清华大学电子工程系

**媒体与认知** 课堂 2

2022-2023 学年春季学期

**作业 3**

Jiaji Liu 2023 年 5 月 5 日

# 理论部分

## 1 单选题（15 分）

### 1.1 C

### 1.2 D

### 1.3 D

### 1.4 D

### 1.5 C

## 2 计算题（15 分）

### 2.1 距离地球很远有一个双星系统，其中有两个星球 A 和 B，从地球观测时，星球 A 和 B 的位置重叠。已知星球 A 有 60% 的部分是海洋，其余是陆地，而星球 B 则全是陆地。某一时刻，观测到星球 A 或 B 的概率相同，假设此时观测到该星球上的陆地，计算该星球是星球 A 的概率。(提示：全概率公式 $P(Y)=\sum_{i=1}^{N}P(Y|X_i)P(X_i)$)

*解.* 设 $X$ 表示观测到陆地，$Y$ 表示该星球是星球 A，则 $\bar{X}$ 表示观测到海洋，$\bar{Y}$ 表示该星球是 B。则

$$P(Y)=P(\bar{Y})=0.5$$

$$P(Y|X)=\frac{P(X|Y)P(Y)}{P(X)}$$

其中，

$$P(X)=P(X|Y)P(Y)+P(X|\bar{Y})P(\bar{Y})=0.4*0.5+1*0.5=0.7$$

因此
$$P(Y|X) = \frac{0.4 * 0.5}{0.7} = \frac{2}{7}$$

□

### 2.2 给定 5 维空间中的 6 个样本点，写成如下的 6x5 维矩阵 $X$，其中每一行代表一个样本点

$$X = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ -3 & -3 & -3 & 0 & 0 \\ 2 & 2 & 2 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 \\ 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & -1 & -1 \end{bmatrix}$$

(i) 计算该数据集的均值;

(ii) 求解矩阵 $X$ 的 SVD 分解;(提示:矩阵 $X$ 的 SVD 分解的形式为 $\begin{bmatrix} a & 0 \\ -3a & 0 \\ 2a & 0 \\ 0 & b \\ 0 & -2b \\ 0 & b \end{bmatrix} \times \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} \times \begin{bmatrix} c & c & c & 0 & 0 \\ 0 & 0 & 0 & d & d \end{bmatrix}$)

(iii) 计算该数据集的第一主成分(即最大特征值对应的归一化特征向量)?

*解.* (i) $\mu = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

(ii) $X^T X = \begin{bmatrix} 14 & 14 & 14 & 0 & 0 \\ 14 & 14 & 14 & 0 & 0 \\ 14 & 14 & 14 & 0 & 0 \\ 0 & 0 & 0 & 6 & 6 \\ 0 & 0 & 0 & 6 & 6 \end{bmatrix}$,其非零特征值为 42、12,特征向量分别为 $\frac{1}{\sqrt{3}} \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \end{bmatrix}^T$ 和 $\frac{1}{\sqrt{2}} \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \end{bmatrix}^T$,

$$XX^T = \begin{bmatrix} 3 & -9 & 6 & 0 & 0 & 0 \\ -9 & 27 & -18 & 0 & 0 & 0 \\ 6 & -18 & 12 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & -4 & 2 \\ 0 & 0 & 0 & -4 & 8 & -4 \\ 0 & 0 & 0 & 2 & -4 & 2 \end{bmatrix}$$，其非零特征值为 42、12，特征向量分别为 $\frac{1}{\sqrt{14}}\begin{bmatrix} 1 & -3 & 2 & 0 & 0 & 0 \end{bmatrix}^T$ 和 $-\frac{1}{\sqrt{6}}\begin{bmatrix} 0 & 0 & 0 & 1 & -2 & 1 \end{bmatrix}^T$，故 X 的 SVD 分解为

$$X = \begin{bmatrix} \frac{1}{\sqrt{14}} & 0 \\ -\frac{3}{\sqrt{14}} & 0 \\ \frac{2}{\sqrt{14}} & 0 \\ 0 & -\frac{1}{\sqrt{6}} \\ 0 & \frac{2}{\sqrt{6}} \\ 0 & -\frac{1}{\sqrt{6}} \end{bmatrix} \times \begin{bmatrix} \sqrt{42} & 0 \\ 0 & 2\sqrt{3} \end{bmatrix} \times$$
$$\begin{bmatrix} \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

(iii) 由于均值 $\mu = 0$，因此协方差为 $X^TX$，其最大的特征值对应的归一化特征向量为第一主成分，也即 $\frac{1}{\sqrt{3}}\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \end{bmatrix}^T$

□

### 2.3 设有两类正态分布的样本集，第一类均值为 $\mu_1 = [1,2]^T$，第二类均值为 $\mu_2 = [-1,-2]^T$。两类样本集的协方差矩阵相等 $\Sigma_1 = \Sigma_2 = \Sigma = \begin{bmatrix} 3.0 & 1.0 \\ 1.0 & 2.0 \end{bmatrix}$，先验概率分别为 $p(\omega_1) = 0.6$，$p(\omega_2) = 0.4$。试计算分类界面，并对样本 $x = [0,0]^T$ 分类。

解. $g_i(x) = -\frac{1}{2}(x-\mu_i)^T\Sigma_i^{-1}(x-\mu_i) + \ln p(\omega_i) + \text{const}$，因此

$$g_1(x) = -\frac{1}{2}[x_1-1, x_2-2]\begin{bmatrix} 3.0 & 1.0 \\ 1.0 & 2.0 \end{bmatrix}^{-1}[x_1-1, x_2-2]^T + \ln 0.6 + \text{const} =$$
$$-0.2x_1^2 - 0.3x_2^2 + x_2 + 0.2x_1x_2 - 1 + \ln 0.6 + \text{const},$$

$g_2(x) = -\dfrac{1}{2}[x_1+1, x_2+2]\begin{bmatrix} 3.0 & 1.0 \\ 1.0 & 2.0 \end{bmatrix}^{-1}[x_1+1, x_2+2]^T + \ln 0.4 + \text{const} =$
$-0.2x_1^2 - 0.3x_2^2 - x_2 + 0.2x_1x_2 - 1 + \ln 0.4 + \text{const}$。
判决平面为 $g_1(x) - g_2(x) = 2x_2 - \ln\dfrac{2}{3} = 0$。
当 $2x_2 - \ln\dfrac{2}{3} > 0$ 时，$g_1 > g_2$，归为第一类，
$2x_2 - \ln\dfrac{2}{3} < 0$ 时，$g_1 < g_2$，归为第二类。
$x = [0, 0]^T$ 时，$g_1 > g_2$，因此将 $x$ 归为第一类。

□

## 2.4 使用 KMeans 算法对 2 维空间中 6 个点 $(0, 2)$，$(2, 0)$，$(2, 3)$，$(3, 2)$，$(4, 0)$，$(5, 4)$ 进行聚类，距离函数选择哈密顿距离 $d = |x_1 - x_2| + |y_1 - y_2|$。

(i) 起始聚类中心选择 $(2, 3)$ 和 $(4, 0)$，计算聚类中心；

(ii) 起始聚类中心选择 $(2, 0)$ 和 $(5, 4)$，计算聚类中心。

解. 记 $d_{ij}$ 为第 $i$ 个点到第 $j$ 个聚类中心的距离。

(i) $d_{11} = 3$，$d_{12} = 6$，$d_{21} = 3$，$d_{22} = 2$，$d_{31} = 0$，$d_{32} = 5$，$d_{41} = 2$，$d_{42} = 3$，$d_{51} = 5$，$d_{52} = 0$，$d_{61} = 4$，$d_{62} = 5$。
将样本分配到最邻近聚类，聚类到 $(2, 3)$ 附近的点为 $(0, 2)$，$(2, 3)$，$(3, 2)$，$(5, 4)$，聚类到 $(4, 0)$ 附近的点为 $(2, 0)$，$(4, 0)$，新的聚类中心为 $(2.5, 2.75)$，$(3, 0)$。
继续计算得到 $d_{11} = 3.25$，$d_{12} = 5$，$d_{21} = 3.25$，$d_{22} = 1$，$d_{31} = 0.75$，$d_{32} = 4$，$d_{41} = 1.25$，$d_{42} = 2$，$d_{51} = 4.25$，$d_{52} = 1$，$d_{61} = 3.75$，$d_{62} = 6$，聚类中心不再改变。
因此得到的聚类中心为 $(2.5, 2.75)$，$(3, 0)$。

(ii) $d_{11} = 4$，$d_{12} = 7$，$d_{21} = 0$，$d_{22} = 7$，$d_{31} = 3$，$d_{32} = 4$，$d_{41} = 3$，$d_{42} = 4$，$d_{51} = 2$，$d_{52} = 5$，$d_{61} = 7$，$d_{62} = 0$。
将样本分配到最邻近聚类，聚类到 $(2, 0)$ 附近的点为 $(0, 2)$，$(2, 0)$，$(2, 3)$，$(3, 2)$，$(4, 0)$，聚类到 $(5, 4)$ 附近的点为 $(5, 4)$，新的聚类中心为 $(2.2, 1.4)$，$(5, 4)$。
继续计算得到 $d_{11} = 2.8$，$d_{12} = 7$，$d_{21} = 1.6$，$d_{22} = 7$，$d_{31} = 1.8$，$d_{32} = 4$，$d_{41} = 1.4$，$d_{42} = 4$，$d_{51} = 3.2$，$d_{52} = 5$，$d_{61} = 5.4$，$d_{62} = 0$，聚类中心不再改变。
因此得到的聚类中心为 $(2.2, 1.4)$，$(5, 4)$。

□

### 2.5 使用 Lagrange 乘子法求 $x^2+2y^2$ 在 $x^2+y^2\leq 1$ 区域内的极值。

解. 用 Lagrange 乘子法来求函数 $f(x,y)=x^2+2y^2$ 在约束条件
$g(x,y)=x^2+y^2-1\leq 0$ 下的极值。
首先构建拉格朗日函数
$L(x,y,\lambda)=f(x,y)+\lambda g(x,y)=x^2+2y^2+\lambda(x^2+y^2-1)$。
求导数，得:
$\frac{\partial L}{\partial x}=2x+2\lambda x$
$\frac{\partial L}{\partial y}=4y+2\lambda y$
$\frac{\partial L}{\partial \lambda}=x^2+y^2-1$
首先考虑约束区域的内部，令前两个约束条件和 $\lambda$ 为 0，得
$x=y=0,\quad f(0,0)=0$; 再考虑约束区域的边界，令三个约束条件为 0，得
$2x+2\lambda x=0\Rightarrow x(1+\lambda)=0$
$4y+2\lambda y=0\Rightarrow y(2+\lambda)=0$
$x^2+y^2-1=0$
当 $\lambda=-1$ 时，得到 $x=\pm1,y=0,f(\pm1,0)=1$；
当 $\lambda\neq-1$ 时，可得 $x=0,y=\pm1,\lambda=-2,f(0,\pm1)=2$。
因此，函数 $f(x,y)=x^2+2y^2$ 在 $x^2+y^2\leq 1$ 区域内的极大值为 2，极小值为 0，分别在点 $(0,\pm1)$ 和 $(0,0)$ 取到。 □

# 编程部分

## 3 编程作业报告

### 3.1 实现 hinge loss 模拟支持向量机并运行自动评判程序

svm_hw.py 中的 Linear 类代码:

```
class Linear(torch.autograd.Function):
    @staticmethod
    def forward(ctx, x, W, b):
        output = torch.matmul(x, W.T) + b
        ctx.save_for_backward(x, W, b)
```

```
        return output
    @staticmethod
    def backward(ctx, grad_output):
        x, W, b = ctx.saved_tensors
        batch, channels = x.shape
        grad_W = torch.matmul(grad_output.T, x)
        grad_b = torch.sum(grad_output, 0)
        return None, grad_W, grad_b
```

Hinge 类代码:

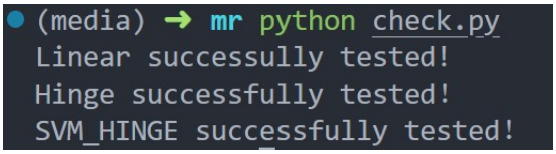
```
class Hinge(torch.autograd.Function):
    @staticmethod
    def forward(ctx, output, W, label, C):
        C = C.type_as(W)
        label = label.unsqueeze(1)
        loss = 0.5 * torch.norm(W, p=2) ** 2 + C * torch.sum(F.relu(1 -
            label * output))
        ctx.save_for_backward(output, W, label, C)
        return loss

    @staticmethod
    def backward(ctx, grad_loss):
        output, W, label, C = ctx.saved_tensors
        grad_output = -C * grad_loss * label * torch.gt(1 - output *
            label, 0)
        grad_W = grad_loss * W
        return grad_output, grad_W, None, None
```

SVM_HINGE 类代码:

```
class SVM_HINGE(nn.Module):
    def __init__(self, in_channels, C):
        super().__init__()
        self.W = nn.Parameter(torch.randn(1, in_channels),
            requires_grad=True)
        self.b = nn.Parameter(torch.randn(1), requires_grad=True)
        self.C = torch.tensor([[C]], requires_grad=False)
    def forward(self, x, label=None):
        output = Linear.apply(x, self.W, self.b)
        if label is not None:
            loss = Hinge.apply(output, self.W, label, self.C)
        else:
```

```
            loss = None
        output = (output > 0.0).type_as(x) * 2.0 - 1.0
        return output, loss
```

运行自动评判程序，得到结果如下图所示：




## 3.2 训练/验证/可视化/比较

### 3.2.1 Hinge loss 模拟 SVM 的训练及验证

classify_hw.py 中 FeatureDataset 的代码：

```
class FeatureDataset(Dataset):
    def __init__(self, file_path):
        self.data = np.load(file_path)
        self.labels = np.concatenate([np.ones(self.data.shape[0] //
            2).astype(np.float32),
        -1.0 * np.ones(self.data.shape[0] // 2).astype(np.float32)],
            axis=0)
    def __len__(self):
        return self.data.shape[0]
    def __getitem__(self, item):
        feature = self.data[item]
        label = self.labels[item]
        return feature, label
```

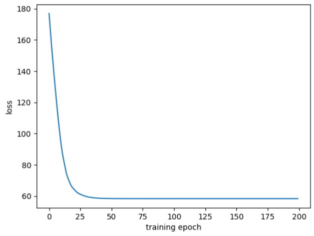
train_val_hinge 的代码：

```
def train_val_hinge(train_file_path, val_file_path,
                    feature_channels, C, n_epochs, batch_size,
                    lr, valInterval, device='cpu'):
    trainloader = dataLoader(train_file_path, batch_size)
    valloader = dataLoader(val_file_path, batch_size)
    C = C * len(trainloader)
    model = SVM_HINGE(feature_channels, C)
    model = model.to(device)
    optimizer = optim.Adam(model.parameters(), lr=lr)
    # training
```

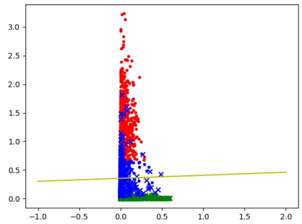
```python
losses = []
for epoch in range(n_epochs):
    model.train()
    total_loss = 0.
    for feas, labels in trainloader:
        feas, labels = feas.float().to(device),
            labels.float().to(device)
        optimizer.zero_grad()
        out, loss = model(feas, labels)
        loss.backward()
        total_loss += loss.item()
        optimizer.step()
    # average of the total loss for iterations
    avg_loss = total_loss / len(trainloader)
    losses.append(avg_loss)
    print('Epoch {:02d}: loss = {:.3f}'.format(epoch + 1, avg_loss))
    # validation
    if (epoch + 1) % valInterval == 0:
        model.eval()
        n_correct = 0.
        n_feas = 0.
        with torch.no_grad():
            for feas, labels in valloader:
                feas, labels = feas.float().to(device),
                    labels.float().to(device)
                out, _ = model(feas)
                predictions = out[:, 0]
                n_correct += torch.sum((predictions == labels).float())
                n_feas += feas.size(0)
        # show prediction accuracy
        print('Epoch {:02d}: validation accuracy =
            {:.1f}%'.format(epoch + 1, 100 * n_correct / n_feas))
```

### 3.2.2 可视化分类结果

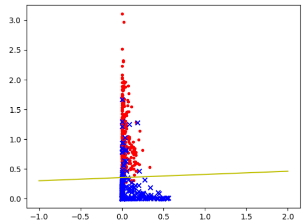
使用 hinge loss 模拟 SVM 以及 libsvm 库对数据集进行分类，结果如下：



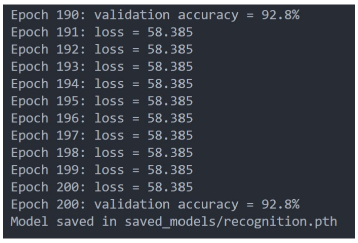

(a) loss 曲线



(b) train 中的特征点分布图



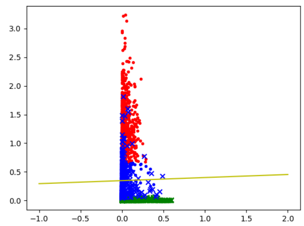
(c) validate 中的特征点分布图



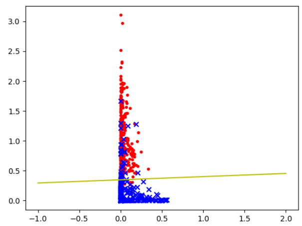
```
Epoch 190: validation accuracy = 92.8%
Epoch 191: loss = 58.385
Epoch 192: loss = 58.385
Epoch 193: loss = 58.385
Epoch 194: loss = 58.385
Epoch 195: loss = 58.385
Epoch 196: loss = 58.385
Epoch 197: loss = 58.385
Epoch 198: loss = 58.385
Epoch 199: loss = 58.385
Epoch 200: loss = 58.385
Epoch 200: validation accuracy = 92.8%
Model saved in saved_models/recognition.pth
```

(d) 后十余轮的 loss 和准确率

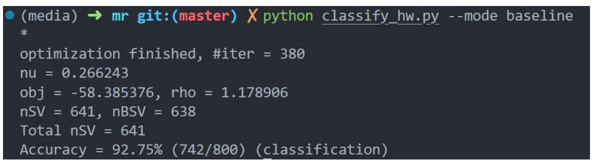
图 1: hinge loss 模拟 SVM 对数据集进行分类



(a) train 中的特征点分布图



(b) validate 中的特征点分布图



```
(media) ➜ mr git:(master) ✗ python classify_hw.py --mode baseline
*
optimization finished, #iter = 380
nu = 0.266243
obj = -58.385376, rho = 1.178906
nSV = 641, nBSV = 638
Total nSV = 641
Accuracy = 92.75% (742/800) (classification)
```

(c) 准确率

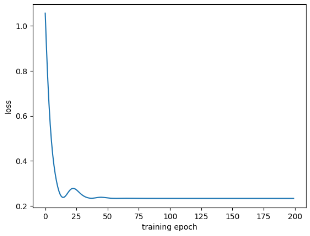
图 2: libsvm 库对数据集进行分类

hinge loss 模拟 SVM 进行分类的准确率和 libsvm 库进行分类的准确率十分接近，约为 93%。支持向量和分类边界的表现也类似。

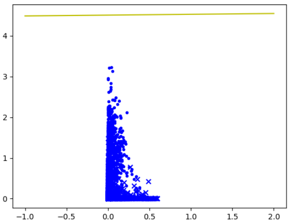
### 3.2.3 调整正则化参数 C，体会不同的 C 对分类效果的影响

分别设置不同的参数 $C = 0.0001, 0.001, 0.01, 0.1, 1, 10$，比较在 C 不同取值下两种方式在验证集上的分类效果：

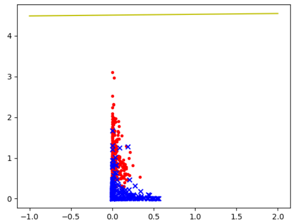
当 $C = 0.0001$ 时：




(a) hinge loss + loss



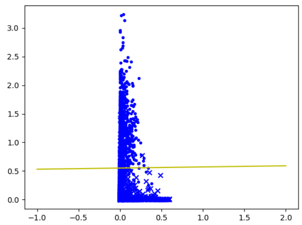
(b) hinge loss + train 中的特征点分布



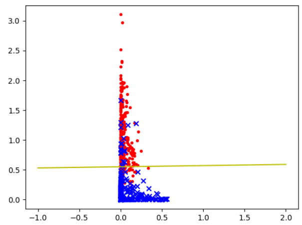
(c) hinge loss + 的 validate 中的特征点分布

```
Epoch 190: validation accuracy = 50.0%
Epoch 191: loss = 0.234
Epoch 192: loss = 0.234
Epoch 193: loss = 0.234
Epoch 194: loss = 0.234
Epoch 195: loss = 0.234
Epoch 196: loss = 0.234
Epoch 197: loss = 0.234
Epoch 198: loss = 0.234
Epoch 199: loss = 0.234
Epoch 200: loss = 0.234
Epoch 200: validation accuracy = 50.0%
Model saved in saved_models/recognition.pth
```

(d) 后十余轮的 loss 和准确率



(e) libsvm + train 中的特征点分布



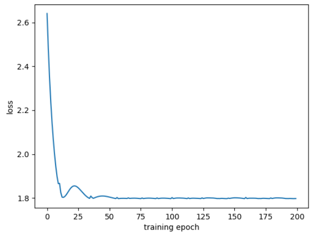
(f) libsvm + validate 中的特征点分布

```
optimization finished, #iter = 1200
nu = 1.000000
obj = -0.233730, rho = 0.180408
nSV = 2400, nBSV = 2400
Total nSV = 2400
Accuracy = 54.875% (439/800) (classification)
```

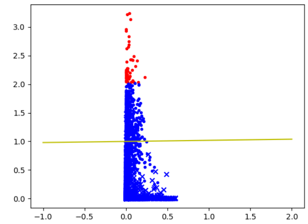
(g) libsvm + 准确率

图 3: C=0.0001

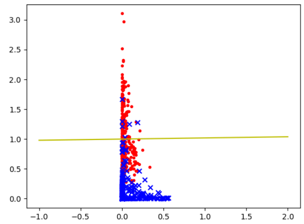
当 $C = 0.001$ 时:




(a) hinge loss + loss



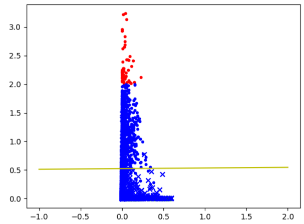
(b) hinge loss + train 中的特征点分布



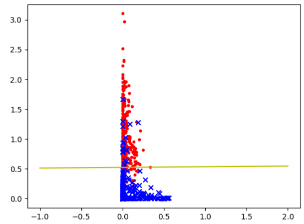
(c) hinge loss + 的 validate 中的特征点分布

```
Epoch 190: validation accuracy = 67.6%
Epoch 191: loss = 1.798
Epoch 192: loss = 1.797
Epoch 193: loss = 1.798
Epoch 194: loss = 1.797
Epoch 195: loss = 1.798
Epoch 196: loss = 1.798
Epoch 197: loss = 1.797
Epoch 198: loss = 1.797
Epoch 199: loss = 1.797
Epoch 200: loss = 1.797
Epoch 200: validation accuracy = 67.6%
Model saved in saved_models/recognition.pth
```

(d) 后十余轮的 loss 和准确率



(e) libsvm + train 中的特征点分布



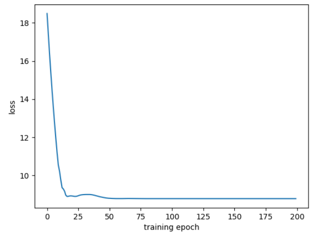
(f) libsvm + validate 中的特征点分布

```
optimization finished, #iter = 1181
nu = 0.956667
obj = -1.796798, rho = 1.000257
nSV = 2296, nBSV = 2296
Total nSV = 2296
Accuracy = 67.625% (541/800) (classification)
```

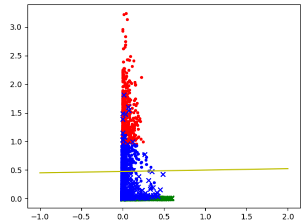
(g) libsvm + 准确率

图 4: C=0.001

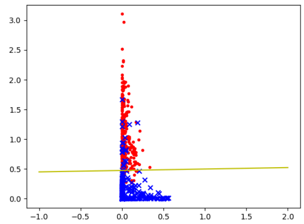
当 $C = 0.01$ 时：



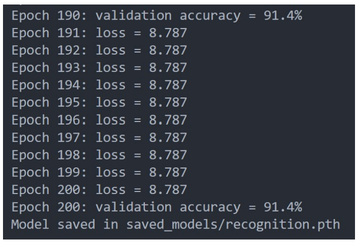

(a) hinge loss + loss



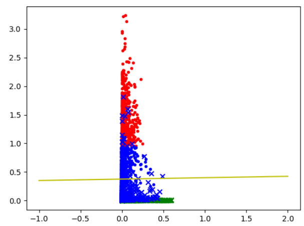
(b) hinge loss + train 中的特征点分布



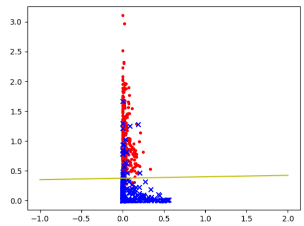
(c) hinge loss + 的 validate 中的特征点分布



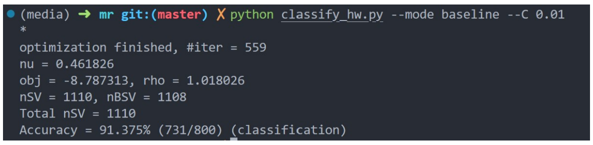

(d) 后十余轮的 loss 和准确率



(e) libsvm + train 中的特征点分布



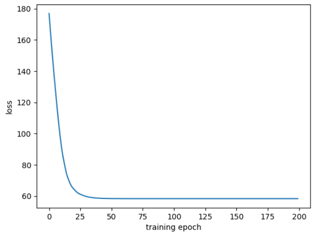
(f) libsvm + validate 中的特征点分布



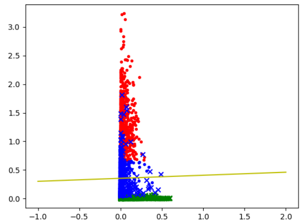

(g) libsvm + 准确率

图 5: C=0.01

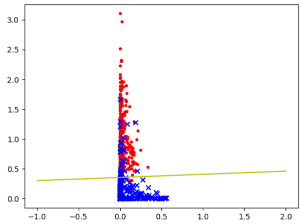
当 $C = 0.1$ 时：



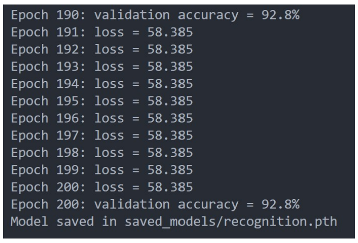

(a) hinge loss + loss



(b) hinge loss + train 中的特征点分布

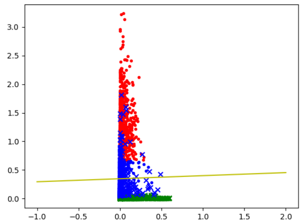


(c) hinge loss + 的 validate 中的特征点分布

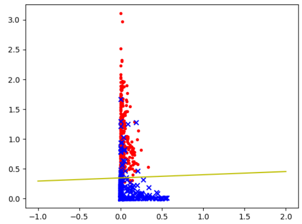


```
Epoch 190: validation accuracy = 92.8%
Epoch 191: loss = 58.385
Epoch 192: loss = 58.385
Epoch 193: loss = 58.385
Epoch 194: loss = 58.385
Epoch 195: loss = 58.385
Epoch 196: loss = 58.385
Epoch 197: loss = 58.385
Epoch 198: loss = 58.385
Epoch 199: loss = 58.385
Epoch 200: loss = 58.385
Epoch 200: validation accuracy = 92.8%
Model saved in saved_models/recognition.pth
```

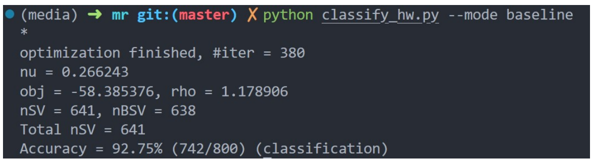
(d) 后十余轮的 loss 和准确率



(e) libsvm + train 中的特征点分布



(f) libsvm + validate 中的特征点分布

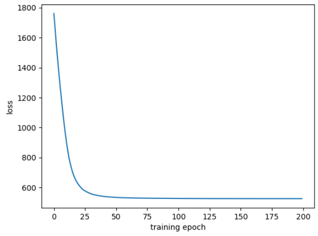


```
(media) ➜ mr git:(master) ✗ python classify_hw.py --mode baseline
*
optimization finished, #iter = 380
nu = 0.266243
obj = -58.385376, rho = 1.178906
nSV = 641, nBSV = 638
Total nSV = 641
Accuracy = 92.75% (742/800) (classification)
```

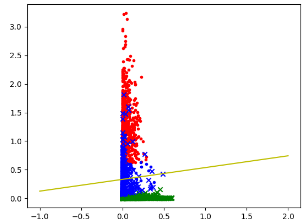
(g) libsvm + 准确率

图 6: C=0.1

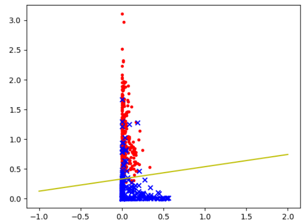
当 $C = 1$ 时：



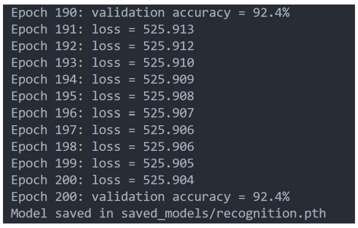

(a) hinge loss + loss



(b) hinge loss + train 中的特征点分布

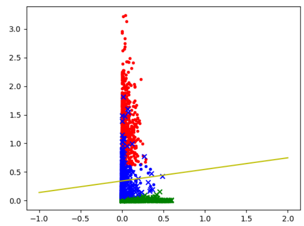


(c) hinge loss + 的 validate 中的特征点分布

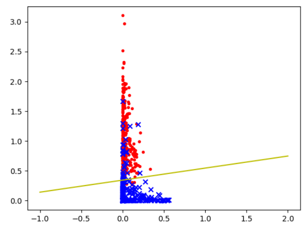


```
Epoch 190: validation accuracy = 92.4%
Epoch 191: loss = 525.913
Epoch 192: loss = 525.912
Epoch 193: loss = 525.910
Epoch 194: loss = 525.909
Epoch 195: loss = 525.908
Epoch 196: loss = 525.907
Epoch 197: loss = 525.906
Epoch 198: loss = 525.906
Epoch 199: loss = 525.905
Epoch 200: loss = 525.904
Epoch 200: validation accuracy = 92.4%
Model saved in saved_models/recognition.pth
```

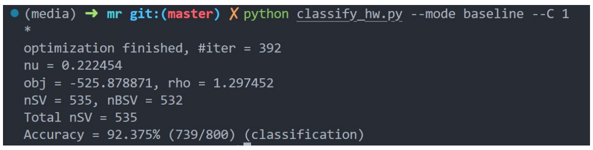
(d) 后十余轮的 loss 和准确率



(e) libsvm + train 中的特征点分布



(f) libsvm + validate 中的特征点分布

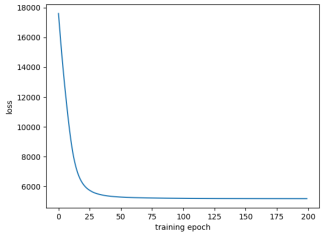


```
(media) ➜ mr git:(master) ✗ python classify_hw.py --mode baseline --C 1
*
optimization finished, #iter = 392
nu = 0.222454
obj = -525.878871, rho = 1.297452
nSV = 535, nBSV = 532
Total nSV = 535
Accuracy = 92.375% (739/800) (classification)
```

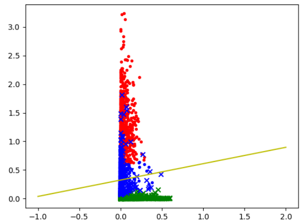
(g) libsvm + 准确率

图 7: C=1

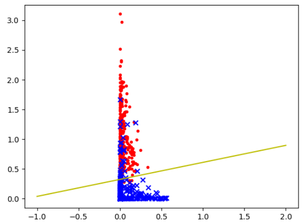
当 $C = 10$ 时：



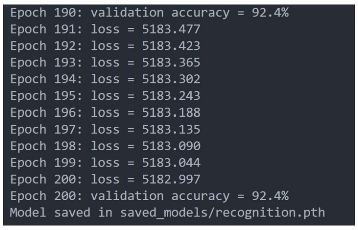

(a) hinge loss + loss



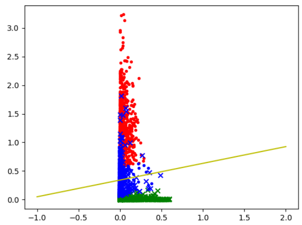
(b) hinge loss + train 中的特征点分布



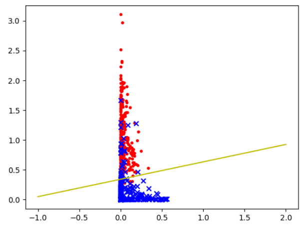
(c) hinge loss + 的 validate 中的特征点分布



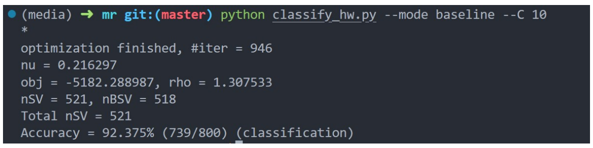

(d) 后十余轮的 loss 和准确率



(e) libsvm + train 中的特征点分布



(f) libsvm + validate 中的特征点分布




(g) libsvm + 准确率

图 8: C=10

总结以上结果，得到：

| C | 0.0001 | 0.001 | 0.01 | **0.1** | 1 | 10 |
|---|---|---|---|---|---|---|
| hinge loss | 50.0% | 67.6% | 91.4% | **92.8%** | 92.4% | 92.4% |
| libsvm | 54.875% | 67.625% | 91.375% | **92.75%** | 92.375% | 92.375% |

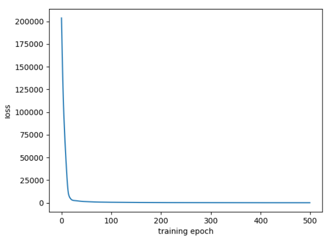
可以看到，正则化参数 C 在 0.1 附近时，两种方法的准确率均达到最大值，分别为 92.8% 和 92.75%；当正则化参数过大时，准确率保持稳定，几乎不会下降。hinge loss 模拟 SVM 和 libsvm 的结果相近，说明 hinge loss 的代码实现是正确的。

### 3.3 使用 SVM 算法在 MNIST 数据集上进行分类

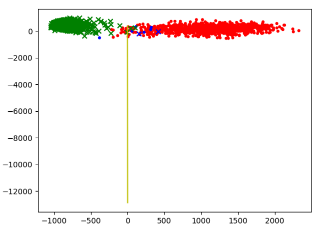
补全 process_mnist.py 中的代码如下：

```
data_mean = np.mean(train_data, axis=0)
data_cov = np.cov(train_data.T)
u, sigma, vh = np.linalg.svd(data_cov)
train_data = np.matmul(train_data - data_mean, u[:, :opt.feat_dim])
val_mean = np.mean(val_data, axis=0)
val_cov = np.cov(val_data.T)
u, sigma, vh = np.linalg.svd(val_cov)
val_data = np.matmul(val_data - val_mean, u[:, :opt.feat_dim])
```

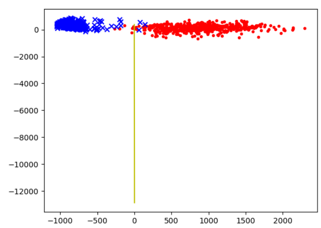
从原始 MNIST 数据集中提取数字 0 和数字 1 的图片特征，通过 PCA 算法降到 10 维，并保存.npy 文件，提取成功后，在高维数据集上使用实现的 SVM 算法进行分类，结果如下。其中，使用 hinge loss 模拟的 epoch 数目设置为 500。



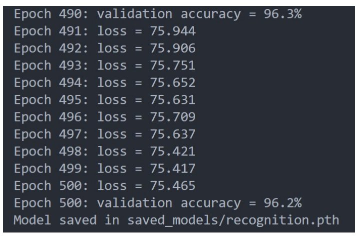

(a) loss 曲线



(b) train 中的特征点分布图

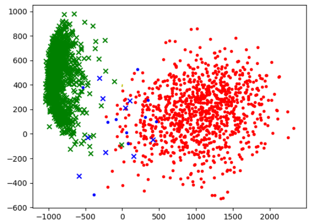


(c) validate 中的特征点分布图

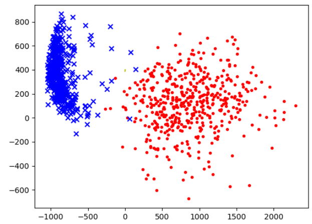



(d) 500 轮的损失和准确率

图 9: hinge loss 模拟 SVM 对 10 维数据集进行分类




(a) train 中的特征点分布图




(b) validate 中的特征点分布图

```
optimization finished, #iter = 3682997
nu = 0.006703
obj = -1.342247, rho = -0.731447
nSV = 19, nBSV = 8
Total nSV = 19
Accuracy = 95.6% (956/1000) (classification)
```

(c) 准确率

图 10: libsvm 库对 10 维数据集进行分类

不管是何种分类方法，准确率都比较高，证明了 PCA 确实能有效地提取特征，减少运算量。

# 4 总结

这次作业涉及了前几周的知识内容，我通过练习和巩固加深了对支持向量机、PCA 特征降维等知识的理解。通过写代码，遇到问题并及时解决，提高了自身的编程能力和实践能力。同时，对于理论部分，能够亲自动手计算并体验，进一步巩固了理解。
遇到的问题其一是 hinge loss 的前向传播函数以及反向传播函数错误，后来发现是矩阵的维度不对应，我通过查看 data 和 label 的尺寸，将 label 进行了 unsqueeze 的操作解决了这个问题。其二是环境的问题，在安装 libsvm 库时未按照文档给出的命令执行，出现了奇怪的错误，后来在他人提示下顺利安装正确的版本。