FedAvg算法+LSTM模型+ Shakespeare数据集——字符预测任务

1. Shakespeare数据集介绍

任务：下一个字符预测

参数说明：总共4,226,15条样本，可使用官方给出的划分代码按照联邦学习场景下1129个client非独立同分布划分。

介绍: 和FEMNST一样，属于专门给联邦学习用的基准数据集leaf的成员之一。

官网：https://leaf.cmu.edu/

官方数据预处理与划分代码：https://github.com/TalwalkarLab

引用方式：LEAF: A Benchmark for Federated Settings

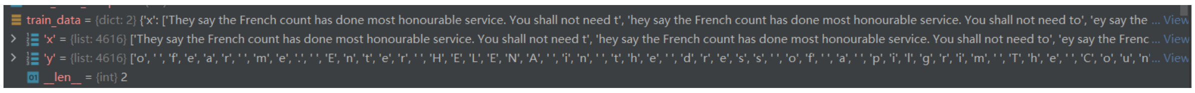
从 http://www.gutenberg.org/files/100/old/1994-01-100.zip 下载数据集解压到data文件夹下的raw_data文件夹，重命名为raw_data.txt

我们从https://github.com/TalwalkarLab下载数据集后，找到对应的Shakespeare数据集文件夹，按照README文件即可成果转换成我们想要的数据集，操作命令如下：

```
./preprocess.sh -s niid --sf 1.0 -k 0 -t sample -tf 0.8 (full-sized dataset)
./preprocess.sh -s niid --sf 0.2 -k 0 -t sample -tf 0.8 (small-sized dataset)
('-tf 0.8' reflects the train-test split used in the [FedAvg paper]
(https://arxiv.org/pdf/1602.05629.pdf))
```

至此，我们就能获得预处理后的非独立同分布的Shakespeare数据集，每个数据集下主要包括内容如下：

每个字符串x长度为80，预测下一个字符的输出，总共分为了有135个客户端数据



```
train_data = {dict: 2} {'x': ['They say the French count has done most honourable service. You shall not need t', 'hey say the French count has done most honourable service. You shall not need to', 'ey say the ... View
> 'x' = {list: 4616} ['They say the French count has done most honourable service. You shall not need t', 'hey say the French count has done most honourable service. You shall not need to', 'ey say the Frenc ... View
> 'y' = {list: 4616} ['o', ' ', 'f', 'e', 'a', 'r', ' ', 'm', 'e', '.', ' ', 'E', 'n', 't', 'e', 'r', ' ', 'H', 'E', 'L', 'E', 'N', 'A', ' ', 'i', 'n', ' ', 't', 'h', 'e', ' ', 'd', 'r', 'e', 's', 's', ' ', 'o', 'f', ' ', 'a', ' ', 'p', 'i', 'l', 'g', 'r', 'i', 'm', ' ', 'T', 'h', 'e', ' ', 'C', 'o', 'u', 'n'... View
  __len__ = {int} 2
```

2. LSTM训练模型

```python
class Model(nn.Module):
    def __init__(self, seed, lr, optimizer=None):
        super().__init__()
        self.lr = lr
        self.seed = seed
        self.optimizer = optimizer
        self.flops = 0
        self.size = 0

    def get_params(self):
        return self.state_dict()

    def set_params(self, state_dict):
        self.load_state_dict(state_dict)

    def __post_init__(self):
        if self.optimizer is None:
            self.optimizer = optim.SGD(self.parameters(), lr=self.lr)

    def train_model(self, data, num_epochs=1, batch_size=10):
        self.train()
        for batch in range(num_epochs):
            for batched_x, batched_y in batch_data(data, batch_size, seed=self.seed):
                self.optimizer.zero_grad()
                input_data = self.process_x(batched_x)
                target_data = self.process_y(batched_y)
                logits, loss = self.forward(input_data, target_data)
                loss.backward()
                self.optimizer.step()
        update = self.get_params()
        comp = num_epochs * (len(data['y']) // batch_size) * batch_size
        return comp, update

    def test_model(self, data):
        x_vecs = self.process_x(data['x'])
        labels = self.process_y(data['y'])
        self.eval()
        with torch.no_grad():
            logits, loss = self.forward(x_vecs, labels)
            acc = assess_fun(labels, logits)
        return {"accuracy": acc.detach().cpu().numpy(), 'loss': loss.detach().cpu().numpy()}


class LSTMModel(Model):
    def __init__(self, seed, lr, seq_len, num_classes, n_hidden):
        super().__init__(seed, lr)
        self.seq_len = seq_len
        self.num_classes = num_classes
        self.n_hidden = n_hidden
        self.word_embedding = nn.Embedding(self.num_classes, 8)
        self.lstm = nn.LSTM(input_size=8, hidden_size=self.n_hidden, num_layers=2, batch_first=True)
        self.pred = nn.Linear(self.n_hidden * 2, self.num_classes)
        self.loss_fn = nn.CrossEntropyLoss()
        super().__post_init__()

    def forward(self, features, labels):
        emb = self.word_embedding(features)
        output, (h_n, c_n) = self.lstm(emb)
        h_n = h_n.transpose(0, 1).reshape(-1, 2 * self.n_hidden)
        logits = self.pred(h_n)
        loss = self.loss_fn(logits, labels)
        return logits, loss

    def process_x(self, raw_x_batch):
        x_batch = [word_to_indices(word) for word in raw_x_batch]
        x_batch = torch.LongTensor(x_batch)
        return x_batch

    def process_y(self, raw_y_batch):
        y_batch = [letter_to_vec(c) for c in raw_y_batch]
        y_batch = torch.LongTensor(y_batch)
        return y_batch
```

3. Client模型

```python
class Client:
    def __init__(self, client_id, train_data, eval_data, model=None):
        self._model = model
        self.id = client_id
        self.train_data = train_data if train_data is not None else {'x': [], 'y': []}
        self.eval_data = eval_data if eval_data is not None else {'x': [], 'y': []}

    @property
    def model(self):
        return self._model

    @property
    def num_test_samples(self):
        if self.eval_data is None:
            return 0
        else:
            return len(self.eval_data['y'])

    @property
    def num_train_samples(self):
        if self.train_data is None:
            return 0
        else:
            return len(self.train_data['y'])

    @property
    def num_samples(self):
        train_size = 0
        if self.train_data is not None:
            train_size = len(self.train_data['y'])
        eval_size = 0
        if self.eval_data is not None:
            eval_size = len(self.eval_data['y'])
        return train_size + eval_size

    def train(self, num_epochs=1, batch_size=128, minibatch=None):
        if minibatch is None:
            data = self.train_data
            comp, update = self.model.train_model(data, num_epochs, batch_size)
        else:
            frac = min(1.0, minibatch)
            num_data = max(1, int(frac * len(self.train_data['y'])))
            xs, xy = zip(*random.sample(list(zip(self.train_data['x'], self.train_data['y'])), num_data))
            data = {
                'x': xs,
                'y': xy
            }
            num_epochs = 1
            comp, update = self.model.train_model(data, num_epochs, num_data)
        num_train_samples = len(data['y'])
        return comp, num_train_samples, update

    def test(self, set_to_use='test'):
        assert set_to_use in ['train', 'test', 'val']
        if set_to_use == 'train':
            data = self.train_data
        else:
            data = self.eval_data
        return self.model.test_model(data)
```

4. Serves模型

```
class Serves:
    def __init__(self, global_model):
        self.global_model = global_model
        self.model = global_model.get_params()
        self.selected_clients = []
        self.update = []

    def select_clients(self, my_round, possible_clients, num_clients=20):
        num_clients = min(num_clients, len(possible_clients))
        np.random.seed(my_round)
        self.selected_clients = np.random.choice(possible_clients, num_clients, replace=False)
        # return [(c.num_train_samples, c.num_test_samples) for c in self.selected_clients]

    def train_model(self, num_epochs=1, batch_size=10, minibatch=None, clients=None):
        if clients is None:
            clients = self.selected_clients
        sys_metrics = {
            c.id: {"bytes_written": 0,
                   "bytes_read": 0,
                   "local_computations": 0} for c in clients}
        for c in clients:
            c.model.set_params(self.model)
            comp, num_samples, update = c.train(num_epochs, batch_size, minibatch)
            sys_metrics[c.id]["bytes_read"] += c.model.size
            sys_metrics[c.id]["bytes_written"] += c.model.size
            sys_metrics[c.id]["local_computations"] = comp

            self.update.append((num_samples, update))
        return sys_metrics

    def aggregate(self, updates):
        avg_param = OrderedDict()
        total_weight = 0.
        for (client_samples, client_model) in updates:
            total_weight += client_samples
```

```python
            for name, param in client_model.items():
                if name not in avg_param:
                    avg_param[name] = client_samples * param
                else:
                    avg_param[name] += client_samples * param

        for name in avg_param:
            avg_param[name] = avg_param[name] / total_weight
        return avg_param

    def update_model(self):
        avg_param = self.aggregate(self.update)
        self.model = avg_param
        self.global_model.load_state_dict(self.model)
        self.update = []

    def test_model(self, clients_to_test=None, set_to_use='test'):
        metrics = {}
        if clients_to_test is None:
            clients_to_test = self.selected_clients

        for client in tqdm(clients_to_test):
            client.model.set_params(self.model)
            c_metrics = client.test(set_to_use)
            metrics[client.id] = c_metrics

        return metrics

    def get_clients_info(self, clients):
        if clients is None:
            clients = self.selected_clients

        ids = [c.id for c in clients]
        num_samples = {c.id: c.num_samples for c in clients}
        return ids, num_samples

    def save_model(self, path):
        """Saves the server model on checkpoints/dataset/model.ckpt."""
        return torch.save({"model_state_dict": self.model}, path)
```

5. 数据处理工具函数

```python
import json
import numpy as np
import os
from collections import defaultdict

import torch

ALL_LETTERS = "\n !\"&'(),-.0123456789:;>?
ABCDEFGHIJKLMNOPQRSTUVWXYZ[]abcdefghijklmnopqrstuvwxyz}"
NUM_LETTERS = len(ALL_LETTERS)


def batch_data(data, batch_size, seed):
    data_x = data['x']
    data_y = data['y']

    np.random.seed(seed)
    rng_state = np.random.get_state()
    np.random.shuffle(data_x)
    np.random.set_state(rng_state)
    np.random.shuffle(data_y)

    for i in range(0, len(data_x), batch_size):
        batched_x = data_x[i:i + batch_size]
        batched_y = data_y[i:i + batch_size]
        yield (batched_x, batched_y)


def assess_fun(y_true, y_hat):
    y_hat = torch.argmax(y_hat, dim=-1)
    total = y_true.shape[0]
    hit = torch.sum(y_true == y_hat)
    return hit.data.float() * 1.0 / total


def word_to_indices(word):
    indices = []
    for c in word:
        indices.append(ALL_LETTERS.find(c))
    return indices


def letter_to_vec(letter):
    index = ALL_LETTERS.find(letter)
    return index
```

6. 数据读取函数

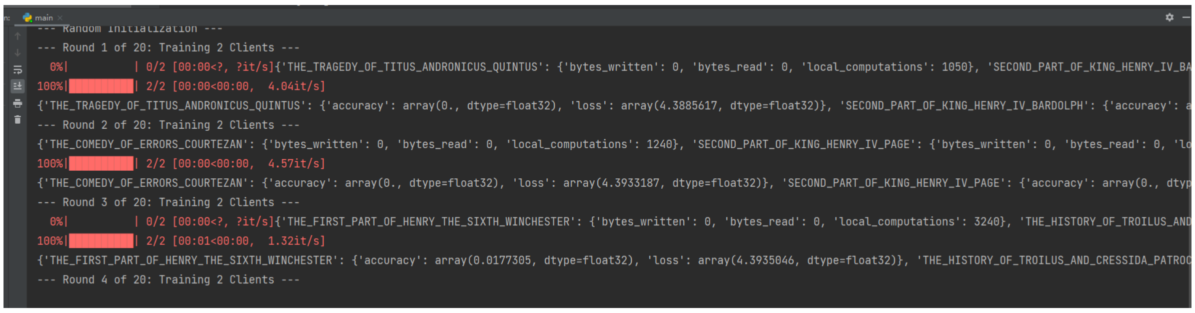
```python
def read_dir(data_dir):
    clients = []
    data = defaultdict(lambda: None)

    files = os.listdir(data_dir)
    files = [f for f in files if f.endswith('.json')]
    for f in files:
        file_path = os.path.join(data_dir, f)
        with open(file_path, 'r') as inf:
            cdata = json.load(inf)
        clients.extend(cdata['users'])
        data.update(cdata['user_data'])

    clients = list(sorted(data.keys()))
    return clients, data


def read_data(train_data_dir, test_data_dir):
    train_clients, train_data = read_dir(train_data_dir)
    test_clients, test_data = read_dir(test_data_dir)

    assert train_clients == test_clients

    return train_clients, train_data, test_data


def create_clients(users, train_data, test_data, model):
    clients = [Client(u, train_data[u], test_data[u], model) for u in users]
    return clients


def setup_clients(model=None, use_val_set=False):
    eval_set = 'test' if not use_val_set else 'test'
    train_data_dir = os.path.join('.', 'data', 'train')
    test_data_dir = os.path.join('.', 'data', eval_set)
    users, train_data, test_data = read_data(train_data_dir, test_data_dir)

    clients = create_clients(users, train_data, test_data, model)

    return clients
```

7. 训练函数

```python
def train():
    seed = 1
    random.seed(1 + seed)
    np.random.seed(12 + seed)
    torch.manual_seed(123 + seed)
    torch.manual_seed(123 + seed)
    lr = 0.0003
    seq_len = 80
    num_classes = 80
    n_hidden = 256
    num_rounds = 20
    eval_every = 1
    clients_per_round = 2
    num_epochs = 1
    batch_size = 10
    minibatch = None
    use_val_set = 'test'
    # 全局模型(服务端)
    global_model = LSTMModel(seed, lr, seq_len, num_classes, n_hidden)
    # 服务端
    server = Serves(global_model)
    # 客户端
    client_model = LSTMModel(seed, lr, seq_len, num_classes, n_hidden)
    clients = setup_clients(client_model, use_val_set)
    client_ids, client_num_samples = server.get_clients_info(clients)
    print(('Clients in Total: %d' % len(clients)))
    print('--- Random Initialization ---')
    # Simulate training
    for i in range(num_rounds):
        print('--- Round %d of %d: Training %d Clients ---' % (i + 1,
num_rounds, clients_per_round))

        # Select clients to train this round
        server.select_clients(i, clients, num_clients=clients_per_round)

        c_ids, c_num_samples = 
server.get_clients_info(server.selected_clients)

        # Simulate server model training on selected clients' data
        sys_metrics = server.train_model(num_epochs=num_epochs, 
batch_size=batch_size,
                                          minibatch=minibatch)
        print(sys_metrics)
        # sys_writer_fn(i + 1, c_ids, sys_metrics, c_num_samples)
        metrics = server.test_model()
        print(metrics)

        # Update server model
        server.update_model()
```

8. 训练结果

因为设备原因，暂时无法训练出论文中的模型



```
--- Random initialization ---
--- Round 1 of 20: Training 2 Clients ---
  0%|          | 0/2 [00:00<?, ?it/s]{'THE_TRAGEDY_OF_TITUS_ANDRONICUS_QUINTUS': {'bytes_written': 0, 'bytes_read': 0, 'local_computations': 1050}, 'SECOND_PART_OF_KING_HENRY_IV_BA
100%|██████████| 2/2 [00:00<00:00,  4.04it/s]
{'THE_TRAGEDY_OF_TITUS_ANDRONICUS_QUINTUS': {'accuracy': array(0., dtype=float32), 'loss': array(4.3885617, dtype=float32)}, 'SECOND_PART_OF_KING_HENRY_IV_BARDOLPH': {'accuracy': a
--- Round 2 of 20: Training 2 Clients ---
{'THE_COMEDY_OF_ERRORS_COURTEZAN': {'bytes_written': 0, 'bytes_read': 0, 'local_computations': 1240}, 'SECOND_PART_OF_KING_HENRY_IV_PAGE': {'bytes_written': 0, 'bytes_read': 0, 'lo
100%|██████████| 2/2 [00:00<00:00,  4.57it/s]
{'THE_COMEDY_OF_ERRORS_COURTEZAN': {'accuracy': array(0., dtype=float32), 'loss': array(4.3933187, dtype=float32)}, 'SECOND_PART_OF_KING_HENRY_IV_PAGE': {'accuracy': array(0., dtyp
--- Round 3 of 20: Training 2 Clients ---
  0%|          | 0/2 [00:00<?, ?it/s]{'THE_FIRST_PART_OF_HENRY_THE_SIXTH_WINCHESTER': {'bytes_written': 0, 'bytes_read': 0, 'local_computations': 3240}, 'THE_HISTORY_OF_TROILUS_AND
100%|██████████| 2/2 [00:01<00:00,  1.32it/s]
{'THE_FIRST_PART_OF_HENRY_THE_SIXTH_WINCHESTER': {'accuracy': array(0.0177305, dtype=float32), 'loss': array(4.3935046, dtype=float32)}, 'THE_HISTORY_OF_TROILUS_AND_CRESSIDA_PATROC
--- Round 4 of 20: Training 2 Clients ---
```