清华大学电子工程系

媒体与认知 课堂 2

2021-2022 学年春季学期

作业 4

孙一 2022 年 4 月 20 日

理论部分

1 单选题（15 分）

1.1 D

1.2 C

1.3 B

1.4 A

1.5 D

2 计算题（15 分）

2.1 假设邮件粗略分为垃圾邮件和正常邮件，且存在一种垃圾邮件的检测方法，其中垃圾邮件被正确检测的概率为 a，正常邮件被误判为垃圾邮件的概率为 b。针对某一邮箱，所有邮件中垃圾邮件占的比例为 c，如果某封邮件被判定为垃圾邮件，根据贝叶斯定理，这封邮件是垃圾邮件的概率是多少？(提示：全概率公式 $P(Y)=\sum_{i=1}^{N} P(Y|X_i)P(X_i)$)

解答过程见图 1

2.2 给定样本集合, 其均值为 $\mu=[1,2]^T$, 样本协方差矩阵为 C，且已知 $CU=U\lambda$。其中 $U=\begin{bmatrix} 0.5 & -0.4 \\ 0.5 & 0.4 \end{bmatrix}$, $\lambda=\begin{bmatrix} 10.7 & \\ 0 & 0.4 \end{bmatrix}$。试用主成分分析 PCA 将样本 $x=\begin{bmatrix} 3 \\ 1 \end{bmatrix}$ 变换至一维。(提示: 样本数据应减去均值；特征向量应归一化)

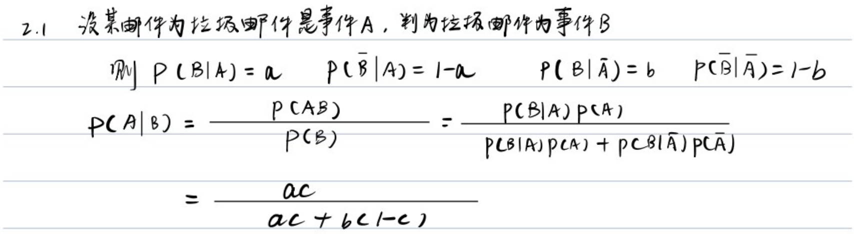


图 1: 2.1

解答过程见图 2

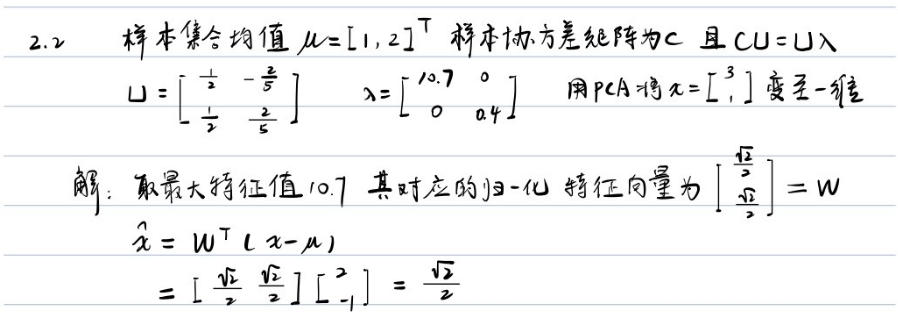


图 2: 2.2

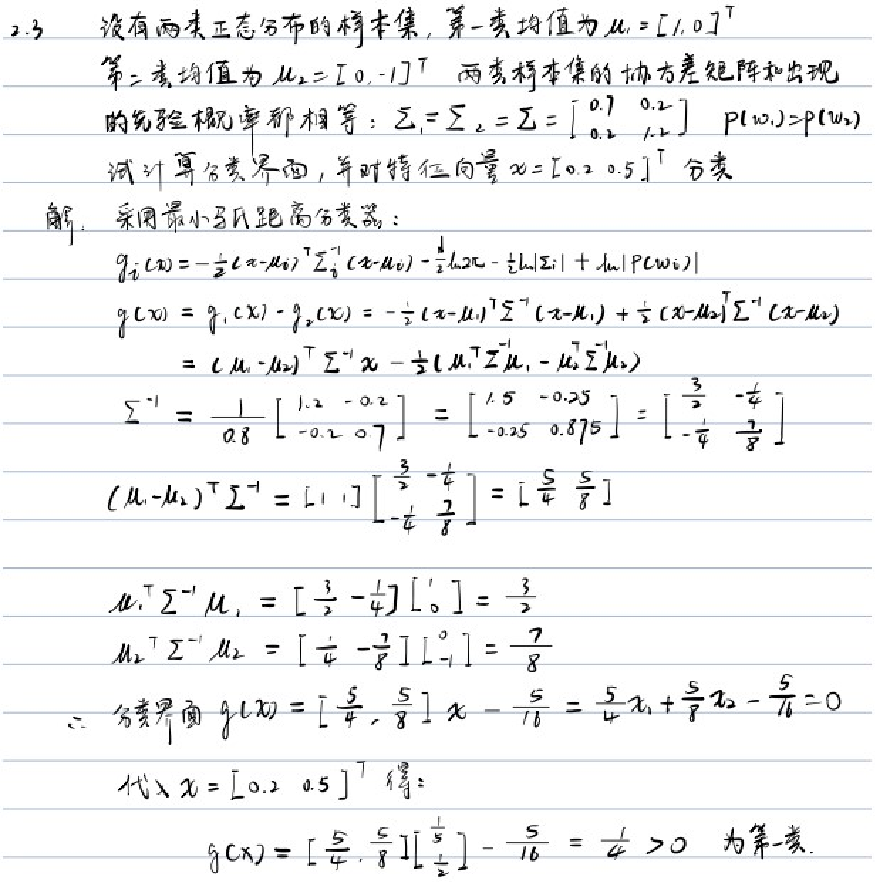
2.3 设有两类正态分布的样本集，第一类均值为 $\mu_1=[1,0]^T$，第二类均值为 $\mu_2=[0,-1]^T$。两类样本集的协方差矩阵和出现的先验概率都相等：$\Sigma_1=\Sigma_2=\Sigma=\begin{bmatrix}0.7 & 0.2\\ 0.2 & 1.2\end{bmatrix}$，$p(\omega_1)=p(\omega_2)$。试计算分类界面，并对特征向量 $x=[0.2,0.5]^T$ 分类。

解答过程见图 3

编程部分

3 目的

使用支持向量机完成线性分类任务：字符/背景图片特征分类，对比 libsvm 库的分类结果和使用线性层 +hinge loss 的模拟结果。



2.3 设有两类正态分布的样本集，第一类均值为 $\mu_1=[1,0]^T$，第二类均值为 $\mu_2=[0,-1]^T$，两类样本集的协方差矩阵和出现的先验概率都相等：$\Sigma_1=\Sigma_2=\Sigma=\begin{bmatrix}0.7 & 0.2\\ 0.2 & 1.2\end{bmatrix}$，$p(\omega_1)=p(\omega_2)$，试计算分类界面，并对特征向量 $x=[0.2\ 0.5]^T$ 分类

解. 采用最小马氏距离分类器：

$$g_i(x)=-\frac{1}{2}(x-\mu_i)^T\Sigma_i^{-1}(x-\mu_i)-\frac{d}{2}\ln 2\pi-\frac{1}{2}\ln|\Sigma_i|+\ln|P(\omega_i)|$$

$$g(x)=g_1(x)-g_2(x)=-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)+\frac{1}{2}(x-\mu_2)^T\Sigma^{-1}(x-\mu_2)$$

$$=(\mu_1-\mu_2)^T\Sigma^{-1}x-\frac{1}{2}(\mu_1^T\Sigma^{-1}\mu_1-\mu_2^T\Sigma^{-1}\mu_2)$$

$$\Sigma^{-1}=\frac{1}{0.8}\begin{bmatrix}1.2 & -0.2\\ -0.2 & 0.7\end{bmatrix}=\begin{bmatrix}1.5 & -0.25\\ -0.25 & 0.875\end{bmatrix}=\begin{bmatrix}\frac{3}{2} & -\frac{1}{4}\\ -\frac{1}{4} & \frac{7}{8}\end{bmatrix}$$

$$(\mu_1-\mu_2)^T\Sigma^{-1}=[1\ 1]\begin{bmatrix}\frac{3}{2} & -\frac{1}{4}\\ -\frac{1}{4} & \frac{7}{8}\end{bmatrix}=[\frac{5}{4}\ \frac{5}{8}]$$

$$\mu_1^T\Sigma^{-1}\mu_1=[\frac{3}{2}\ -\frac{1}{4}]\begin{bmatrix}1\\0\end{bmatrix}=\frac{3}{2}$$

$$\mu_2^T\Sigma^{-1}\mu_2=[\frac{1}{4}\ -\frac{7}{8}]\begin{bmatrix}0\\-1\end{bmatrix}=\frac{7}{8}$$

$\therefore$ 分类界面 $g(x)=[\frac{5}{4},\frac{5}{8}]x-\frac{5}{16}=\frac{5}{4}x_1+\frac{5}{8}x_2-\frac{5}{16}=0$

代入 $x=[0.2\ \ 0.5]^T$ 得：

$$g(x)=[\frac{5}{4},\frac{5}{8}]\begin{bmatrix}\frac{1}{5}\\ \frac{1}{2}\end{bmatrix}-\frac{5}{16}=\frac{1}{4}>0$$ 为第一类.

图 3: 2.3

4 实现 hinge loss 模拟支持向量机并运行自动评判程序

4.1 说明

程序中部分变量的形状见表 1

4.2 实现线性层的前向计算过程

```
# TODO: compute the output of the linear function: output=xW^T + b
output = torch.matmul(x, W.T) + b
ctx.save_for_backward(x, W, b)
```

变量	形状
channels	dimension of features: 2
W	[1, channels = 2]
b	[1,]
X	[batch_size, channels = 2]
output	[batch_size, 1]
grad_output	[batch_size, 1]
grad_W	[1, channels]
grad_b	[1,]

表 1: 变量说明

4.3 实现线性层的反向传播过程

```
# TODO: compute the grad with respect to W and b: dL/dW, dL/db
grad_W = (grad_output * x).sum(0).reshape(1,-1)
grad_b = grad_output.sum(0)#.reshape(1,-1)
```

4.4 实现 hinge loss + L2 norm

```
# TODO: compute the hinge loss (together with L2 norm for SVM):
# loss = 0.5*||w||^2 + C*\sum_i{max(0, 1 - y_i*output_i)}
loss = 0.5*torch.norm(W)**2 + C*torch.sum(
        torch.relu(1-label.view(-1,1) * output), dim=0)
ctx.save_for_backward(output, W, label, C)
```

4.5 实现 loss 层的反向传播过程

反向传播过程的理论解释见图 4

```
# TODO: compute the grad with respect to the output of the linear function and W:
# dL/doutput, dL/dW
grad_output = C * grad_loss * ((1-label.view(-1,1)*output)>0)*(-label.view(-1,1))
grad_W = grad_loss * W
```

4.6 定义线性层的参数 W，b

```
# TODO: define the parameters W and b
self.W = nn.Parameter(torch.randn(1,in_channels),requires_grad = True)
```

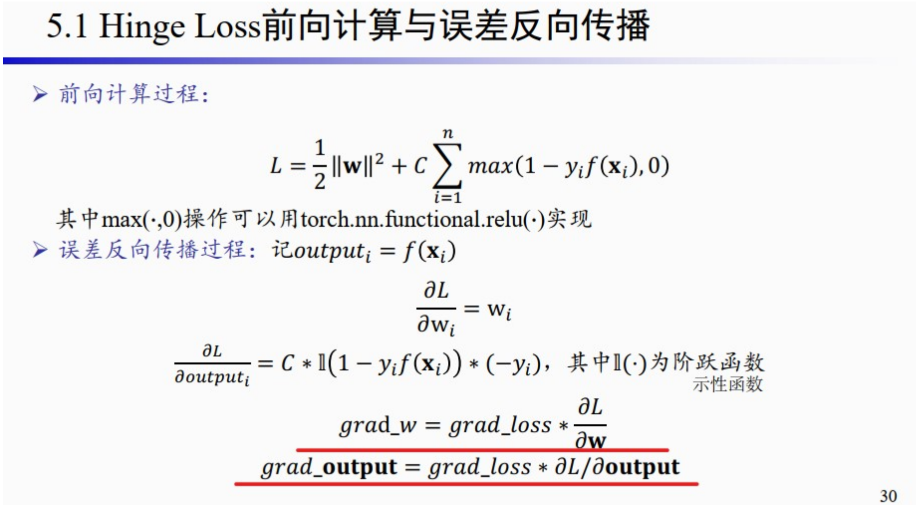


图 4: Hinge Loss 前向计算与误差反向传播

```
self.b = nn.Parameter(torch.randn(1,),requires_grad = True)
self.C = torch.tensor([[C]], requires_grad=False)
```

4.7 运行自动评判程序检验 svm_hw.py 代码实现效果

运行成功结果见图 5

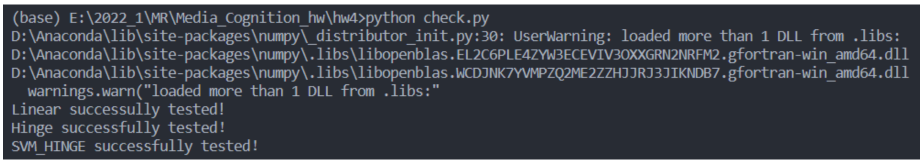


图 5: check_success

5 训练/验证/可视化/比较

5.1 Hinge loss 模拟 SVM 的训练及验证

5.1.1 实现图像特征的数据类

```
# TODO: define the __len__ function of the Dataset class
# return the number of samples (N) in self.data, using .shape[dim]
# the shape of self.data is (N, channels)
def __len__(self):
```

```
        return self.data.shape[0]

    # TODO: define the __getitem__ function of the Dataset class
    # item is an integer >= 0, indicating the index of an sample
    # return the corresponding data and label according to item
    # the returned feature should be of the shape (channels, ) and the
    # returned label should be of the shape (1, )
    def __getitem__(self, item):
        # feature denotes one element in self.data, and label denotes one
        # element in self.labels
        feature = self.data[item]    # with shape (channels, )
        label = self.labels[item]
        return feature, label
```

5.1.2 实现 hinge loss 模拟 SVM 的训练，验证代码

```
# TODO: training and validation data loader using the previous
# self-defined function dataLoader()
trainloader = dataLoader(train_file_path, batch_size)
valloader = dataLoader(val_file_path, batch_size)


# TODO: initialize the hinge-loss type SVM;the SVM_HINGE class needs two
# parameters: in_channels, and C.
model = SVM_HINGE(feature_channels, C)
# TODO: put the model on CPU or GPU
model = model.to(device)
# TODO: initialize the Adam optimizer with model parameters and learning rate
optimizer = optim.Adam(model.parameters(), lr)


# TODO: set the model in training mode
model.train()
# to save total loss in one epoch
total_loss = 0.
# TODO: get a batch of data; you may need enumerate() to iteratively get
# data from trainloader.
```

```python
# you can refer to previous homework, for example hw2
for idx, (feas, labels) in enumerate(trainloader):
    # TODO: set data type (.float()) and device (.to())
    feas, labels = feas.float().to(device), labels.float().to(device)
    # TODO: clear gradients in the optimizer
    optimizer.zero_grad()
    # TODO: run the model with hinge loss; the model needs two inputs:
    #feas and labels
    out, loss = model(feas, labels)
    # TODO: back-propagation on the computation graph
    loss.backward()
    # sum up of total loss, loss.item() return the value of the tensor as
    # a standard python number
    # this operation is not differentiable
    total_loss += loss.item()
    # TODO: call a function to update the parameters of the models
    optimizer.step()


# TODO: set the model in evaluation mode
model.eval()
n_correct = 0.  # number of images that are correctly classified
n_feas = 0.  # number of total images
with torch.no_grad():  # we do not need to compute gradients during validation
    # TODO: inference on the validation dataset, similar to the training
    # stage but use valloader.
    for idx, (feas, labels) in enumerate(valloader):
        # TODO: set data type (.float()) and device (.to())
        feas, labels = feas.float().to(device), labels.float().to(device)
        # TODO:
        # run the model; at the validation step, the model only needs one
        # input: feas
        # _ refers to a placeholder, which means we do not need the
        # second returned value during validating
        out, _ = model(feas)
```

5.2 可视化分类结果

hinge loss 的训练结果如下：

```
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
```

默认参数下（$\mathrm{C} = 0.1$），使用 hinge loss 模拟 SVM 训练的 loss 曲线见图 6。训练集上特征点分布图（包含特征点、支持向量以及分类边界）见 7，验证集上特征点分布图见 8。
默认参数下（$\mathrm{C} = 0.1$），采用 libsvm 库训练，训练集上特征点分布图（包含特征点、支持向量以及分类边界）见 9，验证集上特征点分布图见 10。
对比图 7和图 9、图 8和图 10，可以看出用 hinge loss 和 libsvm 库计算得到的结果几乎完全相同，包括数据点在坐标系中的分布、支持向量的分布以及分类界面的位置，说明用 hinge loss 模拟 SVM 的效果还是非常精确的。
图例说明见表 2

训练数据	说明	验证数据	说明
红点	正类	红点	正类
蓝点	正类支持向量	蓝叉	负类
绿叉	负类	黄线	分类边界
蓝叉	负类支持向量		
黄线	分类边界		

表 2: 图例说明

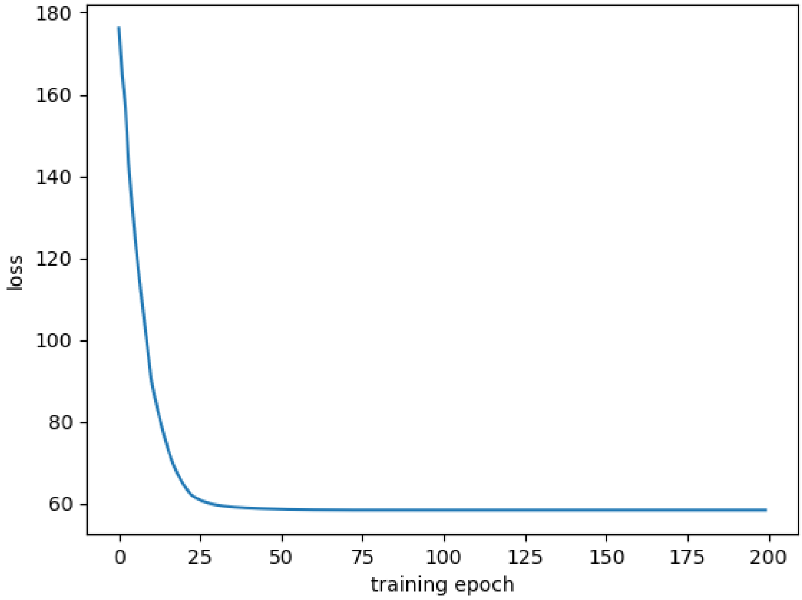


图 6: hinge loss 曲线

5.3 调整正则化系数 C，体会不同的 C 对分类效果的影响

通过分别设置不同的参数 C=0.0001,0.001,0.01,0.1,1,10，比较在 C 的不同取值下两种模式在验证集上的分类效果。$C = 0.1$ 的结果在 5.2中已经给出，其余预测结果图见 11至 20。所有情况下的分类正确率汇总如表 3。

C	hinge loss 模拟 SVM 分类正确率	libsvm 分类正确率
0.0001	50.0%	54.875% (439/800)
0.001	68.6%	67.625% (541/800)
0.01	91.4%	91.375% (731/800)
0.1	92.8%	92.75% (742/800)
1	92.4%	92.375% (739/800)
10	92.4%	92.375% (739/800)

表 3: 不同 C 下验证集上的预测结果

从表 3中可以看出，C 的大小和 SVM 对分类误差的容忍度呈负相关关系，即 C 越大，即要求模型的误差越小，进入间隔区间的点越少，可以防止过拟合；C 越小，即模型的误差越大，训练集上容易出现欠拟合。

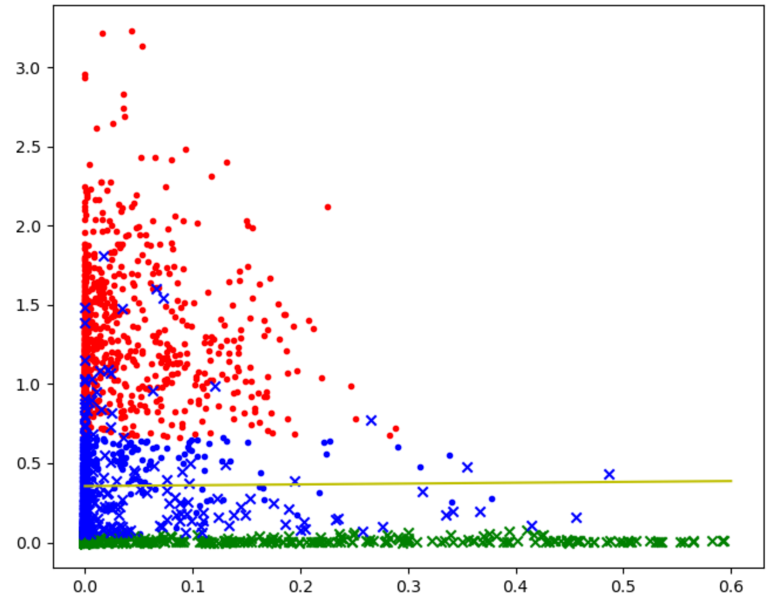


图 7: C=0.1 hinge loss train feas

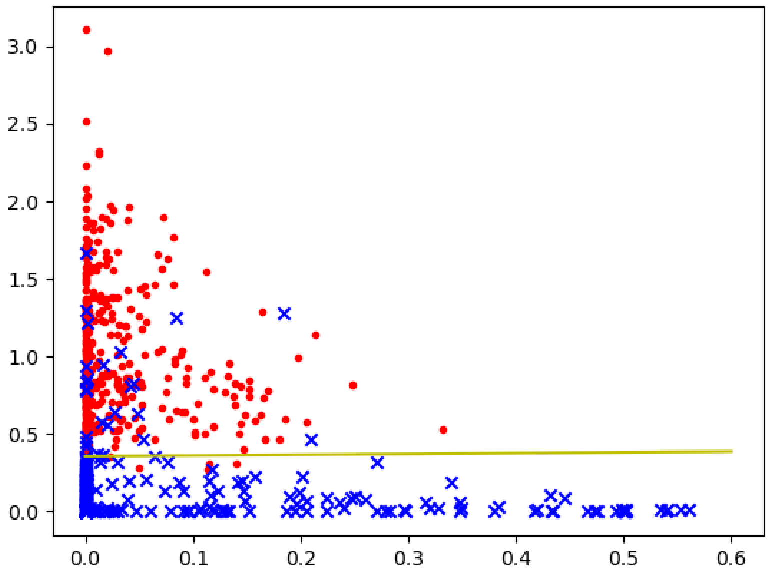


图 8: C=0.1 hinge loss val feas

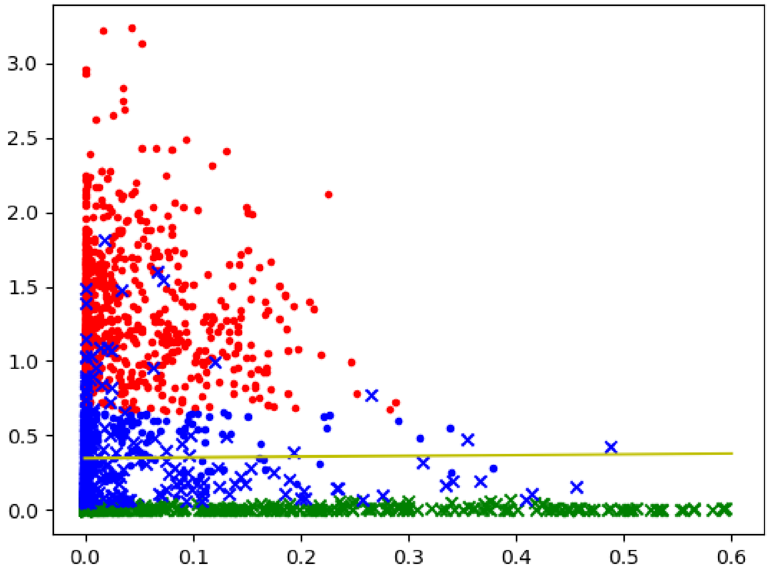


图 9: C=0.1 libsvm train feas

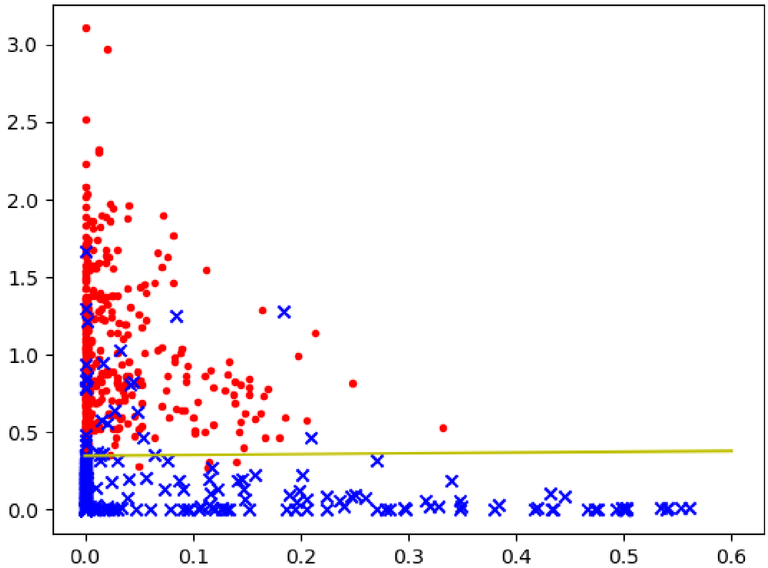


图 10: C=0.1 libsvm val feas

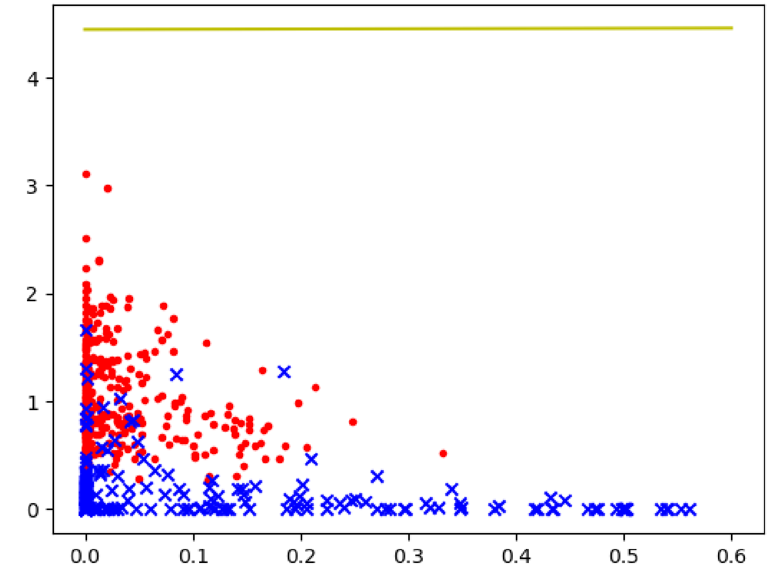


图 11: C = 0.0001 hinge loss val feas

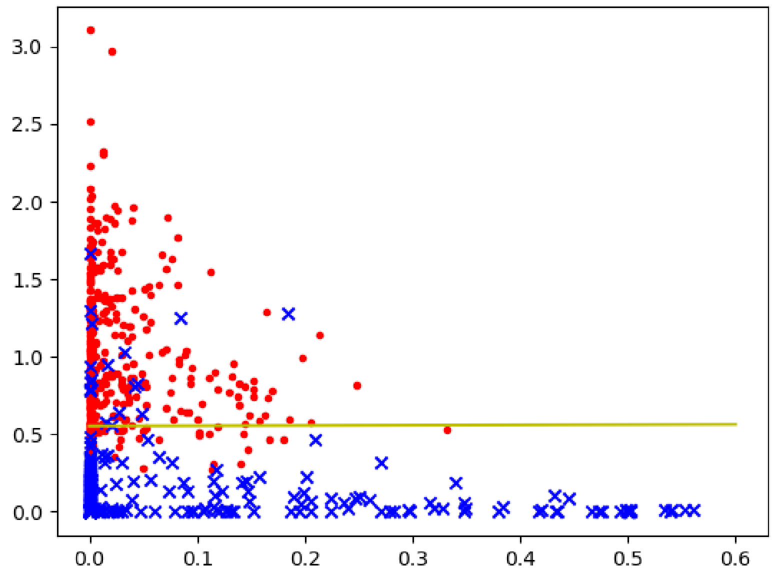


图 12: C = 0.0001 libsvm val feas

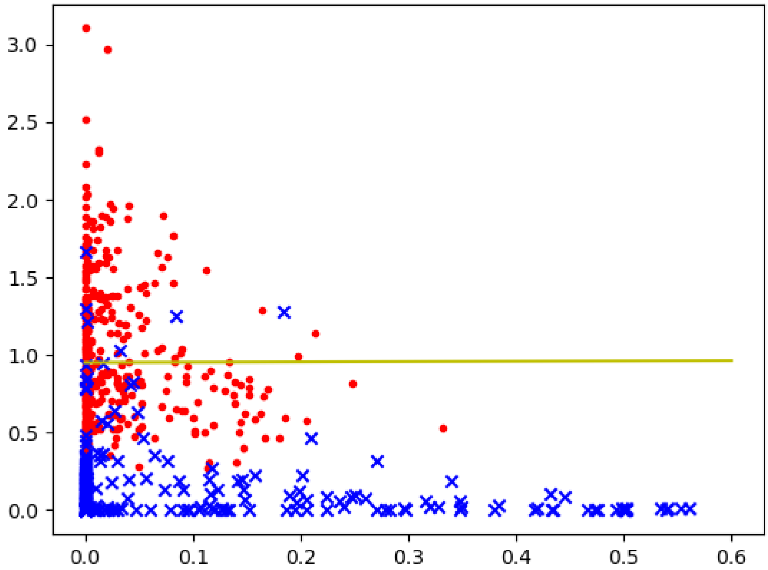


图 13: C = 0.001 hinge loss val feas

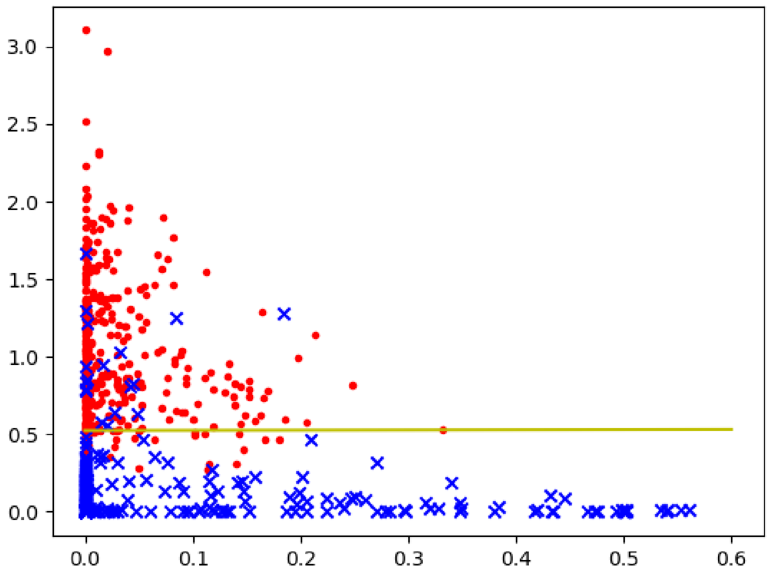


图 14: C = 0.001 libsvm val feas

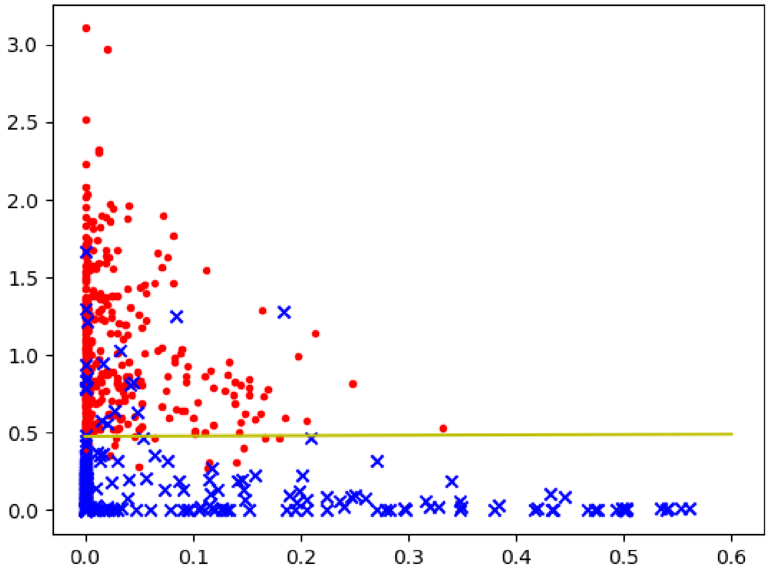


图 15: C = 0.01 hinge loss val feas

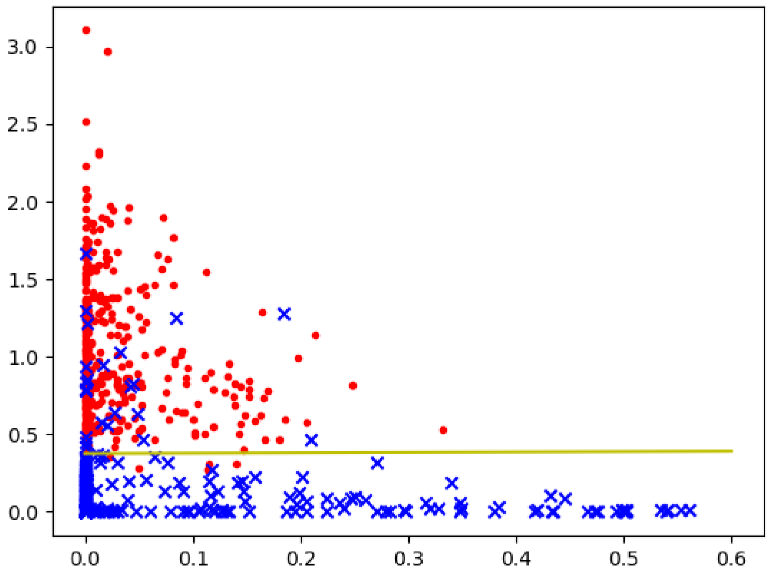


图 16: C = 0.01 libsvm val feas

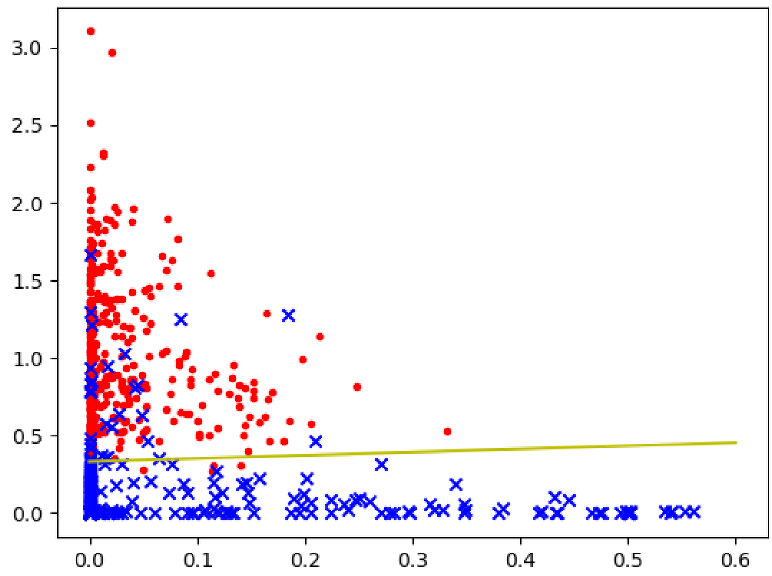


图 17: C = 1 hinge loss val feas

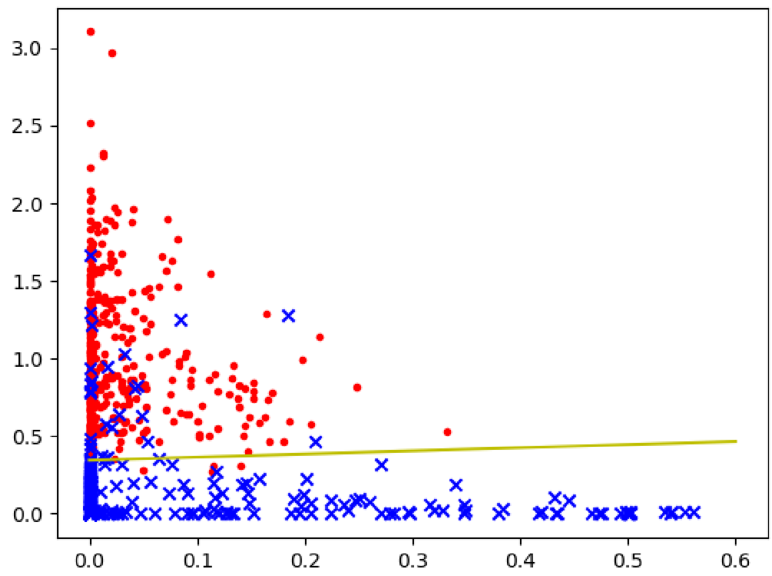


图 18: C = 1 libsvm val feas

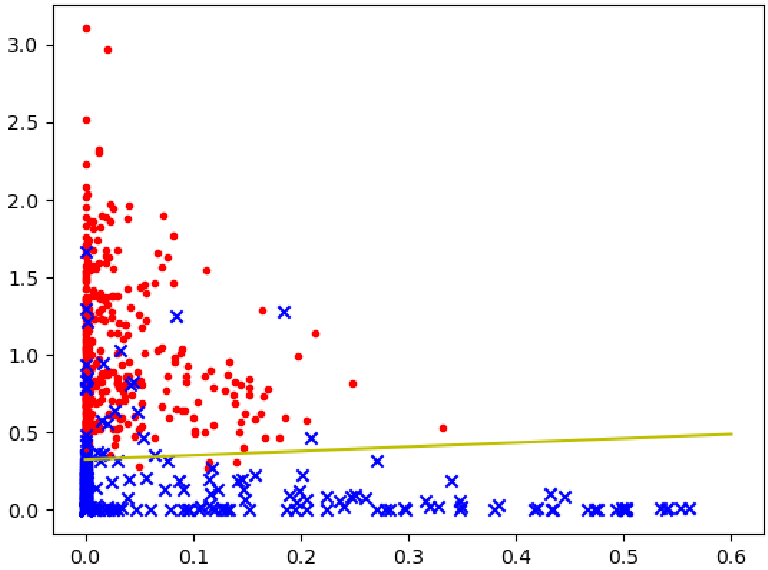


图 19: C = 10 hinge loss val feas

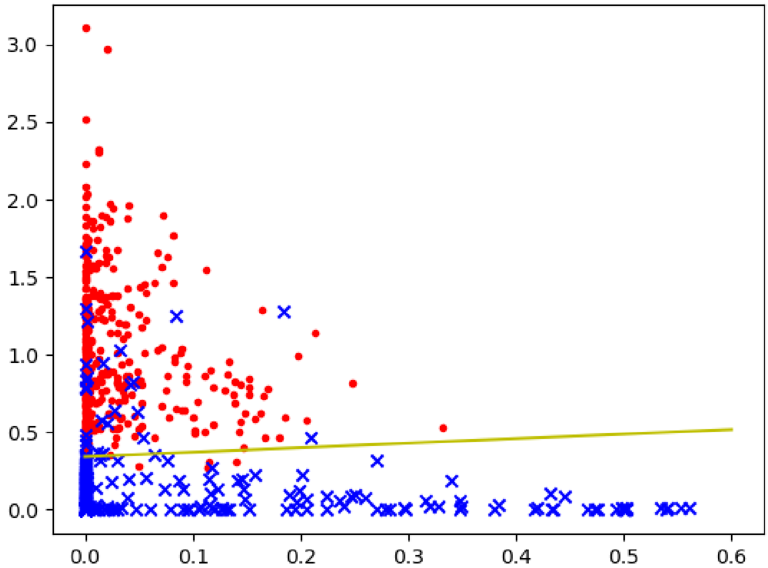


图 20: C = 10 libsvm val feas