软聚类联邦学习 FedSoft 算法的复现

赖伟民

摘要

传统的聚类联邦学习将具有相同数据分布的客户端分组到一个簇中，因此每个客户端都唯一地与一个数据分布相关联。FedSoft 提出了一种软聚类联邦学习方法，通过软聚类将客户端分组，这意味着每个客户端不是严格分配到单个群集中，而是根据其数据分布在多个群集之间有不同的从属度。这种方法允许每个本地数据集遵循多个源分布的混合，使联邦学习能够更灵活地处理各种数据分布。并且，在本地训练中 FedSoft 使用近端正则化更新，限制了客户端的工作量，只要求在每轮通信中的一部分客户端完成一个优化任务。对 FedSoft 算法的复现结果显示，该算法能有效地利用源分布之间的相似性来学习表现良好的个性化和簇模型。

关键词：联邦学习；客户端软聚类

1 引言

FedSoft [13] 的研究背景源于传统聚类联邦学习（Federated Learning, FL）的局限性。在传统聚类联邦学习（Clustered Federated Learning, CFL）中，具有相同数据分布的客户端被分组到一个簇中，每个客户端唯一地与一个数据分布相关联并帮助训练该分布的模型。然而，这种硬性关联假设在实际应用中可能不太现实，因为真实世界中的用户数据更可能遵循多种数据分布的混合。例如，如果每个客户端是一部手机，我们可能期望将客户端分为成人和儿童两类，但实际上成人用户可能偶尔浏览儿童内容，而青少年或父母和孩子共用的设备可能同时包含成人和儿童的数据。此外，传统的 CFL 在实际应用中面临着无法有效利用不同簇之间的相似性的问题。

为应对这一挑战，FedSoft 放宽了硬性关联的假设，采用软聚类联邦学习方法。这种方法允许每个本地数据集遵循多个源分布的混合，更好地反映了现实世界中数据的多样性。但是，在软聚类联邦学习中，由于每个客户端可能来自多个数据源，客户端的工作量会激增，并且集群模型的训练和局部个性化是不同的。

因此，FedSoft 通过使用近端更新来限制客户端的工作量，并在每轮通信中只要求一部分客户端完成一个优化任务，有效地利用源分布之间的相似性来学习个性化和簇模型，并表现出良好的性能。这种方法能够更好地处理客户端数据的多样性，并在保持个性化的同时提高模型的整体性能。此外，FedSoft 在理论上提供了收敛保证，并在实验中验证了其在处理各种混合模式数据下的有效性。

2 相关工作

2.1 联邦学习

在广泛使用的联邦学习算法 FedAvg [11] 中，客户端通过迭代运行本地更新，并将它们的中间本地模型与中央服务器同步，共同训练一个共享的机器学习模型。尽管在诸如下一个词预测 [5] 和电子健康记录学习 [2] 等应用中取得了成功，但 FL 在处理客户端本地数据分布异质性或非独立同分布（Non-IID）数据时，存在模型精度下降、训练缓慢等问题。为了降低统计异质性的影响，FedProx [7] 提出了一种局部近端项来优化局部模型，使用近端项来限制本地模型更新的差异。然而，在复杂的深度学习模型上，FedProx 的效果并不明显。

2.2 聚类联邦学习

硬聚类联邦学习（Hard Clustered Federated Learning, CFL）的训练目标是同时识别簇划分并为每个簇训练一个模型。现有工作通常采用类似于期望最大化的算法，交替进行簇识别和模型训练。根据划分客户端的方式，这些算法可分为四类：

1. **利用模型参数之间的距离**：例如，Long 等人 [8] 提出根据客户端模型和服务器模型之间的距离来确定客户端之间的关联。类似地，Briggs 等人 [1] 提出直接在客户模型上应用基于距离的层次聚类算法。

2. **基于梯度信息确定划分结构**：例如，Sattler 等人 [14] 提出基于客户端梯度的余弦相似度将客户端分成双分区，然后通过检查客户端上的梯度范数来检查分区是否一致（即是否包含 IID 数据）。FedGroup [3] 算法用分解余弦相似性度量的欧氏距离量化客户端梯度之间的相似性，该算法使用奇异值分解将梯度分解为多个方向。

3. **利用训练损失**：例如，在 perCluster [9] 中，每个客户端都被贪婪地分配到其模型在其本地数据上产生最低损失的集群。该算法提供了泛化保证。Ghosh 等人 [4] 提出了一种名为 IFCA 的类似算法，其收敛界是在良好的初始化假设下建立的，并且所有客户端具有相同的数据量。

4. **使用关于数据的外部信息**：例如，Huang 等人 [6] 和 Qayyum 等人 [12] 根据电子病历和成像模式分别将患者分组。这些信息通常需要直接访问用户数据，因此不能用于一般情况。

近期，Marfoq 等人 [10] 提出了一种类似于软聚类 FL 的多任务学习框架，允许客户端数据遵循多种分布混合。他们提出的 FedEM 算法采用 EM 算法，并基于训练损失估计混合系数。然而，FedEM 要求每个客户端在每轮中为每个簇执行本地更新，这比传统的 FedAvg 需要更多的训练时间。与此相反，FedSoft 只要求部分客户端在每轮中仅返回一个优化任务的梯度。

3 本文方法

3.1 本文方法概述

FedSoft 是一种软聚类联邦学习算法，旨在训练群集模型和个性化本地模型，同时降低客户端的工作负担。本文提出的软聚类算法允许客户端数据遵循多个分布的混合，而不是单一分布。这种方法适应于现实世界中数据分布的多样性和复杂性。

近端更新的方法是 FedSoft 的另一个关键特点。这种更新策略通过在本地目标中加入正则化项，结合本地数据和群集模型，允许客户端在执行本地个性化的同时，利用全局共享的知识。近端更新使得每个客户端只需解决一个单一的优化问题，减少了训练负担和复杂度。

整体上，FedSoft 通过其软聚类方法和近端更新策略有效地平衡了个性化需求和群集模型的训练，展示了在各种混合模式下的良好性能。FedSoft 不仅在实验中也证明了其有效性，而且在理论上也提供了收敛性保证，是一种适应于非独立同分布数据的联邦学习方法。

3.2 客户端软聚类算法

在软聚类联邦学习 FedSoft 中，客户端可能拥有来自多个集群的数据。假设每个客户端的数据点从 S 个不同的数据分布 $P_1, ..., P_s$ 中抽样。给定一个损失函数 $L(w; x, y)$，在传统的聚类联邦学习中，客户端只能与一个集群相关联。客户端在本地计算其本地训练数据在所有簇模型上的损失，选择损失最小的簇模型进行联邦学习训练。而 FedSoft 放宽了这一硬关联的假设，允许每个客户端的本地数据集遵循多个源分布的混合。与硬聚类不同，FedSoft 的客户端在计算本地数据在簇模型上的损失时，为每一条数据保留其在不同簇模型上的损失值，根据损失值的大小判断该数据应该属于哪一个集群。

$$j = \arg\min_{s} L(c_s^t, x_k^i, y_k^i) \tag{1}$$

其中，s 是簇的数量，c_s^t 是簇模型。FedSoft 计算每条数据属于某一个簇，根据相对应数据的量来确定当前客户端来自某一个簇的可能性。例如，根据公式 1 计算得出，当前客户端有 25% 的数据属于簇 0，有 30% 的数据属于簇 1，有 45% 的数据属于簇 2。计算所得出的数据量与可能性，在联邦学习的客户端选择已经本地优化中使用。即 FedSoft 在每一轮联邦学习全局迭代中，为每一个簇独立地选择参与本轮训练的客户端，簇中属于该簇的数据量越多，选择的可能性就越大。在客户端本地训练中，根据可能性的大小为近端更新赋予相应的权重。在簇中择客户端被选择参与本轮迭代的可能性的计算公式如下：

$$v_{sk}^t = \frac{u_{ks}^t n_k}{\sum_{k' \in Sel_s^t} u_{k's}^t n_{k'}} \tag{2}$$

其中，$u_{ks}^t = \max\left\{\frac{n_{ks}^t}{n_k}, \sigma\right\}$ 是客户端中属于该簇数量的比例，由 σ 控制最小的可能性，n_k 是当前客户端的数据总量。

3.3 近端局部更新策略

客户端软聚类的策略会导致客户端工作负载急剧增加，并且集群模型的训练和局部个性化是不一致的，因此，作者使用近端局部更新技术降低了客户端的工作负担，在每轮全局通

信只客户端只需要完成一个优化任务。作者在损失函数中加入近端项，近端目标中的正则化项作为局部模型训练的参考点。它允许客户端在利用和引导中心的全局共享知识的同时在自己的特定数据集上工作，如下公式所示：

$$h_k(w_k; c^t, u^t) \triangleq f_k(w_k) + \frac{\lambda}{2} \sum_{s=1}^{S} u_{ks}^t \|w_k - c_s^t\|^2 \tag{3}$$

在这个公式中，h_k 是一个函数，由局部损失函数 f_k 与一个近端正则化项的总和定义。正则化项由 λ 加权，它计算模型参数 w 与簇中心 c_s^t 之间的平方欧氏距离的总和，每个距离都乘以软聚类权重 u_{ks}。正则化项由软聚类权重 u_{ks} 加权，以便客户端更加关注其数据中具有更高份额的分布。

正则化项在近端目标函数中起到了关键作用，复合正则化有助于识别个别中心，可以将公式 (2) 分解为一系列子优化问题：

$$h_k(w_k; c^t) = \sum_{s=1}^{S} u_{ks} \left(F_s(w_k) + \frac{\lambda}{2} \|w_k - c_s^t\|^2 \right) \tag{4}$$

在最小化 h_k 之后，对应于大 u_{ks} 的子问题也将得到近似解。我们可以使用输出局部模型 w_k^* 来更新这些中心，拥有大 u_{ks} 的值。此外，以这种方式训练的 w_k^* 会忽略所有单独的集群分布 $D_s || u_{ks} \neq 0$，这可能分享一些共同知识。因此，这些群集通过训练它们共有的客户端而连接在一起，利用群集之间的相似性。

输出模型 w_k^* 本身就是一个结合了本地客户知识和全局群集信息的模型。在某些条件下，FedSoft 的最优客户模型是最优群集模型的混合体。当 $\sum_{s=1}^{S} u_{ks}^t = 1$ 时，梯度 $\nabla_{w_k} h_k$ 是：

$$\nabla_{w_k} h_k = \nabla f_k(w_k) + \lambda \left(w_k - \sum_{s=1}^{S} u_{ks}^t c_s^t \right) \tag{5}$$

这意味着 w_k^* 应当以 $\sum_{s=1}^{S} u_{ks} c_s^*$ 为中心。因此，通过所有 h_k 的优化，不仅服务器将获得训练有素的群集模型，每个客户端也将获得足够个性化的本地模型。

4 复现细节

4.1 与已有开源代码对比

FedSoft 算法在 GitHub 上公开了代码，但是作者并没有写项目的使用说明，即 readme.md 文件，代码的细节以及实验超参数设置需要仔细对比原文。在复现论文实验时，需要根据文中的算法流程仔细查阅每个 python 函数的作用，以及代码运行的流程。FedSoft 算法如算法 1 所示。

Algorithm 1 FedSoft

Input: Global epoch T, importance weights estimation interval τ, number of clients N, client selection size K, counter smoother σ;

for $t = 0, \ldots, T-1$ **do**
  if $t \mod \tau = 0$ **then**
    Server sends centers $\{c_s^t\}$ to all clients;
    for each client k **do**
      for each data point (x_i^k, y_i^k) **do**
        $j = \arg\min_s \left(l(c_s^t, x_i^k, y_i^k)\right)$;
        $n_{kj}^t = n_{kj}^t + 1$;
      end for
      Send $u_{ks}^t = \max\left\{\frac{n_{ks}^t}{n_k}, \sigma\right\}$ to server;
    end for
  else
    Set $u_{ks}^t = u_{ks}^{t-1}$;
  end if
  Server computes v_{sk}^t as in $v_{sk}^t = \frac{u_{ks}^t n_k}{\sum_{k' \in Sel_s^t} u_{k's}^t n_{k'}}$;
  Server selects S sets of clients $Sel_s^t \subset [N]$ at random for each cluster, where $|Sel_s^t| = K$, and each client gets selected with probability v_{sk}^t;
  Selected clients download $\{c_s^t\}$, then compute $w_k^{t+1} = \arg\min_{w_k} h_k(w_k; c', u^t)$;
  Server aggregates $c_s^{t+1} = \frac{1}{K} \sum_{k \in Sel_s^t} w_k^{t+1}$;
end for

在本次对 FedSoft 算法的复现工作中，我主要完成了三个工作：

1. 首先，我梳理了完整的代码的运行流程；

2. 然后，我找出了作者代码中一段非常关键的错误，作者在验证的过程中，并未使用验证集进行测试。作者使用了训练集进行测试，这会导致在本地测试的时候，本地模型的准确率异常高，几乎趋近于 1。我发现了这一个关键性的错误，并在后续复现中修正了这一错误；

3. 最后，我额外添加了一个混合数据集 Digits5 的实验，Digits5 数据集由 5 个手写数字数据集混合而成（MNIST、MNIST-M、SYN、USPS、SVHN），相比于在实验中对单个数据集进行旋转的操作来生成不同的数据分布，Digits5 数据集能更好地适用于软聚类联邦学习的场景。

FedSoft 算法的核心代码如图 1所示，这段代码描述的是 FedSoft 服务器端的运行流程:

1. 其中代码的 52-92 行是一次完整的全局迭代（Global epoch）过程，由参数 num golbal epochs 控制全局迭代次数。

2. 代码 55-62 行是软聚类过程，在客户端本地计算本地数据属于某一集群的数据量，在服务器端根据数据量计算可能性，由超参数指定软聚类触发轮次。

3. 代码 66-73 行是客户端选择过程，在每个簇中根据可能性独立地选择当前簇参与联邦学习的客户端，然后再将被选客户端整合到一起，去除重复。

4. 代码 76-82 行是被选择的客户端本地训练和服务器聚合簇模型的过程。

5. 代码 85-92 行是在验证集上进行验证的过程，测试本地模型和簇模型的准确率。

```
50      def _run_fedsoft(self, num_global_epochs):
51          validation_dict = {}
52          for t in range(num_global_epochs):
53              start_time=time.time()
54              # Importance estimation
55              if t % self.server_solver.estimation_interval == 0:
56                  self.importance_weights_matrix = []  # dim = (num_clients, num_clusters)
57                  for client in self.client_vec:
58                      client.estimate_importance_weights()
59                      self.importance_weights_matrix.append(client.get_importance())
60                  self.importance_weights_matrix = np.array(self.importance_weights_matrix)
61                  self.importance_weights_matrix /= np.sum(self.importance_weights_matrix, axis=0)
62              print('time for importance estimation is {}'.format(time.time()-start_time))
63              start_time = time.time()
64
65              # Client selection
66              selection = []
67              if self.server_solver.do_selection:
68                  for s in range(self.num_clusters):
69                      selection.append(np.random.choice(a=range(self.num_clients), size=self.server_solver.selection_size,
70                                                        p=self.importance_weights_matrix[:, s], replace=False).tolist())
71                  logger.log_client_selection(self.exp_id, t, self._idx_to_id(selection))
72              else:
73                  selection = np.tile(range(self.num_clients), reps=(self.num_clusters, 1))
74
75              # Local updates
76              for k in np.unique(np.concatenate(selection).ravel()):
77                  self.client_vec[k].run()
78              print('time for local updates is {}'.format(time.time()-start_time))
79              # Aggregation
80              start_time=time.time()
81              self._aggregate_fedsoft(selection)
82              print('time for epoch {} is {}'.format( *args: t, time.time()-start_time))
83
84              # Validation
85              if t%5 ==0:
86                  start_time=time.time()
87                  if self.validator is not None:
88                      validation_dict[str(t)] = self.validator.validate(client_vec=self.client_vec,
89                                                                        cluster_vec=self.cluster_vec, t=t)
90                      if t % validation_data_flush_interval == 0 or t == num_global_epochs - 1:
91                          logger.log_validation_data(self.exp_id, validation_dict)
92                  print('time for validation is {}'.format(time.time()-start_time))
```

图 1. FedSoft 算法的核心代码

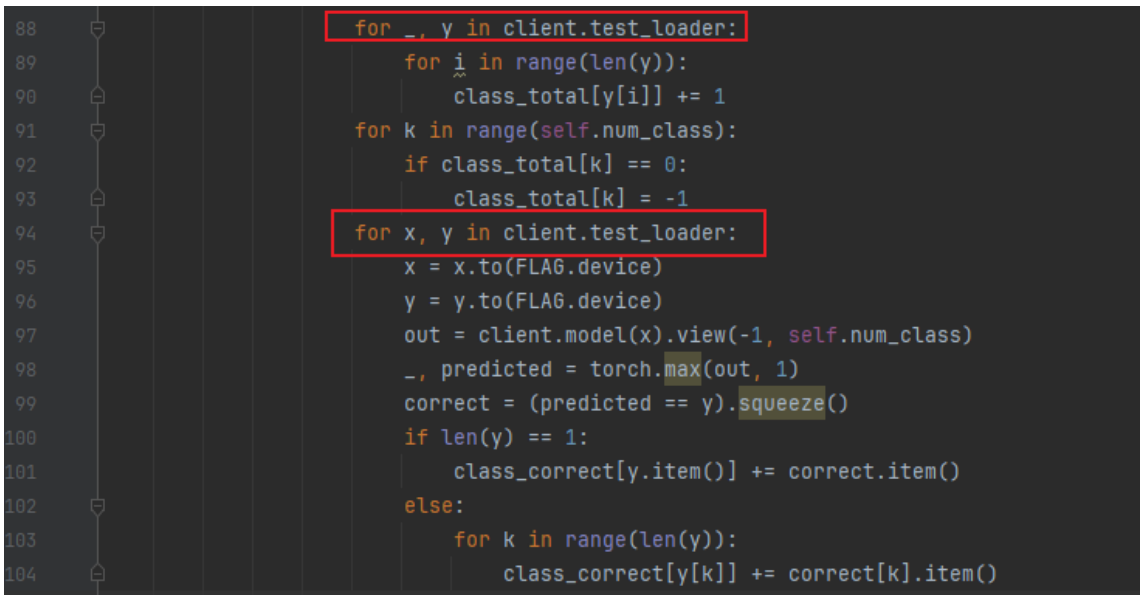
在作者公开的代码中，有一个非常关键的错误：作者使用了训练集进行测试，这会导致本地测试的精度虚高。我在复现的代码中修正了这一段错误，在客户端数据分割的地方，额外地为客户端分割出一个本地测试集，用于在训练过程中的测试。如图 2所示，作者使用训练集进行验证，这会导致图 3中本地测试出现 100% 的准确率。修改后的代码，如图 4所示，本地测试使用额外分割出来的测试集进行测试。

```python
if self.config.do_client_eval:
    client_set = [client_vec[i] for i in self.config.client_eval_idx_vec]
    if t == self.config.num_epochs-1:
        client_set = client_vec
    for i, client in enumerate(client_set):
        client.model.eval()
        info += '*' * 10 + ' Client {} '.format(client.ID) + '*' * 10 + '\n'
        validation_dict['client_eval']['client_' + str(client.ID)] = {}

        class_correct = list(0. for _ in range(self.num_class))
        class_total = list(0. for _ in range(self.num_class))
        for _, y in client.loader:
            for i in range(len(y)):
                class_total[y[i]] += 1
        for k in range(self.num_class):
            if class_total[k] == 0:
                class_total[k] = -1
        for x, y in client.loader:
            x = x.to(FLAG.device)
            y = y.to(FLAG.device)
            out = client.model(x).view(-1, self.num_class)
            _, predicted = torch.max(out, 1)
            correct = (predicted == y).squeeze()
            if len(y) == 1:
                class_correct[y.item()] += correct.item()
            else:
                for k in range(len(y)):
                    class_correct[y[k]] += correct[k].item()

        class_accuracy = list(
            1.0 * class_correct[k] / class_total[k] for k in range(self.num_class))
        average_accuracy = 1.0 * sum(class_correct) / sum(class_total)
        class_accuracy = [round(acc, 3) for acc in class_accuracy]
        info += 'Client {0:d} has average ' \
                'accuracy = {1:.3f}, class_accuracy = {2:}\n'.format( *args: client.ID, average_accuracy,
                                                                      class_accuracy)
        validation_dict['client_eval']['client_' + str(client.ID)]['average_accuracy'] = average_accuracy
        validation_dict['client_eval']['client_' + str(client.ID)]['class_accuracy'] = class_accuracy
```

图 2. 原代码中使用训练集进行测试

```
Client 93 has average accuracy = 1.000, class_accuracy = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
********** Client 94 **********
Client 94 has average accuracy = 1.000, class_accuracy = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
********** Client 95 **********
Client 95 has average accuracy = 1.000, class_accuracy = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
********** Client 96 **********
Client 96 has average accuracy = 1.000, class_accuracy = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
********** Client 97 **********
Client 97 has average accuracy = 1.000, class_accuracy = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
********** Client 98 **********
Client 98 has average accuracy = 1.000, class_accuracy = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
********** Client 99 **********
Client 99 has average accuracy = 0.997, class_accuracy = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.929, 1.0]
```

图 3. 本地测试准确率虚高



```
for _, y in client.test_loader:
    for i in range(len(y)):
        class_total[y[i]] += 1
for k in range(self.num_class):
    if class_total[k] == 0:
        class_total[k] = -1
for x, y in client.test_loader:
    x = x.to(FLAG.device)
    y = y.to(FLAG.device)
    out = client.model(x).view(-1, self.num_class)
    _, predicted = torch.max(out, 1)
    correct = (predicted == y).squeeze()
    if len(y) == 1:
        class_correct[y.item()] += correct.item()
    else:
        for k in range(len(y)):
            class_correct[y[k]] += correct[k].item()
```

图 4. 修改后的代码

最后，我还在 FedSoft 的复现任务中，额外加入了 Digits5 数据集的实验，编写了 Digits5 数据处理代码 generate-Digit5.py，将 Digits5 数据分配给联邦学习的 100 个客户端，进行软聚类联邦学习训练。

4.2 实验环境搭建

本文实验在 GPU 服务器上运行，使用 Nvidia Tesla P100 显卡进行计算，CUDA 版本为 10.2。使用的 python 环境包含：python = 3.6、pytorch = 1.10.0+cu102、torchvision = 0.11.0+cu102、Numpy = 1.18.5、Scikit-learn=0.24.2。

4.3 界面分析与使用说明

本文的复现工作，复现了 FedSoft 算法在基于 Cifar10、MNIST、EMNIST 和 Digits5 生成的 7 种数据集上的实验。具体为：

1. 由 Cifar10 和 Cifar10 旋转 90 度数据集生成本地数据的实验，运行 main-cifar-2set.py 文件。
2. MNIST 和 MNIST 90° 数据集的实验，运行 main-mnist-2set.py 文件。
3. MNIST、MNIST 90°、MNIST 180° 和 MNIST 180° 数据集的实验，运行 main-mnist-4set.py 文件。
4. 由 EMNIST 小写字母和 EMNIST 大写字母构成，来生成本地数据进行实验，运行 main-letters-2set.py 文件。
5. 由 EMNIST 小写字母、EMNIST 大写字母构成，并将其分别旋转 90°，组成 4 个数据集混合的实验，运行 main-letters-4set.py 文件。

6. 由 EMNIST 小写字母、EMNIST 大写字母构成，并将其分别旋转 90°、180°、270°，组成 8 个数据集混合的实验，运行 main-letters-8set.py 文件。

7. Digits5 数据集的实验（Digits5 数据集由 MNIST、MNIST-M、SYN、USPS、SVHN 混合而成），运行 main-digits5.py 文件。

使用 nohup python -u main-digits5.py > main-digits5.log 2>&1 将程序挂起，输出结果保存到 main-digits5.log 文件中。代码运行过程的输出如图 5所示，验证过程分为本地验证和簇模型验证。分别打印了每个类的验证准确率，对于本地模型验证而言，使用与本地训练集同分布的验证集进行测试；对于簇模型验证而言，使用平衡的原始的测试集进行验证。

```
********** Client 14 **********
Client 14 has average accuracy = 0.720, class_accuracy = [0.636, 0.733, 0.6, 0.5, 0.545, 0.5, 0.889, 0.909, 0.75, 0.875]
********** Client 15 **********
Client 15 has average accuracy = 0.660, class_accuracy = [0.857, 0.636, 0.429, 0.5, 0.75, 0.5, 0.769, 0.625, 0.778, 0.833]
********** Client 16 **********
Client 16 has average accuracy = 0.670, class_accuracy = [0.75, 0.833, 0.6, 0.444, 0.583, 0.333, 0.867, 0.636, 0.875, 0.75]
********** Client 17 **********
Client 17 has average accuracy = 0.570, class_accuracy = [0.688, 0.727, 0.625, 0.25, 0.375, 0.7, 0.714, 0.5, 0.75, 0.4]
********** Client 18 **********
Client 18 has average accuracy = 0.560, class_accuracy = [0.385, 0.667, 0.25, 0.385, 0.545, 0.8, 0.8, 0.6, 0.8, 0.615]
********** Client 19 **********
Client 19 has average accuracy = 0.710, class_accuracy = [0.643, 0.9, 0.6, 0.5, 0.667, 0.5, 0.667, 0.8, 1.0, 0.7]
********** Cluster 0 **********
Cluster 0 - test ds 0 has average accuracy = 0.715, class_accuracy = [0.763, 0.826, 0.557, 0.525, 0.67, 0.63, 0.813, 0.731, 0.831, 0.802]
Cluster 0 - test ds 1 has average accuracy = 0.731, class_accuracy = [0.754, 0.846, 0.581, 0.548, 0.692, 0.624, 0.829, 0.721, 0.875, 0.836]
********** Cluster 1 **********
Cluster 1 - test ds 0 has average accuracy = 0.719, class_accuracy = [0.749, 0.84, 0.596, 0.503, 0.649, 0.645, 0.84, 0.742, 0.85, 0.777]
Cluster 1 - test ds 1 has average accuracy = 0.728, class_accuracy = [0.762, 0.841, 0.607, 0.518, 0.635, 0.619, 0.849, 0.755, 0.864, 0.832]
```

图 5. FedSoft 迭代过程的验证结果

4.4 创新点

在后续复现中修正了这一错误本次实验的创新之处为，在作者原有的实验的基础之上，添加了一个混合数据集 Digits5（由 MNIST、MNIST-M、SYN、USPS、SVHN 混合而成）的实验，相比于对单个数据集进行旋转，Digits5 数据集更适合数据非独立同分布的实验。并且，找出了作者代码中一段非常关键的错误，在复现工作中修正了这一错误。

5 实验结果分析

FedSoft 算法在 7 种生成的数据集上的实验结果如表 1所示，数据集的细节在 4.3 节中已描述，其中论文中没有报告的结果用“-”表示。

从复现实验的结果来看，簇模型的准确率和本地模型的准确率都比原论文中的结果要低。特别是本地模型的测试准确率与原文的结果差距非常大，在 Cifar10-2set 数据集上差了 36.7%，在 Letters-2set 数据集上差了 21.3%。

出现测试结果相差非常大的这种问题，是因为在作者公开的代码中，作者使用本地训练集进行本地模型的测试，这是非常严重的错误，会导致本地测试的精度看起来非常高，即在 Cifar10-2set 数据集上达到了 98.8%。在修改了作者的代码后，即额外加入了独立的本地测试集，使用本地测试集进行本地测试，使得本地测试的复现比原文的结果差很多。

表 1. FedSoft 算法的复现结果。20 或 100 个联邦学习客户端，在完成 200 轮联邦迭代后的本地模型和簇模型的测试准确率。

	Cifar10-2set	Letters-2set	Letters-4set	MNIST-2set
簇模型准确率-原论文结果	78.9	73.0	70.6	-
簇模型准确率-复现结果	73.0	68.9	59.9	77.9
本地模型准确率-原论文结果	98.8	86.5	-	-
本地模型准确率-复现结果	62.1	65.2	77.4	66.9
	MNIST-4set	Letters-8set	Digits5	
簇模型准确率-原论文结果	-	-	-	
簇模型准确率-复现结果	66.8	40.0	83.7	
本地模型准确率-原论文结果	-	-	-	
本地模型准确率-复现结果	58.1	42.4	74.3	

6 总结与展望

FedSoft 提出了一种软聚类联邦学习方法，对客户端进行软聚类分组，并且在本地训练中，使用近端正则化更新，降低了客户端的工作量。该算法能有效地利用源分布之间的相似性来学习表现良好的本地个性化模型和簇模型。

通过广泛的复现实验表明，FedSoft 所提出的软聚类联邦学习方法，能够取得不错的效果。虽然作者在本地测试的代码中有一处关键性的错误，但是 FedSoft 方法依然是有效的，能够适用于联邦学习中的非独立同分布数据。

但对 FedSoft 的实验仅局限于简单的图像分类任务，在未来需进一步研究软聚类联邦学习在其他任务上的效果，例如文本预测、智能推荐等场景。

参考文献

[1] Christopher Briggs, Zhong Fan, and Peter Andras. Federated learning with hierarchical clustering of local updates to improve training on non-iid data. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–9. IEEE, 2020.

[2] Theodora S Brisimi, Ruidi Chen, Theofanie Mela, Alex Olshevsky, Ioannis Ch Paschalidis, and Wei Shi. Federated learning of predictive models from federated electronic health records. *International journal of medical informatics*, 112:59–67, 2018.

[3] Moming Duan, Duo Liu, Xinyuan Ji, Renping Liu, Liang Liang, Xianzhang Chen, and Yujuan Tan. Fedgroup: Efficient federated learning via decomposed similarity-based clustering. In *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pages 228–237. IEEE, 2021.

[4] Avishek Ghosh, Jichan Chung, Dong Yin, and Kannan Ramchandran. An efficient framework for clustered federated learning. *Advances in Neural Information Processing Systems*, 33:19586–19597, 2020.

[5] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

[6] Li Huang, Andrew L Shea, Huining Qian, Aditya Masurkar, Hao Deng, and Dianbo Liu. Patient clustering improves efficiency of federated machine learning to predict mortality and hospital stay time using distributed electronic medical records. *Journal of biomedical informatics*, 99:103291, 2019.

[7] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine learning and systems*, 2:429–450, 2020.

[8] Guodong Long, Ming Xie, Tao Shen, Tianyi Zhou, Xianzhi Wang, and Jing Jiang. Multi-center federated learning: clients clustering for better personalization. *World Wide Web*, 26(1):481–500, 2023.

[9] Yishay Mansour, Mehryar Mohri, Jae Ro, and Ananda Theertha Suresh. Three approaches for personalization with applications to federated learning. *arXiv preprint arXiv:2002.10619*, 2020.

[10] Othmane Marfoq, Giovanni Neglia, Aurélien Bellet, Laetitia Kameni, and Richard Vidal. Federated multi-task learning under a mixture of distributions. *Advances in Neural Information Processing Systems*, 34:15434–15447, 2021.

[11] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[12] Adnan Qayyum, Kashif Ahmad, Muhammad Ahtazaz Ahsan, Ala Al-Fuqaha, and Junaid Qadir. Collaborative federated learning for healthcare: Multi-modal covid-19 diagnosis at the edge. *IEEE Open Journal of the Computer Society*, 3:172–184, 2022.

[13] Yichen Ruan and Carlee Joe-Wong. Fedsoft: Soft clustered federated learning with proximal local updating. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8124–8131, 2022.

[14] Felix Sattler, Klaus-Robert Müller, and Wojciech Samek. Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints. *IEEE transactions on neural networks and learning systems*, 32(8):3710–3722, 2020.