Assignment 6

黎郡 2020E8017782051

Part 1

1. **有N个样本$x_1,\dots,X_N$,每个样本维数D，希望将样本位数降到K，请给出PCA算法的计算过程。**

   - 第一步：计算样本的均值

   $$\bar{x}=\frac{1}{N}\sum_{n=1}^{N}x_n,\bar{x}\in R^{D\times 1}$$

   - 第二步：计算样本的协方差矩阵

   $$S=\frac{1}{N}\sum_{n=1}^{N}(x_n-\bar{x})(x_n-\bar{x})^T,S\in R^{D\times D}$$

   - 第三步：对$D\times D$的矩阵 S 进行特征值分解求出特征向量μ和特征值λ
   - 第四步：将特征值从大到小排序，取前 K 个特征值$\mu_1,\dots.,\mu_k$ (s.t. $\lambda_1\geq\lambda_2\geq\dots.\lambda_{k-1}\geq\lambda_K$)
   - 第五步：将这些特征值组合成$D\times K$的投影矩阵$U=[\mu_1,\mu_2,\dots,\mu_k]$
   - 第六步：对每个样本x_n进行投影从而达到降维

   $$z_n=U^Tx_n,z_n\text{是一个}K\times 1\text{维的向量}$$

2. **根据自己的理解简述结构风险最小化与VC维。**

   - 结构风险最小化：

     结构风险最小化是指我们希望在测试集上有非常小的错误率。在训练集上最小化错误率叫做经验风险最小化。所以结构风险最小化就是指保证在经验风险最小化的同时，降低模型的复杂度，让模型在测试集上获得较低的错误率。结构风险小的模型往往泛化能力强，对训练数据以及未知的数据均能较好的预测。

   - VC维

     VC维指的是模型所能完全正确分类的最大样本数。K为超平面上VC为是K+1。VC维不一定反应参数量。一般情况下，VC维越高，模型越复杂。不过对于大多数分类器而言VC维难以定性计算。

3. **请推导出Hard-Margin SVM的优化目标。**

   - 假设训练集$D=(x_1,y_1),(x_2,y_2),\dots,(x_n,y_n),y_i\in\{1,-1\}$是线性可分的。当 所有的样本点到分类器的分界面的最小距离最大时，这时的分界面是最鲁棒的， 这就是 Hard-SVM 的优化目标。
   - 假设分界面为$WX+b=0$，则最大化margin即最小化样本点到分类器分界面的距离

   $$\text{margin}=\arg\min_{x\in D}d(x)=\arg\min_{x\in D}\frac{|X\cdot W+b|}{\sqrt{\sum_{i=1}^{d}w_i^2}}=\arg\min_{x\in D}\frac{|X\cdot W+b|}{\|W\|}$$

- 目标函数可以表示为，使得所有的样本点到分类器的分界面的最小距离 (margin) 最大:

$$\arg\max_{w,b} \text{margin}(\text{w}, \text{b}, \text{D})$$

$$\arg\max_{\omega,b} \arg\min_{x_i \in D} \frac{|\omega^T x_i + b|}{\|\omega\|}$$

- 为保证分界面可以将两类数据分开，即分界面在满足最小距离最大化时，还需对两类数据可分。即分界面要位于两类数据之间，上述目标函数需满足以下约束条件:

$$\text{s.t. } y_i \left(\omega^T x_i + b\right) \geq 0$$

- 假设 $|\omega^T x_i + b| \geq 1, \forall x_i \in D$, 此时

$$\arg\min_{x_i \in D} \frac{|\omega^T x_i + b|}{\|\omega\|} \geq \arg\min_{x_i \in D} \frac{1}{\|\omega\|} = \frac{1}{\|\omega\|}$$

- 原问题可以化简为:

$$\arg\max_{\omega,b} \frac{1}{\|\omega\|}$$

$$\text{s.t. } y_i \left(\omega^T x_i + b\right) \geq 1$$

- 为了便于求解，继续化为一个二次规划问题，即:

$$\arg\min_{\omega,b} \|\omega\|^2$$

$$\text{s.t. } y_i \left(\omega^T x_i + b\right) \geq 1$$

上式即为 Hard-Margin SVM 的优化目标与约束条件。

4. **请解释出Hinge Loss在SVM中的意义。**

Hinge Loss是在soft margin的情况下进行推到的。对于线性不可分的样本，Hard-Margin将不再使用。我们需要引入松弛来进行一个权衡(trade-off)。因此在Hard-Margin上引入松弛变量即变为Soft-Margin

$$\arg\min_{\omega,b} \|\omega\|^2 + c\sum_{i=1}^{n} \varepsilon_i$$

$$\text{s.t. } y_i \left(\omega^T x_i + b\right) \geq 1 - \varepsilon_i, \varepsilon_i \geq 0, i = 1, 2, \ldots, n$$

对分类结果进行分析:

- 在分类正确的情况下，$y_i \left(\omega^T x_i + b\right) \geq 1$, 则 $y_i \left(\omega^T x_i + b\right) \geq 1 - \varepsilon_i$ 成立，要使 $\min_{w,b} \varepsilon_i$ 成立,此时只需要 $\varepsilon_i = 0$ 。
- 在分类错误的情况下，$y_i \left(\omega^T x_i + b\right) < 1$, 要使 $\min_{\omega,b} \varepsilon_i$ 成立, 只需要 $\varepsilon_i = 1 - y_i \left(\omega^T x_i + b\right)$ ，即 ε_i等于1减其本身。

$$\min_{\omega,b} \varepsilon_i = \max\left\{0, 1 - y_i \left(\omega^T x_i + b\right)\right\}$$

至此得出了 Hinge Loss 形式:

$$\begin{aligned} hinge(y, \hat{y}) &= max(0, 1 - y\hat{y}) \\ hine(x) &= max(1 - x) \end{aligned}$$

所以目标函数变为:

$$\arg\min_{\omega,b} \|\omega\|^2 + c\sum_{i=1}^{n} hinge\left(y_i, \omega^T x_i + b\right)$$

Hinge Loss 的的意义:

- Hinge Loss是 0-1损失的上界，把Hinge Loss最小化了就意味着最小化0-1损失。
- Hinge Loss是一个凸函数，且分段线性，利于计算。
- Hinge Loss由Soft Margin 推导而来，等于在SVM中添加软约束，允许错分样本，让模型的泛化能力更强，模型更鲁棒。
- 保持了 SVM 的稀疏性。Hinge Loss 的零区域对应的是非支持向量的普通样本，这些样本都不参与最终超平面的决定，从而对训练样本的依赖大大减少，提高了训练效率。

5. **简述核方法的基本原理。**

> 核方法的基本思想是将低维线性不可分的数据，映射到高维变成线性可分的数据。在高维空间中学习线性支持向量机。在线性支持向量机的对偶问题中，把低维到高维的非线性变化的内积形式用核函数进行代替。

Part 2

从MNIST数据集中任意选择两类，对其进行SVM分类，可调用现有的SVM工具如LIBSVM，展示超参数C以及核函数参数的选择过程。

实现思路

数据集获取

> - 从官网下载Minist数据集：http://yann.lecun.com/exdb/mnist/
> - Minst数据集有四个文件，分别为：
>   - train-images-idx3-ubyte: training set images
>   - train-labels-idx1-ubyte: training set labels
>   - t10k-images-idx3-ubyte: test set images
>   - t10k-labels-idx1-ubyte: test set labels
>
>   训练集有60000个样本，测试机有10000个样本

数据集的理解

- MNIST Handwritten Digits字符库中含有数字0-9的训练数据集和数字0-9测试数据集两种图片，**每张图片都是灰度图，位深度为8。**数据集中图像的灰度值为0-255，可以先进行归一化再训练
- 数据是以二进制存储的，我们**读取的时候要以'rb'方式读取**，其次，**真正的数据只有[value]这一项，其他的[type]等**只是来描述的，并不真正在数据文件里面，这样写只是在向我们解释文件的结构

```
[offset] [type]          [value]          [description]
0000     32 bit integer  0x00000803(2051) magic number
0004     32 bit integer  60000            number of images
0008     32 bit integer  28               number of rows
0012     32 bit integer  28               number of columns
0016     unsigned byte   ??               pixel
0017     unsigned byte   ??               pixel
........
xxxx     unsigned byte   ??               pixel
Pixels are organized row-wise. Pixel values are 0 to 255. 0 means background
(white), 255 means foreground (black).
```

- 由offset我们可以看出真正的pixel式从16开始的，所以在读取pixel之前我们要读取4个32 bit integer，也就是`magic number`, `number of images`, `number of rows`, `number of columns`

> **注意：magic number是用来对文件做校验的**
>
> 只有读取到的magic number是2051才是合法的MNIST图像数据，只有读取到的`magic number`是2049才是合法的MNIST标记数据。在这些头信息之后便是实际数据了

- **根据说明文件存储的信息是`row-wise`（行式）的**，即在存储时是按照行的顺序来存储图片的像素，即先将第一行的每个像素的值依次存储起来，然后是第二行，第三行，直到所有的图片存储完成，所以我们在读取数据还原图片时也应该是行式的。

实验结果

```
D:\Anaconda\python.exe D:/concentrated_teaching_period/Pa
--------------1: 不同的C对结果影响-----------------
C=0.0001, kernel=rbf, accuracy=0.5237655745269958
C=0.001, kernel=rbf, accuracy=0.9667743424088602
C=0.1, kernel=rbf, accuracy=0.9963082602676512
C=1, kernel=rbf, accuracy=0.997692662667282
C=5, kernel=rbf, accuracy=0.997692662667282
C=10, kernel=rbf, accuracy=0.997692662667282
C=100, kernel=rbf, accuracy=0.997692662667282
C=1000, kernel=rbf, accuracy=0.997692662667282
C=10000, kernel=rbf, accuracy=0.997692662667282
--------------2: 不同的kernel对结果影响 -----------------
C=0.5, kernel=linear, accuracy=0.9903091832025842
C=0.5, kernel=poly, accuracy=0.9967697277341947
C=0.5, kernel=rbf, accuracy=0.9963082602676512
C=0.5, kernel=sigmoid, accuracy=0.9547761882787263

Process finished with exit code 0
```

分析实验结果我们可以发现：

- C 太低会造成欠拟合，C 太高会造成过拟合，本例中欠拟合现象较为明显，过拟合不明显。
- 不同的核函数也会影响分类的结果，在固定c的情况下poly 核函数具有较好的性能，精度是最高的。sigmod核函数在本次分类中表现效果最差。

实验代码

数据处理部分代码

```
"""
    TRAINING SET IMAGE FILE (train-images-idx3-ubyte):
    [offset] [type]          [value]          [description]
    0000     32 bit integer  0x00000803(2051) magic number
    0004     32 bit integer  60000            number of images
    0008     32 bit integer  28               number of rows
    0012     32 bit integer  28               number of columns
    0016     unsigned byte   ??               pixel
    0017     unsigned byte   ??               pixel
    ........
    xxxx     unsigned byte   ??               pixel
    Pixels are organized row-wise. Pixel values are 0 to 255.
    0 means background (white), 255 means foreground (black).
"""


def load_imgs(path):
    """
    加载图像数据
    :param path: 图像数据文件路径
    :return: imgs: 返回图像数据(num, img.size) 每行为一个数据
    """
    with open(path, 'rb') as f:  # 'rb'以二进制读取文件
        #  读取image文件前4个整型数字
        #  unpack(fmt, string)按照给定的格式(fmt)解析字节流string，返回解析出来的tuple
        # >4I 大端模式 4个Int I: int 4*4=16
        # f.read(16)读取前16个字节的数据
        magic, num, rows, cols = struct.unpack('>4I', f.read(16))  # 读完后缓冲区少了这
16字节内容
        #  整个images数据大小为60000*28*28
```

```
        # imgs_size = num * rows *cols
        imgs = np.fromfile(f, dtype=np.uint8).reshape(num, rows * cols)  # nums *
img_size

    return imgs


"""
    TRAINING SET LABEL FILE (train-labels-idx1-ubyte):
    [offset] [type]          [value]          [description]
    0000     32 bit integer  0x00000801(2049) magic number (MSB first)
    0004     32 bit integer  60000            number of items
    0008     unsigned byte   ??               label
    0009     unsigned byte   ??               label
    ........
    xxxx     unsigned byte   ??               label
    The labels values are 0 to 9.
    """


def load_labels(path):
    """
    加载label数据
    :param path: Lable数据文件路径
    :return: labels 每行为一个label的标签(n,) n*1维np.array对象，n为图片数量
    """
    with open(path, 'rb') as f:
        magic, num = struct.unpack('>2I', f.read(8))
        #  读取label文件前2个整形数字，label的长度为num
        labels = np.fromfile(f, dtype=np.uint8)
    return labels


def generate_data(img_path, label_path, class_list):
    """
    生成指定类别的数据集
    :param img_path: 图像数据文件路径
    :param label_path: 类别标签数据路径
    :param class_list:  需要生成的类别号列表
    :return: [img_class, label_class]:
        img_class为图像数据数组,每行为一个图像的数据(n,img.size);
        label_class为标签数据数组,每行为一个标签的数据(n,)
    """

    # 加载图像数据和label
    imgs = load_imgs(img_path)
    labels = load_labels(label_path)

    # 选取特定类别生成所需要的数据
    img_class = []
    label_class = []
    for c in class_list:
        index = np.where(labels == c)  # 返回的是label为指定值的index，格式([index的
list],dtype)
        img_class.extend(imgs[index[0]])
        label_class.extend((labels[index[0]]))
    # 将数据进行归一化，即图像幅值变为0-1
    img_class = np.array(img_class) / 255
    label_class = np.array(label_class)

    return [img_class, label_class]
```

主函数代码

```
if __name__ == "__main__":

    # 文件路径
    TRAIN_IMG_PATH = "./MNIST_data/train-images.idx3-ubyte"
    TRAIN_LABEL_PATH = "./MNIST_data/train-labels.idx1-ubyte"
    TEST_IMG_PATH = "./MNIST_data/t10k-images.idx3-ubyte"
    TEST_LABEL_PATH = "./MNIST_data/t10k-labels.idx1-ubyte"
    # 训练&测试的指定类别
    img_class = [1, 2]
    # 数据加载
    train_data = generate_data(TRAIN_IMG_PATH, TRAIN_LABEL_PATH, img_class)
    test_data = generate_data(TEST_IMG_PATH, TEST_LABEL_PATH, img_class)

    # 1: 不同的超参数C对结果影响
    print('--------------1: 不同的C对结果影响-----------------')
    C = [0.0001, 0.001, 0.1, 1, 5, 10, 100, 1000, 10000]
    kernel = 'rbf'
    for i in C:
        s = svm.SVC(C=i, kernel=kernel)
        s.fit(train_data[0], train_data[1])
        y_pred = s.predict(test_data[0])
        print('C={}, kernel={}, accuracy={}'.format(i, kernel, s.score(test_data[0],
test_data[1])))

    # 2: 不同的kernel对结果影响
    print('--------------2: 不同的kernel对结果影响 -----------------')
    C = 0.5
    kernel = ['linear', 'poly', 'rbf', 'sigmoid']
    for i in kernel:
        s = svm.SVC(C=C, kernel=i)
        s.fit(train_data[0], train_data[1])
        y_pred = s.predict(test_data[0])
        print('C={}, kernel={}, accuracy={}'.format(C, i, s.score(test_data[0],
test_data[1])))
```