城市计算与跨域学习联合建模

Urban computing and Cross-domain learning joint modeling

董天文
East China Normal University
2019.3.28

上海市高可信计算重点实验室
Shanghai Key Laboratory of Trustworthy Computing

城市计算（Urban Computing）

- 定义
- 城市时空大数据的挖掘与学习
- 大数据下的挑战
- 应用

跨域学习联合建模

- 联合建模框架
- 联邦学习在联合建模中的应用

上海市高可信计算重点实验室

Shanghai Key Laboratory of Trustworthy Computing


城市计算（Urban Computing）
定义
城市时空大数据的挖掘与学习
应用
城市计算面临的挑战

城市计算（Urban Computing）

城市数据采集、管理、分析挖掘和服务提供

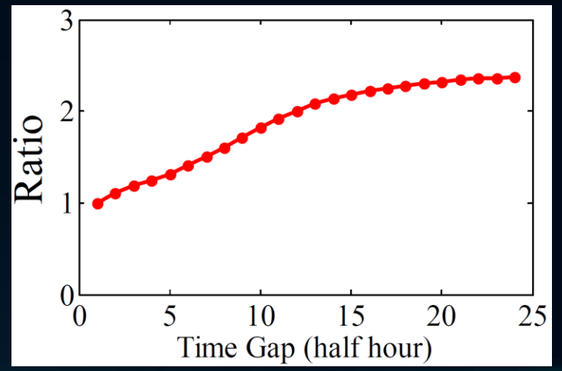
交通、规划、环境、能耗、公共安全、商业、医疗等痛点

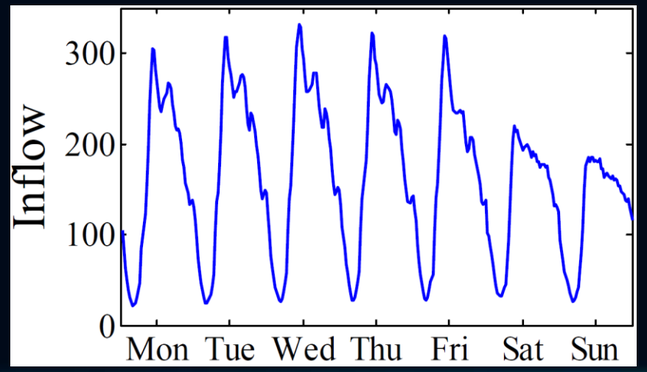
云计算+大数据+AI+城市场景

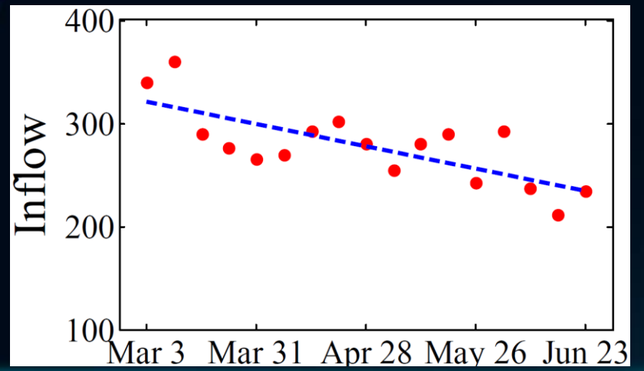
城市数据有其独特性

- 数据异构、多源
- 时空动态

城市大数据（时间属性）

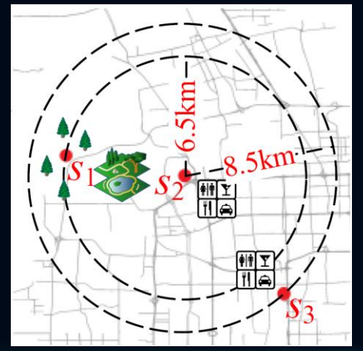


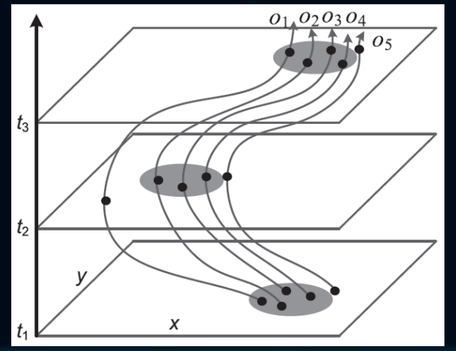




时间连续性　　周期性　　趋势性

城市大数据（空间属性）





空间距离

- 空间相关性

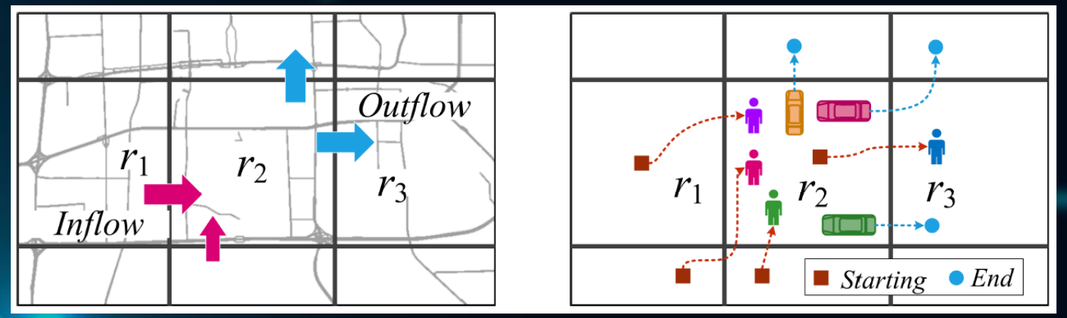
空间层次

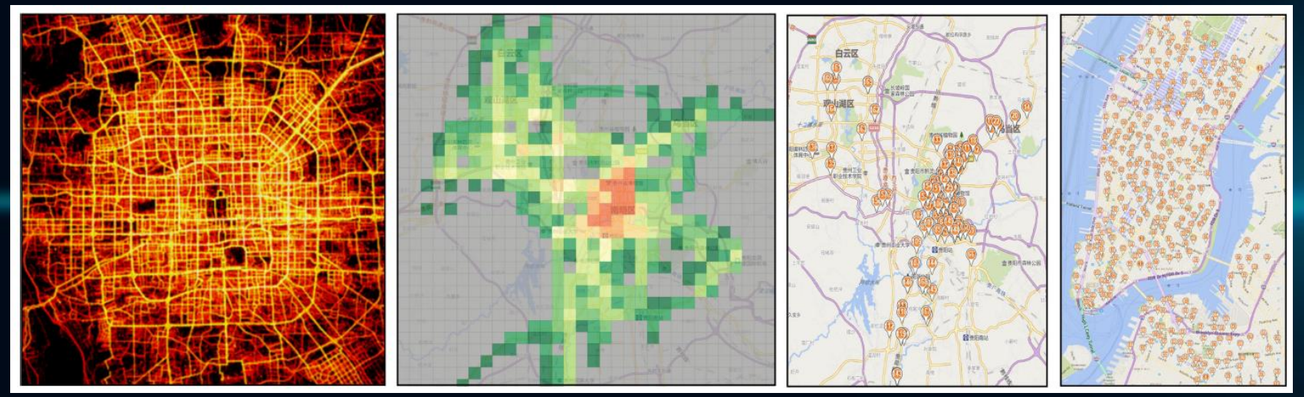
- 不同的空间粒度
- 城市结构

城市计算的应用场景

DNN-Based Urban Flow Prediction

预测每个区域的人群在整个活动期间的下一个时间间隔内的流入和流出

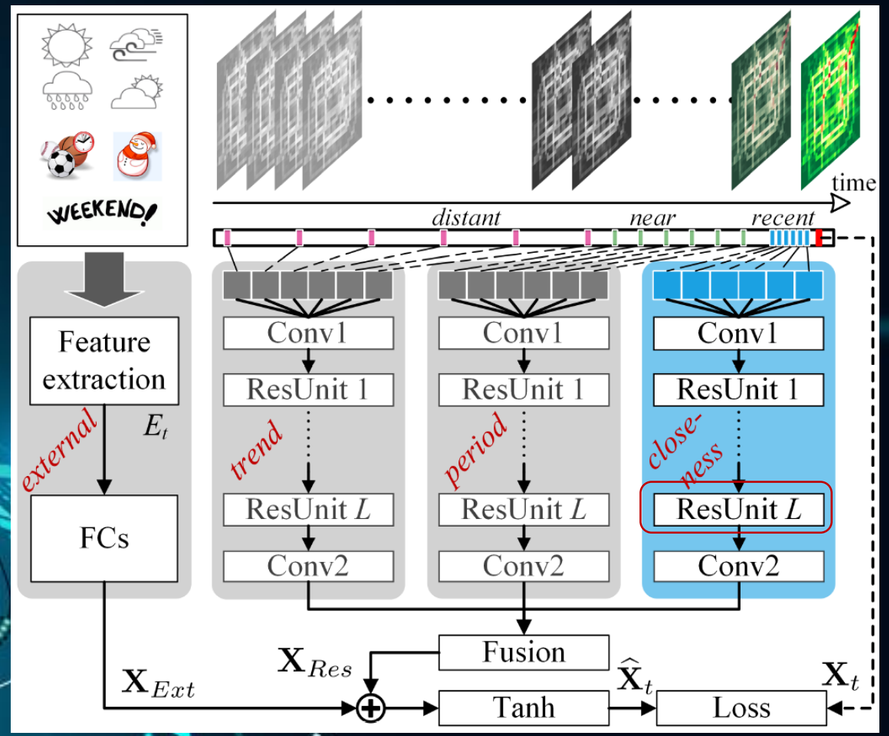




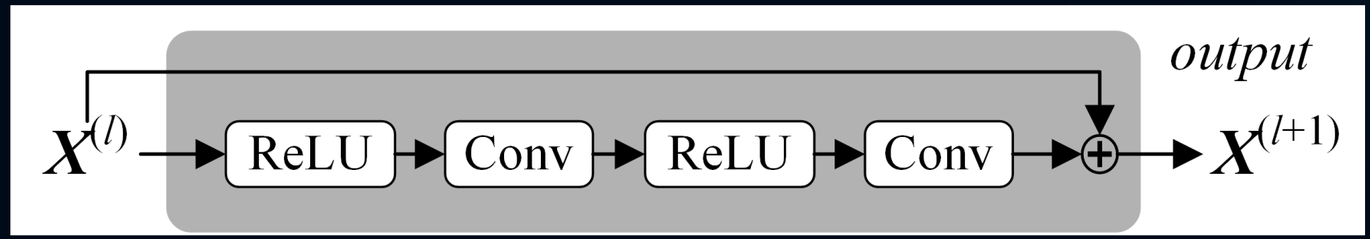
交通管理、风险估计、公共安全

城市计算的应用场景

ST-ResNet: A Collective Prediction

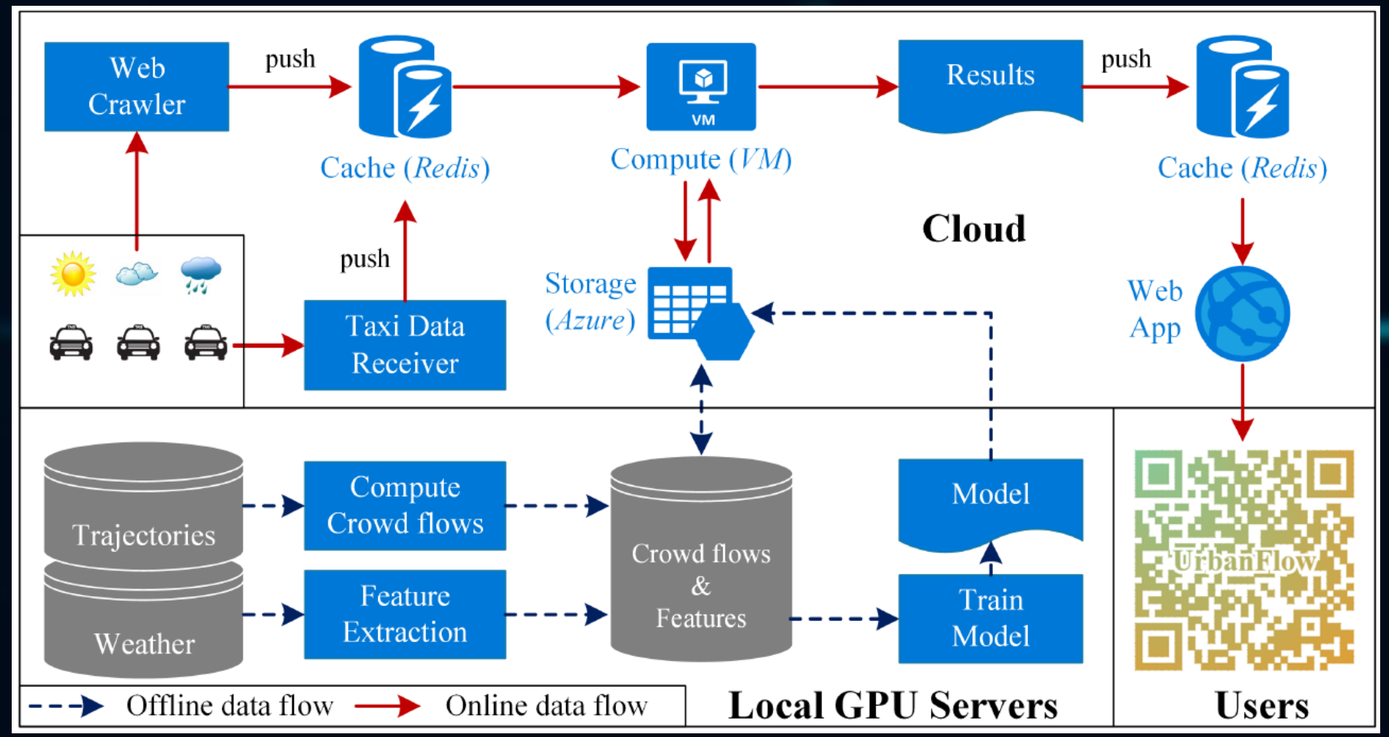


- 群体预测
- 捕捉时间连续性，周期性，趋势
- 捕捉外部因素
- 捕捉近距离与远距离的空间相关性
- 不同地区的融合因素不同



城市计算的应用场景

ST-ResNet: A Collective Prediction



城市计算面临的挑战

跨平台场景复杂

- 数据孤岛
- 不同政府机关和企事单位数据平台架构不同
- 数据加密等级多样
- 数据类型及标准多样

现有模型算法

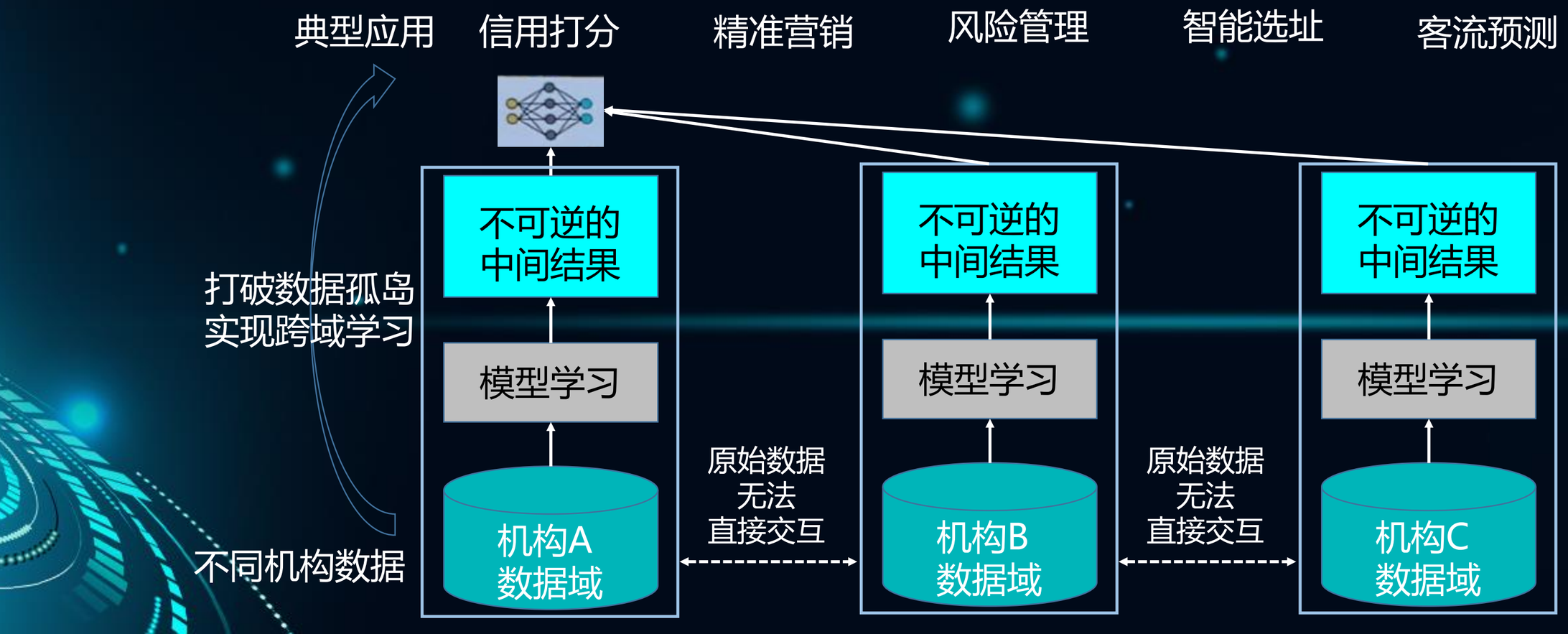
- 如何保护原始数据不被泄露
- 如何保证模型的准确度和模型效率
- 网络安全问题
- 数据类型及标准多样

跨域学习联合建模

联合建模框架

联邦学习在联合建模中的应用

跨域学习联合建模框架

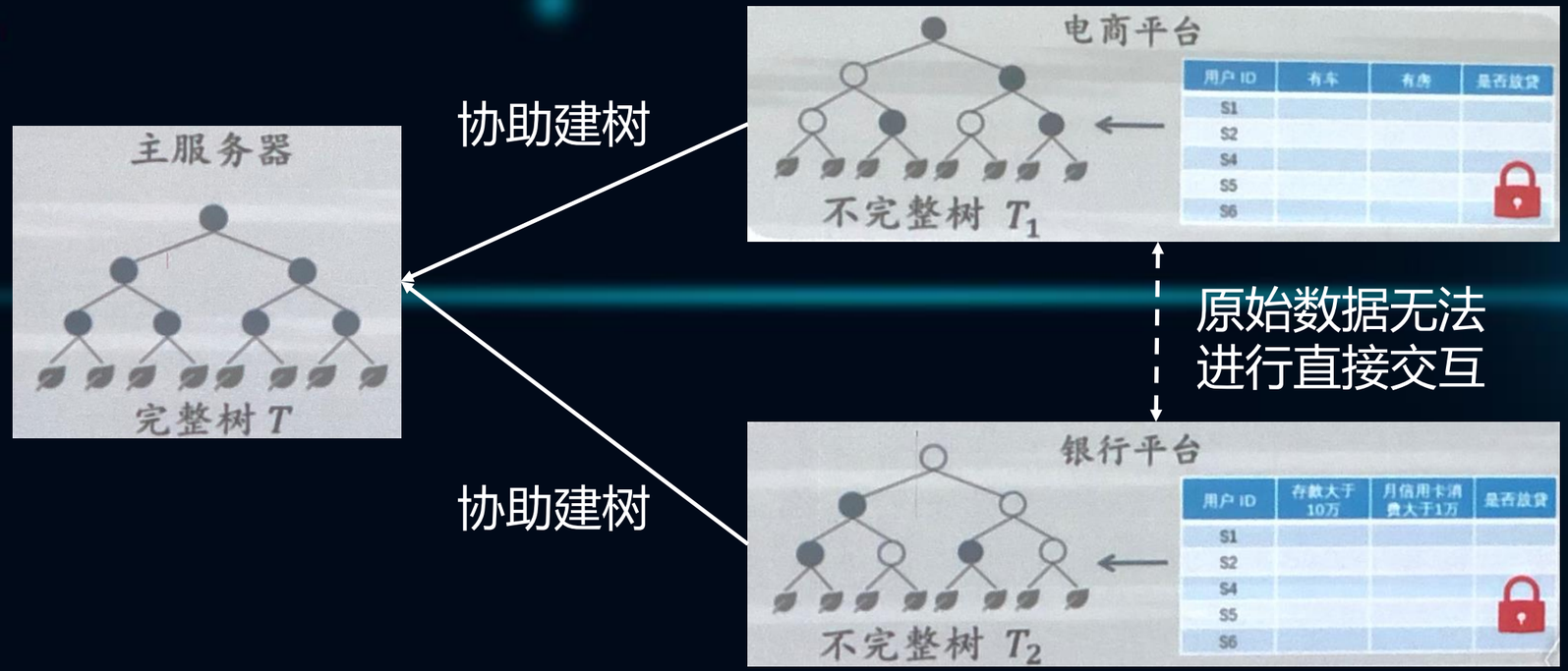

典型应用
信用打分
精准营销
风险管理
智能选址
客流预测
打破数据孤岛
实现跨域学习
不同机构数据
不可逆的
中间结果
模型学习
机构A
数据域
原始数据
无法
直接交互
不可逆的
中间结果
模型学习
机构B
数据域
原始数据
无法
直接交互
不可逆的
中间结果
模型学习
机构C
数据域

联邦学习联合建模应用

联邦学习+随机森林->联邦随机森林

利用机构A和机构B的数据对客户放贷进行决策分析

- 安全&隐私保护
- 模型精度无损
- 效率高
- 可拓展性强



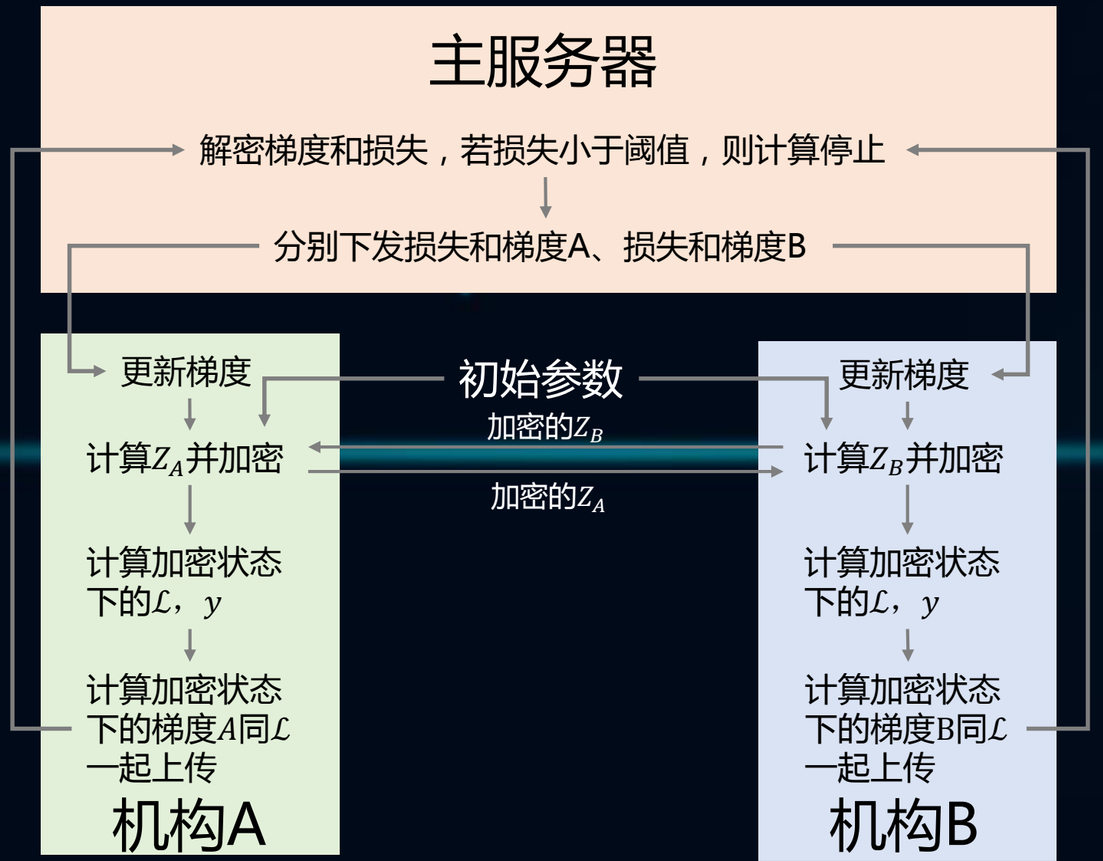
联邦学习联合建模应用

联邦学习+逻辑回归->联合企业信用评估

双机构共同完成企业信用评级

- 安全&隐私保护
- 模型精度无损
- 效率高
- 可拓展性强

公司名称
营业时长
收入等级
企业影响力
风险用户风险值
…

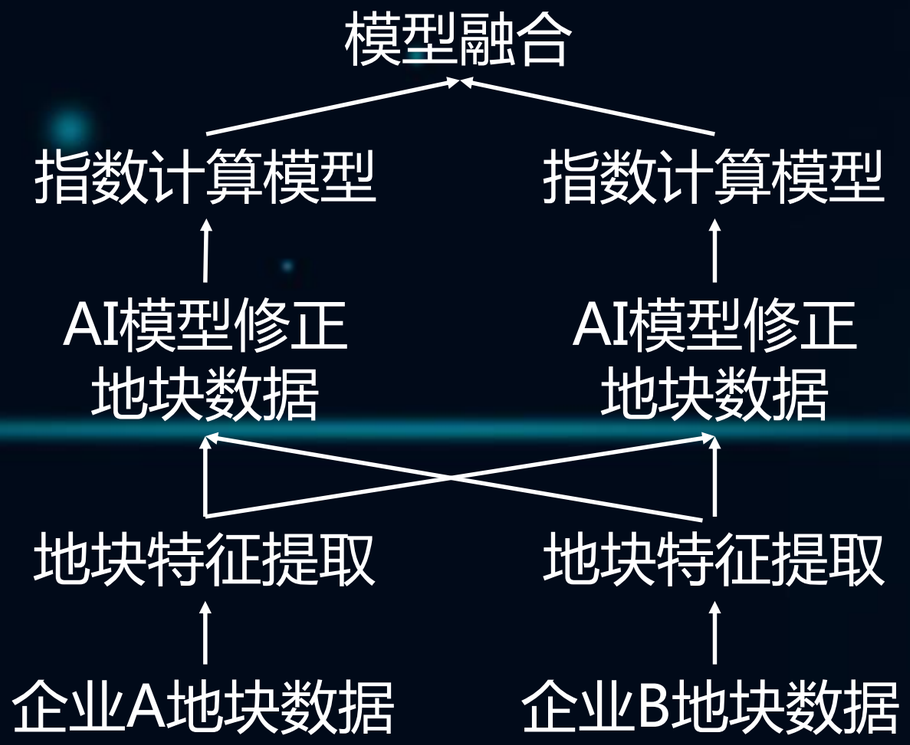


公司名称
所属行业及类型
企业状态
注册资金
企业异常信息
…

联邦学习联合建模应用

联邦学习+地块特征学习->智能地块排序

- 统一地块ID唯一标准
- 地块特征修正
- 可实施性强
- 可拓展性强



TTF框架自定义联邦算法

Federated Core简介

实现联合平均

Federated Core简介

TFF的一个显着特点是允许在联合数据上紧凑地表达基于TensorFlow的计算，其基本组合单元是**联合计算**——可以接受联合值作为输入并将联合值作为输出返回逻辑部分。

将所有设备上的整个数据项集合（例如，分布式阵列中所有传感器的温度读数集合）建模为单个联合值。

```
1 federated_float_on_clients = tff.FederatedType(tf.float32, tff.CLIENTS)
```

```
1 str(federated_float_on_clients)
```

```
'{float32}@CLIENTS'
```

Federated Core简介

如果每个传感器不仅保持一个温度读数，而且保持多个。
TFF计算使用tf.data.Dataset.reduce运算符计算单个本地数据集中的平均温度。

```
@tff.tf_computation(tff.SequenceType(tf.int32))
def foo(x):
  return x.reduce(np.int32(0), lambda x, y: x + y)

foo([1, 2, 3])
```

```
6
```

假设有一组传感器，每个传感器都有一个本地温度读数序列。我们可以通过平均传感器的局部平均值来计算全局温度平均值。

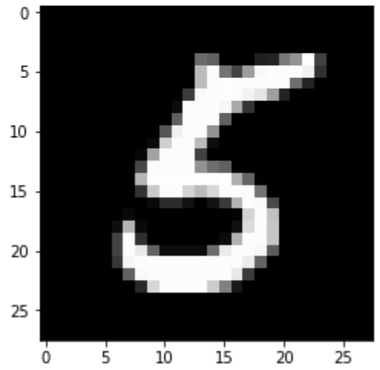
```
@tff.federated_computation(
    tff.FederatedType(tff.SequenceType(tf.float32), tff.CLIENTS))
def get_global_temperature_average(sensor_readings):
  return tff.federated_average(
      tff.federated_map(get_local_temperature_average, sensor_readings))
```

联合平均的实现

Local training and evaluation

```
#@test {"output": "ignore"}
from matplotlib import pyplot as plt

plt.imshow(federated_train_data[5][-1]['x'][-1].reshape(28, 28), cmap='gray')
plt.grid(False)
plt.show()
```




```
locally_trained_model = local_train(initial_model, 0.1, federated_train_data[5])
```

```
#@test {"output": "ignore"}
print('initial_model loss =', local_eval(initial_model, federated_train_data[5]))
print('locally_trained_model loss =', local_eval(locally_trained_model, federated_train_data[5]))
```

```
initial_model loss = 23.025854
locally_trained_model loss = 0.4348469
```

```
#@test {"output": "ignore"}
print('initial_model loss =', local_eval(initial_model, federated_train_data[0]))
print('locally_trained_model loss =', local_eval(locally_trained_model, federated_train_data[0]))
```

```
initial_model loss = 23.025854
locally_trained_model loss = 74.500755
```

联合平均的实现

Federated evaluation

```
@tff.federated_computation(SERVER_MODEL_TYPE, CLIENT_DATA_TYPE)
def federated_eval(model, data):
  return tff.federated_average(
      tff.federated_map(local_eval, [tff.federated_broadcast(model), data]))
```

将模型分发给客户端，让每个客户端调用其本地数据部分的本地evaluation，然后平均loss。

```
#@test {"output": "ignore"}
print('initial_model loss =', federated_eval(initial_model, federated_train_data))
print('locally_trained_model loss =', federated_eval(locally_trained_model, federated_train_data))
```

```
initial_model loss = 23.025852
locally_trained_model loss = 54.43263
```

正如预料的那样，损失增加了。
因此为了改善所有用户的模型，需要培训每个人的数据。

联合平均的实现

Federated training

```
SERVER_FLOAT_TYPE = tff.FederatedType(tf.float32, tff.SERVER, all_equal=True)

@tff.federated_computation(
    SERVER_MODEL_TYPE, SERVER_FLOAT_TYPE, CLIENT_DATA_TYPE)
def federated_train(model, learning_rate, data):
  return tff.federated_average(
      tff.federated_map(
          local_train,
          [tff.federated_broadcast(model),
           tff.federated_broadcast(learning_rate),
           data]))
```

```
#@test {"timeout": 600, "output": "ignore"}
model = initial_model
learning_rate = 0.1
for round_num in range(5):
  model = federated_train(model, learning_rate, federated_train_data)
  learning_rate = learning_rate * 0.9
  loss = federated_eval(model, federated_train_data)
  print('round {}, loss={}'.format(round_num, loss))
```

```
round 0, loss=21.605525970458984
round 1, loss=20.365678787231445
round 2, loss=19.27480125427246
round 3, loss=18.31110954284668
round 4, loss=17.45725440979004
```

```
#@test {"output": "ignore"}
print('initial_model test loss =', federated_eval(initial_model, federated_test_data))
print('trained_model test loss =', federated_eval(model, federated_test_data))
```

```
initial_model test loss = 22.795593
trained_model test loss = 17.278767
```

上海市高可信计算重点实验室

Shanghai Key Laboratory of Trustworthy Computing

Thanks

董天文

East China Normal University

2019.3.28