数据隐私实验二

傅申 PB20000051

目录

1. 代码实现方法

Q1. 补全 $\hat{y}$ 计算流程

这一部分对应于论文中算法的 Step 2 和 Step 3.

对于 Active Party A, 它首先计算 $\Theta^A x_i^A$, 并接收来自 Passive Party B 的 $\Theta^B x_i^B$. 然后二者相加得到 Θx_i, 再根据 Activation Function 计算 $\hat{y}_i = h_\Theta(x_i)$. 代码实现如下:

active.py line 83 ~ 90

```
83  # Q1. Active party calculates y_hat
84  # ---------------------------------------------------------------------
85  x = self.x_train[batch_idxes]
86  active_wx = self.params.dot(x.T)
87  passive_wx = self.messenger.recv()
88  full_wx = active_wx + passive_wx
89  y_hat = self.activation(full_wx)
90  # ---------------------------------------------------------------------
```

对于 Passive Party B, 它只需要计算 $\Theta^B x_i^B$, 并发送给 Active Party A 即可, 代码实现如下:

passive.py line 92 ~ 97

```
92  # Q1. Calculate wx and send it to active party
93  # ---------------------------------------------------------------------
94  x = self.x_train[batch_idxes]
95  passive_wx = self.params.dot(x.T)
96  self.messenger.send(passive_wx)
97  # ---------------------------------------------------------------------
```

Q2. 补全梯度计算流程

这一部分对应于论文中算法的 Step 3, Step 4 和 Step 5.

首先, Active Party A 计算 $\hat{y}$ 和 y 之间的误差 $(y_i - \hat{y}_i)$, 并将误差加密后得到 $[\![(y_i - \hat{y}_i)]\!]$. 然后 A 将加密后的误差发送给 Passive Party B, 代码实现如下:

active.py line 95 ~ 104

```
95  residue = self.y_train[batch_idxes] - y_hat
96  residue = np.array(
97      [round(res, self.RESIDUE_PRECISION) for res in residue]
98  )
99
100 # Q2. Active party helps passive party to calculate gradient
101 # ---------------------------------------------------------------------
102 enc_residue = self.cryptosystem.encrypt_vector(residue)
103 enc_residue = np.array(enc_residue)
104 self.messenger.send(enc_residue)
```

然后, Passive Party B 接收加密后的误差, 并由此计算 Θ^B 的 (加密后的) 梯度 $[\![\frac{\partial L}{\partial \Theta^B}]\!]$. 然后, B 生成随机掩码 (Mask) R_B, 并在加密后的梯度上加上加密后的随机掩码, 再将结果 $[\![\frac{\partial L}{\partial \Theta^B}]\!] + [\![R_B]\!] = [\![\frac{\partial L}{\partial \Theta^B} + R_B]\!]$ 发送给 Active Party A. 代码实现如下:

passive.py line 100 ~ 105

```
100 # Q2. Receive encrypted residue and calculate masked encrypted gradients
101 # ---------------------------------------------------------------------
102 enc_residue = self.messenger.recv()
103 enc_grad = self._gradient(enc_residue, batch_idxes)
104 enc_mask_grad, mask = self._mask_grad(enc_grad)
105 self.messenger.send(enc_mask_grad)
```

其中, `_mask_grad()` 的实现如下:

passive.py line 45 ~ 49 (LinearPassive._mask_grad)

```
45  def _mask_grad(self, enc_grad):
46      mask = np.random.normal(0, 1.0, enc_grad.shape)
47      enc_mask = self.cryptosystem.encrypt_vector(mask)
48      enc_mask_grad = enc_grad + enc_mask
49      return enc_mask_grad, mask
```

Active Party A 在接收加密后的梯度后, 将其解密并发送回 Passive Party B. 代码实现如下:

active.py line 105 ~ 108

```
105 enc_passive_grad = self.messenger.recv()
106 passive_grad = self.cryptosystem.decrypt_vector(enc_passive_grad)
107 self.messenger.send(passive_grad)
108 # ---------------------------------------------------------------------
```

最后, Passive Party B 接收解密后的梯度, 并去除其中的掩码, 得到解密后的梯度, 由此进行梯度下降. 代码实现如下:

passive.py line 106 ~ 111

```
106 # Receive decrypted masked gradient and update model
107 mask_grad = self.messenger.recv()
108 true_grad = self._unmask_grad(mask_grad, mask)
109 # ---------------------------------------------------------------------
110
111 self._gradient_descent(self.params, true_grad)
```

在上述过程中, Active Party A 同时 (在代码中表现为发送解密后的梯度后) 计算自己的梯度 $\frac{\partial L}{\partial \Theta^A}$, 并进行梯度下降. 代码实现如下:

active.py line 110 ~ 112

```
110  # Active party calculates its own gradient and update model
111  active_grad = self._gradient(residue, batch_idxes)
112  self._gradient_descent(self.params, active_grad)
```

Q3. 补全训练过程中模型准确率计算流程

这一部分需要补全 `active.py` 中的 `LinearActive._acc()` 函数. 代码实现如下:

active.py line 144 ~ 151 (LinearActive._acc)

```
144  def _acc(self, y_true, y_hat):
145      # Q3. Compute accuracy
146      # ------------------------------------------------------------------
147      y_pred = np.where(y_hat >= 0.5, 1, 0)
148      acc = np.sum(y_true == y_pred) / len(y_true)
149      # ------------------------------------------------------------------
150
151      return acc
```

2. 实验结果

先后运行 `play_active.py`, 即可开始训练纵向联邦逻辑回归模型. 运行效果如图 1 所示. 从截图中可以看出, 对于最后一个 batch, 模型的 loss 为 0.0844, 准确率为 100%.

记录训练过程每个 epoch 的模型的 loss 和准确率, 并绘制对应的曲线, 如图 2 所示. 从曲线中可以看出, 模型达到了较低的 loss 和较高的准确率, 可以认为模型已经收敛. 因此, 可以认为模型的训练是成功的.

3. 文字描述说明

- 请说明代码中 `scale()` 函数的原理及作用

  原理 `scale()` 函数通过使用 `sklearn` 库中的 `preprocessing.scale()` 函数将数据集中每个特征的值进行标准化, 使得每个特征的数据都近似于标准正态分布.

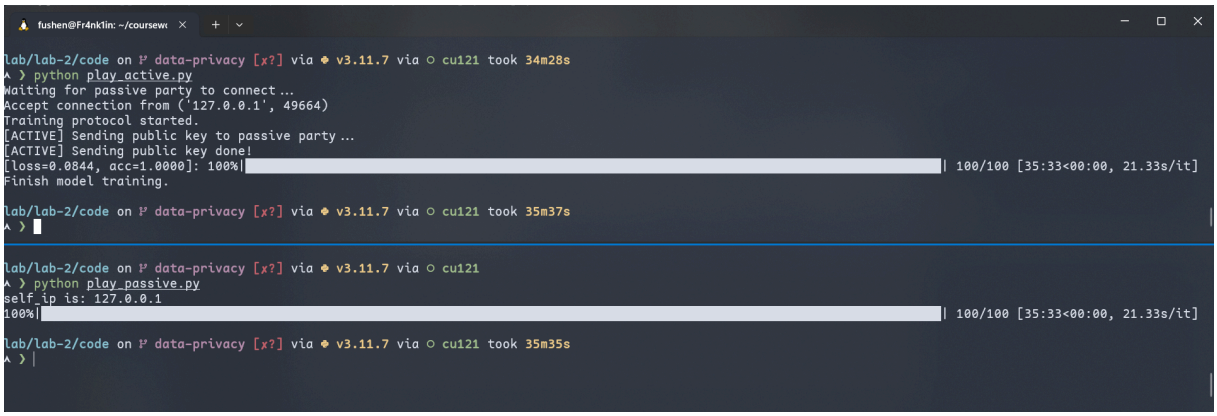

图 1 训练过程截图

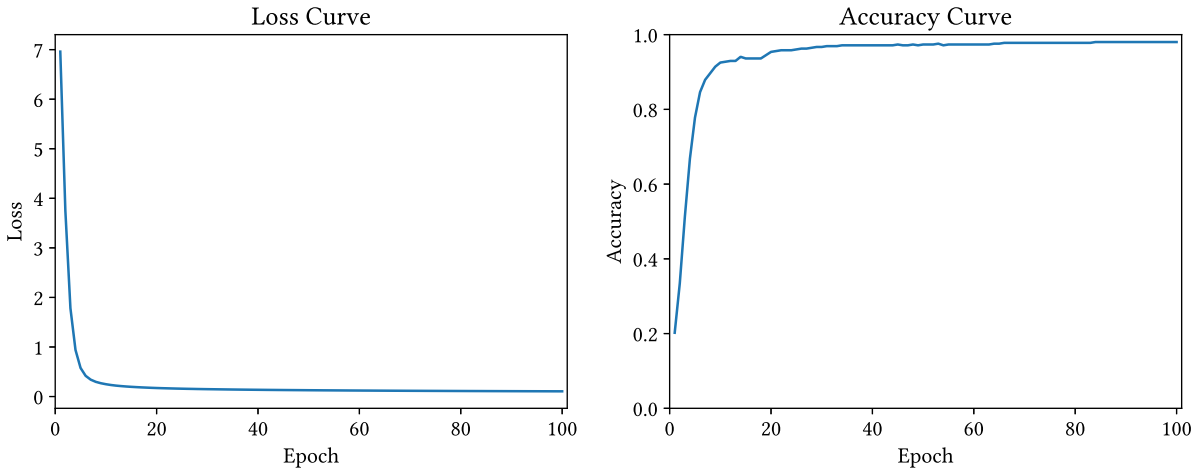

图 2 训练过程中的模型性能, 左图为 loss 曲线, 右图为准确率曲线

作用 这样做可以避免规模较大的特征主导模型参数, 有助于平衡各个特征对模型参数的影响, 并提升模型的性能.

- 当前代码在每个 epoch 开始时使用 epoch 值作为随机数种子, 请说明含义, 并实现另一种方式以达到相同的目的

  含义 不断使用动态的随机数种子 (epoch 值) 重设 (伪) 随机数生成器, 可以避免 Active Party 通过分析随机掩码 (Mask) 的模式而推断出 Passive Party 的模型参数. 这种方式可以进一步保证隐私性.

  另一种实现 可以使用其他动态参数作为随机数种子. 例如基于当前时间 t 计算出 $f(t)$ 作为随机数种子, 其中 f 对 Active Party 是不可见的.

- 试分析 VFL-LR 训练流程中潜在的隐私泄露风险, 并简要说明可能的保护方式

  风险 如果 Active Party 是恶意的, 它可以在 Step 3 中设置误差为只含一个 1, 其余位置均为 0 的向量, 加密后发送给 Passive Party, 导致 Passive Party 在 Step 4/5 中计算得到的梯度中直接包含了模型的部分参数, 并被 Active Party 获取, 造成隐私泄露.

  可能的保护方式 Passive Party 可以增加随机掩码 (Mask) 的规模, 并在训练过程中随机重排特征和参数的顺序 (二者重排顺序相同, 不影响 $\Theta^B x^B$ 的值), 从而增加 Active Party 推断模型参数的难度, 保护隐私.