联邦学习是一种数据保留在本地，多个客户端协同训练模型而不分享数据，最终对多个客户端训练的模型进行聚合的学习机制。尽管联邦学习避免了将数据直接暴露给第三方，但是其中依然存在大量隐私泄露的风险。比如：（1）我们不能保证所有的客户机参与方都是可靠的，不可靠的参与方增加了隐私泄露的风险；（2）联邦学习需要把本地训练好的模型参数上传至中心服务器，攻击者可以通过模型参数反推出用户信息；（3）训练完成的模型也面临着隐私泄露的风险。针对联邦学习所面临的隐私风险，目前有两种解决思路，分别是加密和扰动。加密最常用的方法是利用同态加密和秘密分享技术为联邦训练过程中数据的传输提供隐私保护，这两种技术是常用的密码学工具，但因为其计算代价高，会带来通信开销大的问题。扰动就是通过差分隐私等技术在本地数据或模型训练过程产生的梯度数据上添加噪声，使发布的模型在保持可用性的同时得到保护。

看差分隐私的介绍中总是会看到 k 匿名化和差分隐私做对比，说差分隐私多么好。去标识化容易受到链接攻击和差分攻击，这些说明其实都很简单，但是我自己从来没有代码实现过。自己实现一遍感觉理解得更深刻了。

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
adult = pd.read_csv("adult_with_pii.csv")
print(adult.head())
# 去标识化
adult_data = adult.copy().drop(columns=['Name', 'SSN'])
print(adult_data.head(1))
adult_pii = adult[['Name', 'SSN', 'DOB', 'Zip']]
print(adult_pii.head(1))
#链接攻击
karries_row = adult_pii[adult_pii['Name'] == 'Karrie Trusslove']
print(pd.merge(karries_row, adult_data, left_on=['DOB', 'Zip'],
right_on=['DOB', 'Zip']))
# 差分攻击
adult['Age'].sum()
adult[adult['Name'] != 'Karrie Trusslove']['Age'].sum()
print(adult['Age'].sum() - adult[adult['Name'] != 'Karrie
Trusslove']['Age'].sum())
```

```
D:\Anaconda3\envs\pytorch1\python.exe D:\pythonProject2\py3.py
               Name          SSN       DOB  ...  Age Capital Gain Capital Loss
0  Karrie Trusslove  732-14-6110  9/7/1967  ...   56         2174            0

[1 rows x 18 columns]
56
```

总结

链接攻击涉及将辅助数据与去识别化数据相结合，以重新识别个人身份。在最简单的情况下，可以通过包含这些数据集的两个表的联接来执行链接攻击。而差分攻击即使在聚合统计信息位于非常大的组上，也可以继续进行。那么

1、发布有用的数据使得确保隐私变得非常困难。

2、我们无法区分恶意和非恶意查询。

差分隐私

定义：
满足差分隐私的函数通常称为机制。我们说，如果对于所有相邻数据集x和x′以及所有可能的输出S，

$$\frac{\Pr[F(x) = S]}{\Pr[F(x') = S]} \leq e^{\epsilon}$$

定义中的 ϵ 参数称为隐私参数或隐私预算。ϵ 提供了一个旋钮来调整定义提供的"隐私量"。当 ϵ 的值较小时，要求 F FF 在给定相似的输入时提供非常相似的输出，因此提供更高级别的隐私;当 ϵ 的值较大时，允许输出中的相似性降低，因此提供的隐私较少。

拉普拉斯机制

差分隐私通常用于回答特定查询。

对数据集进行查询

```
print(adult[adult['Age'] >= 60].shape[0])
```

```
6957
```

实现此查询的差分隐私的最简单方法是在其答案中添加随机噪声。关键的挑战是添加足够的噪声来满足差分隐私的定义，但又不能过多地使答案变得过于嘈杂而无用。为了使这个过程更容易，在差分隐私领域已经开发了一些基本机

制，这些机制准确地描述了使用哪种噪声以及使用多少噪声。其中之一被称为拉普拉斯机制。

根据拉普拉斯机制，对于返回数字的函数$f(x)$，以下$F(x)$定义满足ϵ-差分隐私：

$$F(x) = f(x) + Lap(\frac{s}{\epsilon})$$

其中s是f的敏感度，Lap（S）表示从中心为 0 且比例为S的拉普拉斯分布中采样。

```
sensitivity = 1
epsilon = 0.1
print(adult[adult['Age'] >= 60].shape[0] + np.random.laplace(loc=0,
scale=sensitivity/epsilon))
```

多次运行此代码来查看噪声的影响

发现每次的输出都会发生变化，但大多数时候，答案与 6957 都足够接近

那么多少噪声就足够了？

我们怎么知道拉普拉斯机制增加了足够的噪声来防止数据集中个体的重新识别？

```
karries_row = adult[adult['Name'] == 'Karrie Trusslove']
print(karries_row[karries_row['Target'] == '<=50K'].shape[0])
```

发现真实情况为 1，这个人的收入小于 50k

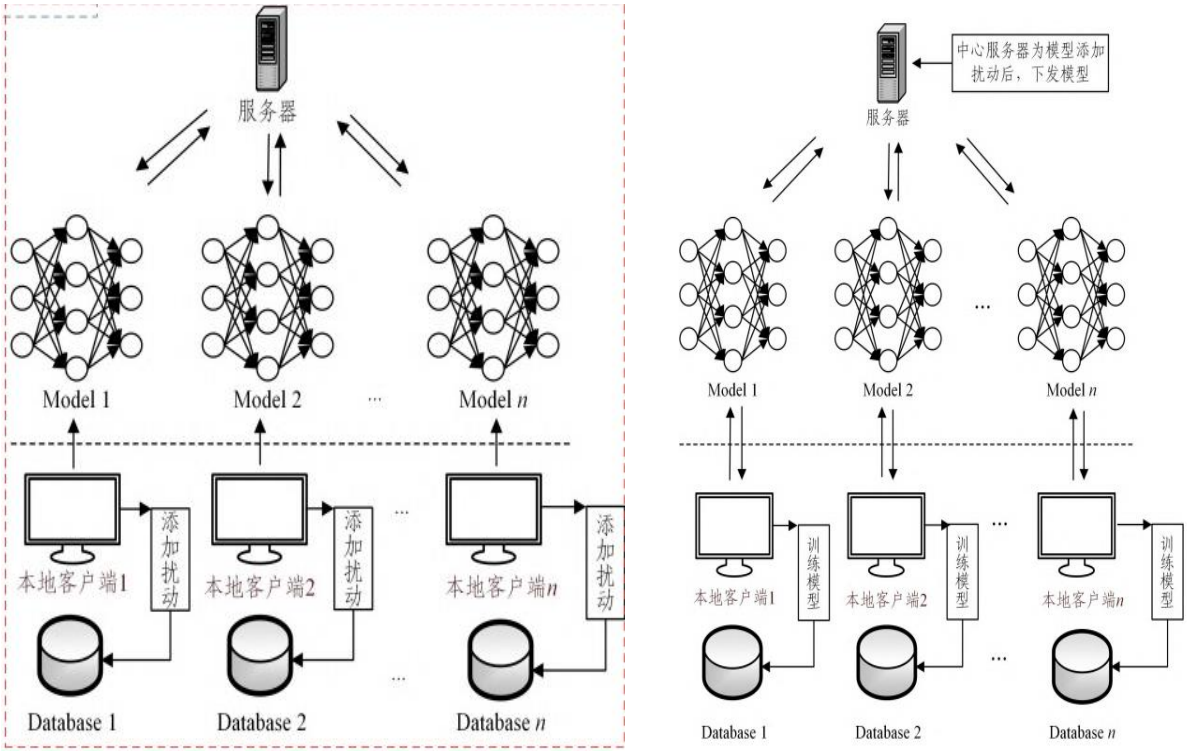
```
sensitivity = 1
epsilon = 0.1
karries_row = adult[adult['Name'] == 'Karrie Trusslove']
for i in range(1,5):
    print(karries_row[karries_row['Target'] == '<=50K'].shape[0] +
np.random.laplace(loc=0, scale=sensitivity/epsilon))
```

```
1
-2.0662400118647906
-15.826995041997549
-18.857033686028647
22.678895607818376
```

真正的答案是 0 还是 1？噪音太大，无法可靠地分辨出来。

这就是差分隐私拉普拉斯机制的工作方式，它不会拒绝被确定为恶意的查询，而是增加了足够的噪音，使恶意查询的结果没有用处。

中心化差分隐私和本地化差分隐私



前面所提到的模型只考虑了差分隐私的中心模型，其中敏感数据被集中收集在单个数据集中。在此设置中，我们假设分析师是恶意的，但有一个受信任的数据管理者持有数据集并正确执行分析师指定的差分隐私机制。但此设置通常不现实。在许多情况下，数据管理者和分析师是相同的，实际上没有受信任的第三方来保存数据和执行机制。事实上，收集最敏感数据的组织往往正是我们不信任的组织，这样的组织当然不能充当受信任的数据管理者。差分隐私的中心模型的替代方案是差分隐私的本地模型，其中数据在离开数据主体的控制之前变得满足差分隐私。例如，你可能会在将设备上的数据发送给数据管理机构之前，先向数据添加噪音。在本地模型中，数据管理者不需要信任，因为它们收集的数据已经满足了差分隐私。因此，与中心模型相比，本地模型具有一个巨大的优势：数据主体除了他们自己之外，不需要信任其他人。但是本地模型也有一个明显的缺点：在中心差分隐私下，对于与相同查询相同的隐私成本，本地模型中查询结果的准确性通常要低几个数量级。

```
adult = pd.read_csv("adult_with_pii.csv")
def laplace_mech(v, sensitivity, epsilon):
```

```
    return v + np.random.laplace(loc=0, scale=sensitivity / epsilon)
def pct_error(orig, priv):
    return np.abs(orig - priv)/orig * 100.0


def rand_resp_sales(response):
    truthful_response = response == 'Sales'

    # first coin flip
    if np.random.randint(0, 2) == 0:
        # answer truthfully
        return truthful_response
    else:
        # answer randomly (second coin flip)
        return np.random.randint(0, 2) == 0
print(pd.Series([rand_resp_sales('Sales') for i in
range(200)]).value_counts())
responses = [rand_resp_sales(r) for r in adult['Occupation']]
print(pd.Series(responses).value_counts())
len(adult[adult['Occupation'] == 'Sales'])
responses = [rand_resp_sales(r) for r in adult['Occupation']]
fake_yesses = len(responses)/4
num_yesses = np.sum([1 if r else 0 for r in responses])
true_yesses = num_yesses - fake_yesses
rr_result = true_yesses*2
true_result = np.sum(adult['Occupation'] == 'Sales')
print(pct_error(true_result, rr_result))
print(pct_error(true_result, laplace_mech(true_result, 1, 1)))
```

```
2.1232876712328768
0.013354828736986723
```

使用这种方法，相当大的计数（例如，在这种情况下超过 3000），我们通常会得到"可接受"的错误率低于 5%。但是，当计数较小时，误差将很快变大。

在这里，我们得到的误差约为 0.01%，即使我们对中心模型的 ϵ 值略低于我们用于随机响应的 ϵ 值。

本地模型有更好的算法，但是在提交数据之前必须添加噪声的固有局限性意味着本地模型算法的准确性始终比中心模型算法差。

客户端实现 DP-FedAvg 算法

```
def local_train(self, model):
    for name, param in model.state_dict().items():
        self.local_model.state_dict()[name].copy_(param.clone())

    optimizer = torch.optim.SGD(self.local_model.parameters(),
lr=self.conf['lr'],momentum=self.conf['momentum'])
    self.local_model.train()
    for e in range(self.conf["local_epochs"]):

        for batch_id, batch in enumerate(self.train_loader):
            data, target = batch
            if torch.cuda.is_available():
                data = data.cuda()
                target = target.cuda()

            optimizer.zero_grad()
            output = self.local_model(data)
            loss = torch.nn.functional.cross_entropy(output, target)
            loss.backward()
            optimizer.step()
            if self.conf["dp"]:
                model_norm = models.model_norm(model, self.local_model)
                norm_scale = min(1, self.conf['C'] / (model_norm))
                for name, layer in self.local_model.named_parameters():
                    clipped_difference = norm_scale * (layer.data -
model.state_dict()[name])
                    layer.data.copy_(model.state_dict()[name] +
clipped_difference)

        print("Epoch %d done." % e)
    diff = dict()
    for name, data in self.local_model.state_dict().items():
        diff[name] = (data - model.state_dict()[name])

    return diff
```

通过调整本地模型的参数来实现的，以确保它们与全局模型（或初始模型）之间的差异保持在一定的范围内

具体来说，相比于常规的本地训练，其主要修改点是在每一轮的梯度下降结束后，对参数进行裁剪