数：$K(u,v)=e^{-\frac{||u-v||^2}{\sigma}}$
- Sigmoid 核函数：$K(u,v)=tanh(\beta u\cdot v+\theta)$

在引入核函数后，我们不再关心具体的映射方式，而只关心映射完之后两个样本点之间得到的结果。于是应用核函数后得到的问题目标为，

$$\max\ -\frac{1}{2}\sum_i\sum_j\alpha_i\alpha_j y_i y_j K(\boldsymbol{x}_i\ \ \boldsymbol{x}_j)+\sum_i\alpha_i$$
$$\text{s.t. }\forall i.\alpha_i\geq 0,\ \sum_i\alpha_i y_i=0$$

3. SMO 算法

在上面的推导中，我们将原来非常复杂的优化问题转化为了只含一个变量 $\boldsymbol{\alpha}$ 的条件极值问题。然而，$\boldsymbol{\alpha}$ 的维度往往高于 2，我们很难直接利用数学推导给出条件极值的解。SMO 算法给了计算机一种逐步逼近条件极值的方法，它的核心思想是在每次迭代中只优化两个变量，将其他变量视为常数，从而逐步进行优化直到基本达到条件极值。本节的书写主要参考了斯坦福大学 CS229 课程的一篇 Lecture Notes，可点击此链接访问原文。

SMO 算法的主要步骤如下：

(1) 计算误差:

$$E_i=(\sum_{j=1}^{N}\alpha_j y_j K(\boldsymbol{x}_j,\boldsymbol{x}_i)+b)-y_i,\ i=1,2$$

(2) 计算上下边界 L,H:

$$\begin{cases}L=\max(0,\alpha_2^{old}+\alpha_1^{old}-C)\\H=\min(C,\alpha_2^{old}+\alpha_1^{old})\end{cases},\ y_1=y_2$$

$$\begin{cases}L=\max(0,\alpha_2^{old}-\alpha_1^{old})\\H=\min(C,C+\alpha_2^{old}-\alpha_1^{old})\end{cases},\ y_1\neq y_2$$

(3) 计算 η:

$$\eta=K_{11}+K_{22}-2K_{12}$$

(4) 更新 α_2:

$$\alpha_2^{new,unc}=\alpha_2^{old}+\frac{y_2(E_1-E_2)}{\eta}$$

(5) 根据取值范围修剪 α_2:

$$\alpha_2^{new}=\begin{cases}H,\ \alpha_2^{new,unc}\geq H\\\alpha_2^{new,unc},\ L<\alpha_2^{new,unc}<H\\L,\ \alpha_2^{new,unc}\leq L\end{cases}$$

(6) 更新 α_1:

$$\alpha_1^{new}=\alpha_1^{old}+y_1y_2(\alpha_2^{old}-\alpha_2^{new})$$

(7) 更新 $b_1^{new}, b_2^{new}, b^{new}$:

$$\begin{aligned} b_1^{new} =& -E_1 - y_1K_{11}(\alpha_1^{new} - \alpha_1^{old}) \\ & - y_2K_{21}(\alpha_2^{new} - \alpha_2^{old}) + b^{old} \\ b_2^{new} =& -E_2 - y_1K_{22}(\alpha_1^{new} - \alpha_1^{old}) \\ & - y_2K_{22}(\alpha_2^{new} - \alpha_2^{old}) + b^{old} \\ b^{new} =& \frac{b_1^{new} + b_2^{new}}{2} \end{aligned}$$

4. SVM 解决多分类问题

简单 SVM 由于是对样本空间划分超平面，因此只能解决简单的二分类问题，对于多分类问题，我的处理方式也非常朴素，就是对任何两个训练一个 SVM，这样就可以得到 45 个二分类器。对于每个数据，我们都分别通过这 45 个二分类器去验证这个数据属于这个二分类器所对应的哪个类，并统计该数据被分到每一类的次数之和。由于对每个类别只有 9 个二分类器，因此这个次数之和不会超过 9。我们将最后被分到次数最多的那个类别就当做这个数据最终被多分类器分到的类别。

5. 代码实现及测试

5.1. 数据集生成

我将生成小型 MNIST 数据集和 CIFAR10 数据集的代码分别放在了”data_generate_m.py” 和”data_generate_c.py” 中。在此对生成方式做简单描述：首先从文件中读入原始 MNIST 和 CIFAR10 的训练集，然后使用 np.where() 函数统计各个类别的 index，分别取出各个类别的图像后对每个类别只保留前 500 张图像，然后重新整合并打乱生成训练集。测试集的生成方式与训练集同理，不再赘述。训练集数据生成对应的代码如下：

```
读取MNIST/CIFAR10数据集的训练集图像存入x_train中
读取MNIST/CIFAR10数据集的训练集标签存入y_train中
labels0 = np.where(y_train == 0)
images0 = x_train[labels0]
…… …… …… …… …… …… …… …… …… …… …… …… …… …… …… …… ……
labels9 = np.where(y_train == 9)
images9 = x_train[labels9]
imagesf = np.concatenate((images0[0:500],…… ,
    images9[0:500]),axis=0)
labelsf = np.concatenate((np.zeros(500),…… ,np.
    zeros(500)+9))
randomize = np.arange(5000)
np.random.shuffle(randomize)
imagesf = imagesf[randomize]
labelsf = labelsf[randomize]
np.save('images_c_train.npy',imagesf)
np.save('labels_c_train.npy',labelsf)
```

5.2. 二分类 SVM 与多核的改进

我将实现二分类的 SVM 类的代码放在了”SVM2.py” 中。多核的改进代码实现我放在了 SVM2 类的初始化函数中，根据传入的核函数名字（”linear””rbf””mix”）来计算对应的结果。对应的代码如下：

```
for i in range(self.x_axis):
    for j in range(self.x_axis):
        if self.name == 'linear':
            self.Kernal_Mat[i,j] = sum(self.data[i
    ,:] * self.data[j,:])
        elif self.name == 'rbf':
            self.Kernal_Mat[i,j] = np.exp(sum((
    self.data[i,:] - self.data[j,:]) * (self.data
    [i,:] - self.data[j,:]))/(-1 * self.theta**2)
    )
        elif self.name == 'mix':
            self.Kernal_Mat[i,j] = self.
    coefficient * sum(self.data[i,:] * self.data[
    j,:]) + (1 - self.coefficient) * np.exp(sum((
    self.data[i,:] - self.data[j,:]) * (self.data
    [i,:] - self.data[j,:]))/(-1 * self.theta**2)
    )
```

我选用的多核组合方式为线性组合，输入组合系数为 coefficient，输出为对应比例的线性核函数和 rbf 核函数的线性组合。

5.3. 多分类 SVM

多分类 SVM 本质上就是训练 (类数 *(类数-1))/2 个二分类 SVM，实现的过程我放在了主程序”SVM.py” 中。对应的代码如下：

```
for i in range(10):
    for j in range(i + 1, 10):
        data = [dataSet[i], dataSet[j]]
        label = [1.0] * 500
        label.extend([-1.0] * 500)
        svm = SVM2(np.array(data), np.array(label)
    , C, toler, maxIter, name, theta, coefficient
    )
        svm.fit()
```

```
        folder.append(svm) # folder用于保存每个训练好的SVM
```

Listing 1: 多个二分类器的训练

```
for i in range(10):
    for j in range(i + 1, 10):
        s = folder[index]
        t = s.predict([test_img[k]])[0]
        if t > 0.0:
            result[i] += 1
        else:
            result[j] += 1
        index += 1
label = result.index(max(result)))
```

Listing 2: 多个二分类器组合成一个多分类器

5.4. 测试结果

在 MNIST 数据集上，该模型在测试集上的准确率为 0.8828；在 CIFAR10 数据集上，该模型在测试集上的准确率为 0.4043。可以看到，在 MNIST 数据集上，该模型取得了较好的效果，与简易的深度学习算法效果接近，而在 CIFAR10 数据集上，虽然该模型学习到了一定的特征，有一定的效果，但效果并不尽如人意。

6. 改进过程和方式

6.1. 时间复杂度的改进

在第一次实现二分类 SVM 的时候，我在每次计算的时候都需要重新计算两个数据的核函数的结果，在网上相关资料的启发下，我在初始化这个二分类器的时候就先计算出一个核矩阵，存储每两个数据的核函数的结果，这样大大减少了计算量，每一个二分类运行时间从原来的 2 分钟左右缩短到原来的一半，改进后一次训练的总时长在 50 分钟左右，测试一轮的时间在 3 分钟左右。

6.2. 分类精度的改进 · 参数消融实验

由于在 CIFAR10 上训练的效率不高，参数消融实验我只在 MNIST 数据集上进行。我一共进行了九组不同参数的实验，对于线性核、rbf 核和多核分别进行了三组实验。由于列宽限制，我将用 T 表示参数 toler，I 表示参数 max_iter。

表 1: 线性核 *(I=500)*

C	T	Acc
200	0.0001	0.525
200	0.001	0.488
300	0.001	0.297

表 2: *rbf* 核 *(C=200, T=0.0001)*

I	θ	Acc
2000	20	0.873
500	20	0.762
500	10	0.539

表 3: *mix*核 *(C=200, T=0.001, I=500)*

θ	coefficient	Acc
20	0.2	0.785
20	0.5	0.598
20	0.8	0.544

通过实验可以明显看出，rbf 核的效果明显好于线性核，而 mix 核的效果并没有和预期的一致比 rbf 核效果好。可能是因为在此问题中，线性核并不能达到很好的效果，因此将线性核和 rbf 核做加权可能反而导致更差的结果。由于时间有限，并没有尝试在第二节 SVM 原理中提到的拉普拉斯核和 Sigmoid 核，但是如果能够找到一个不同种类的核函数更好地刻画数据的分布，从理论上说可以取得更好的效果。

6.3. 简单特征的提取

查阅资料并与同学们讨论后得知，如果对 CIFAR10 数据集进行简单特征的提取，例如提取 HOG 特征突出刻画照片的形状特征，能取得较好的效果。由于时间有限，对于此部分只停留在理论阶段，并没有真正编程验证。