机器学习导论期末实验报告

吕志存, 张含笑

2020 年 6 月 19 日

1 摘要

本文是针对” 华为云杯”2020 深圳开放数据应用创新大赛 · 深圳北站周边交通拥堵指数预测撰写的实验报告. 根据官方所给的深圳北站附近网约车的 GPS 和交通拥堵指数等信息, 我们针对每个不同的路口整理出训练集和测试集, 其中训练集中样本的特征包含 6 个相邻时间段相关信息, 对应的标签则是需要预测之后的的三个相邻时间段的拥堵指数, 测试集的样本具有相同特征. 我们主要采用集成学习方法——随机森林、XGBoost、LigntGBM——针对 12 个路口分别训练模型用于预测交通拥堵指数. 另外我们还将样本特征进行降维处理, 尝试获得更好的结果. 最后我们将测试集在四个不同模型下预测的结果加权融合, 获得了比单个模型预测更好的结果.

2 介绍

队名: nzhxjlzcu

成员: 人工智能学院 吕志存 181220038; 人工智能学院 张含笑 181220067

在交通拥堵指数预测的题目中, 需要根据过去的交通拥堵指数和某一时间段内的网约车 GPS 数据预测某一时刻的交通拥堵指数 TTI. 经预处理和特征工程之后, 我们将问题转化为一个回归问题. 回归问题是机器学习中一个基本问题, 受到了很多研究人员的关注, 该问的解决将有助于提升机器学习技术在多个应用场景中的表现, 比如在城市管理中对交通数据进行预测、推荐系统中对客户进行个性化推荐.

目前, 回归任务常用集成学习算法, 比如随机森林、XGBoost、lightGBM 这三种.

其中, 随机森林是一种传统方法, 提出时间较早. Leo Breiman 于 2001 年在 Random Forests 中给出了详细介绍 [1], 该算法的独特之在于对训练集进行随机自助采样来训练每一棵决策树和训练决策树时引入随机属性划分. 引入这两点的随机性之后, 随机森林不仅在准确性上和 AdaBoost 不相上下, 而且对噪音更加鲁棒.

XGBoost 由来自华盛顿大学的陈天奇于 2016 年在 XGBoost: A Scalable Tree Boosting System 中提出 [2]. Boosting 算法的基本思想是将许多弱学习器串行集成在一起得到强分类器. XGBoost 是一种典型的 Boosting 算法, 将许多树模型集成在一起, 所用到的树模型则是 CART 回归树模型. 作者还从计算机系统方向做了大量优化, 使模型在训练中的速度大大提升, 同时也可以利用硬盘 IO 处理超大规模数据, 甚至优化了缓存. 对比传统的 GBDT, XGBoost 在目标函数上加入了惩罚项, 使模型的泛化能力大大增强, 且对行列支持降采样, 优化了计算速度.

LightGBM 是一种新兴的方法. 为了处理在训练集特征维数过高和数据规模过大时 GBDT 的效率和可扩展性问题, 微软团队在 2017 年提了 lightGBM 算法 [3]. 该算法的独特之处在于提出了两种新技术 GOSS 和 EFB: 其中 GOSS 保留梯度大的样本, 随机去掉梯度小的样本 EFB 将互斥特

征进行 Bundling, 将该问题归约至图着色问题, 并使用贪心算法解决. 和传统的 GBDT 算法比较, lightGBM 在保持几乎同的准确率的前提下, 将速度提高了约 20 倍.

针对深圳北站交通拥堵指数预测的具体问题, 我们分别使用这三种算法实验, 但均存在超过预期的泛化误差. 我们针对已有方法存在的问题, 将这三个模型加权结合, 找到了使得预测的 TTI 更加准确的方法.

本报告结构安排如下: 第 3-5 部分详细介绍了我们在本次比赛中使用的模型和算法, 第 6-8 部分对算法进行分析, 第 9-10 部分对实验中数值结果和实验过程进行展示分析以及讨论, 最后在第 11 部分进行实验总结, 初赛排名在结论部分.

3 模型

观察数据, 得到每个文件中数据的基本内容:

- 201901_201903.csv/201910_11.csv/20191201_20191220.csv/toPredict_train_gps.csv: 网约车订单追踪数据, 每一条记录是一个网约车订单, 其中包含了订单编号 id_order、用户编号 id_user 和数量不定的五元组 gps_records. (经度, 纬度, 速度, 方向, 时间戳)
- train_TTI.csv: 记录了每个路口从 2019/1/1 0:00 - 2019/12/21 23:50 的交通信息, 每十分钟一条记录, 包括 (路段代号 id_road, 交通拥堵指数 TTI, 路过车辆平均速度 speed, 标准时间 time)
- toPredict_train_TTI.csv: 记录了每个路口从 2019/12/21 7:30 - 2020/1/1 20:50 的交通信息, 具体有记录的时间段为 (奇数时:30 - 偶数时:20), 例如 (7:30 - 8:20, 9:30 - 10:20), 每十分钟一条记录, 包括 (路段代号 id_road, 交通拥堵指数 TTI, 路过车辆平均速度 speed, 标准时间 time).
- toPredict_noLabel.csv: 是需要预测交通拥堵指数的样本. (样本号 id_sample, 路段代号 id_road, 标准时间 time). 时间段一般为某小时中的前半小时或后半小时, 与 toPredict_train_TTI.csv 中的时间段几乎没有重合.

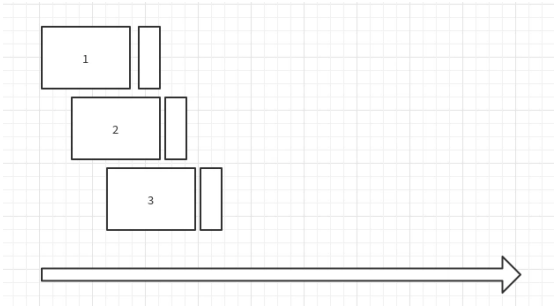

图 1: 将所有数据按照时间窗口切分

我们需要先对 201901_201903.csv/201910_11.csv/20191201_20191220.csv 中的数据进行预处理, 得到 processed_train_data, 对 toPredict_train_gps.csv 预处理得到 processed_test_data. 利用 processed_train_data 和 train_TTI.csv 中的数据可以训练模型, 由于需要用前 1 小时的数据对后 10-30 分钟进行预测, 因此我们选择将所有数据按照时间窗口切分, 分为训练集和验证集, 并训练时间序列模型.

在数据预处理的部分, 基本想法是遍历训练集, 根据网约车 gps 的信息、运动方向和各路口的位置统计出各时间段经过各路口的车辆数. 由于训练集中已有路过车辆的平均速度, 因此暂时不考

虑统计网约车的速度情况. 由于路段具有一定长度, 并且不是一条直的线段, 所以我们用平行四边形近似表示路段区域, 将网约车 GPS 看作一个点的坐标, 用点是否在区域中判断车辆是否经过这一路段. 再通过比较网约车的方向和路段整体走向, 得出网约车具体经过哪一个方向的路段. 网约车 gps 数据清理的代码在 /data_processing 中经过数据预处理, 得出每个路口在各个时刻的车流量、TTI 和通过路口的平均车速, 存入 train_#name.csv 中.

在特征工程的部分, 基本思路是使用前 60 分钟的数据预测后 30 分钟的数据, 即用前 60 分钟的数据构成一个 24 维的向量, 预测出一个三维向量. 这里的 TimeSlice 是当天的第 i 个 10 分钟. 获取输入和输出向量的方法如下:

$$\begin{aligned} x_i \leftarrow [&Speed1, Car1, TTI1, TimeSlice1, Speed2, Car2, TTI2, TimeSlice2, \\ &Speed3, Car3, TTI3, TimeSlice3, Speed4, Car4, TTI4, TimeSlice4, \\ &Speed5, Car5, TTI5, TimeSlice5, Speed6, Car6, TTI6, TimeSlice6] \\ y_i \leftarrow [&TTI7, TTI8, TTI9],\ x_i \in \mathbb{R}^{24},\ y_i \in \mathbb{R}^3 \end{aligned}$$

我们希望训练出一个 h:

$$h : \mathbb{R}^{24} \to \mathbb{R}^3$$

基于 mae 误差, 求解以下优化问题:

$$\min_h \frac{1}{|D|} \left[\sum_{x_i \in D_x} \sum_{j=1}^{3} |y_{ij} - h(x_i)_j| \right]$$

4 基本理论和方法

集成学习通过构建并结合多个学习器来完成学习任务. 根据集成学习 “好而不同” 的原理, 使用随机森林、XGBoost、LightGBM 作为个体学习器, 将他们进行结合, 以获得比单一学习器更好的泛化能力. 集成学习主要有两种方法: boosting 和 bagging. boosting 算法的主要思想是首先根据初始训练集训练出一个基学习器, 然后根据基学习器的表现调整样本权重, 使得基学习器预测较差的样本在后续受到更多关注, 重复这个过程, 直到基学习器数量到达预先设定的 T 为止. bagging 算法的主要思想是采用自主采样的方法从训练集中采样得到 T 个样本集合, 使用每个采样集训练出一个学习器, 然后把这 T 个学习器的结果相结合.

随机森林是一种 bagging 的方法, 它和传统 bagging 的区别在于它又引入了一种随机性——属性选择的随机性. 随机森林算法对于决策树的每个节点, 先从该节点的属性集合种随机选择包含 k 个属性的子集, 然后从中选出最优属性用于划分.

XGBoost 是一种 boosting 方法, 在梯度增强框架下实现机器学习算法. XGBoost 提供了一种并行的 boosting 算法 (也称为 GBDT、GBM), 可以快速、准确地解决许多数据科学问题.

LightGBM 基于 XGBoost 算法, 再引入一种随机性: 保留梯度大于阈值的向量, 但是随机保留梯度小于阈值的向量. Ke [3] 通过理论分析和实验证明这种随机性的引入可以使得结果优于随机采样.

5 方法

在数据预处理、特征工程 (细节见 Sec. 3 模型) 之后, 为了便于后期处理, 我们对每个路口进行了训练集和测试集的划分, 使得该问题转变为了一个典型的回归问题. 我们使用了开源机器学习包 sklearn、lightGBM、XGBoost、pytorch 来完成回归任务.

首先, 我们使用 random forest 这一传统的集成学习方法. 在 gen_train 和 gen_test 函数中, 将输入的 24 维向量堆叠, 对每一个路口构成 2872×24 维矩阵, 将输出的 3 维向量堆叠, 构成 2872×3 矩阵, 同理构成 84×24 维测试输入矩阵. 使用 train_test_split 划分训练集和验证集, 用 718×24 的矩阵作为验证集输入, 2154×24 作为训练集输入. 随机森林参数使用默认参数, 调用 RandomForestRegressor 的 fit 函数, 进行训练. 训练结束之后, 在验证集上计算 l1 误差. 对于每一个路口调用 RandomForestRegressor 的 predict 函数进行预测.

其次, 我们使用 XGBoost 这种典型的 boosting 方法. 依然是利用 gen_train 和 gen_test 函数整理出输入向量, 用 train_test_split 划分训练集和验证集. 输入向量 X 维度为 2872, y 维度为 2872×3, 随机划分比例为 3:1. 训练模型时, 树的最大深度为 6, 学习率为 0.1, 最小子权重为 5, 生成的最大树的数目为 150, 对于每棵树随机采样的比例为 0.8, 防止过拟合, 设置早停条件为验证集上 10 轮没有改善即停. 调用 XBGRegressor 中的 fit 函数进行训练, 并用 predict 函数进行预测.

同时我们也使用了 PCA 降维之后的 20 维向量作为输入, 训练 xgboost 模型.

然后, 我们使用 lightGBM 算法. 和前文相同, 进行训练集、验证集、测试集划分. 但是这里有一点值得注意：lightGBM 只支持预测实数值. 所以我们训练出 3 个模型分别用来预测后面的三个时段, 最后将这三个结果拼接成一个向量. 在参数 params 设定中, 经过多轮实验和参数调优, 最终我们设置参数中目标函数为 l1 regression, 评估函数为 l1 for regression, 学习率为 0.1, 设置提升类型为 gbdt, 最大迭代轮数为 50 轮, 早停条件为验证集上 5 轮没有改善. 设置了早停的条件可以有效避免过拟合的问题. 调用 lightgbm 中的 train 函数进行训练, 最后使用 lightgbm 中的 predict 进行预测.

最后, 我们通过将四种模型的预测结果加权融合, 得到最终预测结果.

6 方法分析

我们比较了不同算法性能、收敛速度、稳定性、适用性四方面. 对于性能, 我们用训练耗费的时间 (单位秒) 度量；对于收敛速度, 我们用训练耗费的迭代轮数来度量; 对于稳定性, 我们用 5 次不同训练验证集划分下的 mae 的方差来度量；对于适用性, 在 TTI 预测问题中, 这两种模型都适用, 所以我们设其为 1. 绘制出雷达图如图 2:

可以看出 lightgbm 的训练速度快于 XGBoost, 这和 Ke [3] 论文中提出来的论证结果是一致的, 也体现了 lightgbm 在面对大量训练数据、高维特征的优越性, 尽管这个问题的数据量比较大, 但是 lightgbm 通过 GOSS 的加入, 大大提高了运行速度.

在迭代轮数方面, 两者相差很少.

在稳定性方面, 实验次数不是很多, 初步体现出 xgboost 优于 lightGBM, 但这一点还有待更多实验.

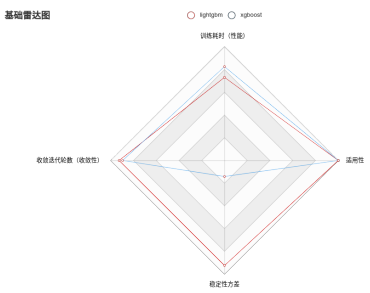

图 2: 不同算法性能、收敛速度、稳定性、适用性

7 算法

随机森林:

Algorithm 1 Random Forest

Input: Training set S , Features F and the number of trees in forest B
function RandomForest(S,F)
$H \leftarrow \emptyset$
for $i \in 1, ..., B$ **do**
  $S^{(i)} \leftarrow A\ bootstrap\ sample\ from\ S$
  $h_i \leftarrow RandomizedTreeLearn(S^{(i)}, F)$
  $H \leftarrow H \cup h_i$
end for
return H
function RandomizedTreeLearn(S,F)
for all node **do**
  $f \leftarrow very\ samll\ subset\ of\ F$
  split on best feature in f
end for
return learned tree

XGBoost:

Algorithm 2 Exact Greedy Algorithm for Split Finding

Input: I : instance set of current node
Input: d : feature dimension
gain $\leftarrow 0$, $G \leftarrow \sum_{i \in I} g_i$, $H \leftarrow \sum_{i \in I} h_i$
for $k \in 1, ..., m$ **do**
  $G_L \leftarrow 0, H_L \leftarrow 0$
  for $j \in$sorted(I, by x_{jk}) **do**
    $G_L \leftarrow G_L + g_j, H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L, H_R \leftarrow H - H_L$
    score $\leftarrow max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end for
end for

近似搜索算法: 当数据太多不能装载到内存时, 不能进行精确搜索分裂, 只能近似. 根据特征分布的百分位数, 提出特征的一些候选分裂点, 将连续特征值映射到桶里 (候选点对应的分裂), 然后根据桶里样本的统计量, 从这些候选中选择最佳分裂点.

Algorithm 3 Approximate Algorithm for Split Finding

for $k \in 1, ..., m$ **do**

Propose $S_k = \{s_{k1}, s_{k2}, ..., s_{kl}\}$ by precentiles on feature k.

Proposal can be done per tree (global), or per split (local).

end for

for $k \in 1, ..., m$ **do**

$G_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq x_{jk} \geq s_{k,v-1}\}} g_j$

$H_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq x_{jk} \geq s_{k,v-1}\}} h_j$

end for

Follow same step as in previous sectio to find max

score only among proposed splits.

LightGBM：

Algorithm 4 Gradient-based One-Side Sampling

Input: I : training data, d : iterations

Input: a : sampling ratio of large gradient data

Input: b : sampling ratio of small gradient data

Input: $loss$: loss function, L : weak learner

models ← {}, fact ← $\frac{1-a}{b}$

topN ← a × len(I), randN ← b × len(I)

for $i \in 1, ..., d$ **do**

preds ← models.predict(I)

g ← $loss$(I,preds), w ← 1,1,...

sorted ← GetSortedIndices(abs(g))

topSet ← sorted[1:topN]

randSet ← RandomPick(sorted[topN:len(I)],randN)

usedSet ← topSet + randSet

w[randSet] × = fact ▷ Assign weight $fact$ to the small gradient data

newModel ← L(I[usedSet],-g[usedSet],w[usedSet])

models.append(newModel)

end for

Algorithm 5 Merge Exclusive Features

Input: $numData$: number of data
Input: F : One bundle of exclusive features
Output: newBin, binRanges
binRanges ← {0}, totalBin ← 0
for $f \in 1, ..., F$ **do**
  totalBin += f.numBins
  binRanges.append(totalBin)
end for
newBin ← new Bin(numData)
for $i \in 1, ..., numData$ **do**
  newBin[i] ← 0
  for $j \in 1, ..., len(F)$ **do**
    if $F[j].bin[i] \neq 0$ **then**
      newin[i] ← F[j].bin[i] + binRanges[j]
    end if
  end for
end for

模型融合:

Algorithm 6 Final

Input: x∈ $\mathbb{R}^{m\times 24}$, y∈ $\mathbb{R}^{m\times 3}$, x_test∈ $\mathbb{R}^{n\times 24}$
Output: y_test∈ $\mathbb{R}^{n\times 3}$
x_PCA = PCA(x), x_PCA_test = PCA(x_test)
x_PCA_train, x_PCA_valid, y_PCA_train, y_PCA_valid = split(x_PCA, y)
x_train, x_valid, y_train, y_valid = split(x, y)
validset = (x_valid, y_valid)
validPCAset = (x_PCA_valid, y_PCA_valid)
model1 = xgboost.train(x_train, y_train, validset)
model2 = lightgbm.train(x_train, y_train, validset)
model3 = random_forest.train(x_train, y_train, validPCAset)
model4 = xgboost.train(x_PCA_train, y_PCA_train)
pred1 = model1.predict(x_test)
pred2 = model2.predict(x_test)
pred3 = model3.predict(x_test)
pred4 = model4.predict(x_PCA_test)
return 0.405 * pred1 + 0.45 * pred2 + 0.1 * pred3 + 0.045 * pred4

8 算法分析

由于我们的具体问题是一个工程性的回归预测问题, 所以算法分析部分不是本次报告的重点. 下面我们简要的论述一下各个算法的时间复杂度分析结果.

1. 首先是 random forest 的算法分析. 根据 Louppe [4] 的分析, 随机森林的时间复杂度可以概括为 $\Theta(MK\tilde{N}\log^2\tilde{N})$(best case), $O(MK\tilde{N}^2\log\tilde{N})$(worst case), $\Theta(MK\tilde{N}\log^2\tilde{N})$(average case), 这里 M 是基学习器数量, N 表示样本数, K 表示每个节点随机选择的属性数量, $\bar{N}$ 表示 0.632*N.

2. 其次是 XGBoost, 其目标函数定义为:

$$\mathcal{L} = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

目标函数由两部分组成, $\sum_{i=1}^{n} l(y_i, \hat{y}_i)$ 用于衡量预测值与真实值的差距, $\sum_{k=1}^{K} \Omega(f_k)$ 则是正则化项. 正则化项同样包含两部分, $\Omega(f) = \gamma T + \frac{1}{2}\lambda\|\omega\|^2$. T 表示叶子结点的个数, ω 表示叶子节点的分数, γ 可以控制叶子结点的个数, λ 可以制叶子节点的分数不会过大, 防止过拟合.

生成 t 棵树后, 预测分数可以写成 $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$. 代入目标函数后在 $f_t = 0$ 处泰勒展开, 去掉前 t 棵树的预测分数与 的残差 (对目标函数不影响), 简化目标函数为:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

其中 $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$, $h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$, 为一阶和二阶梯度.

上式是将每个样本的损失函数值加起来, 而每个样本都最终会落到一个叶子结点中, 所以我们可以将所以同一个叶子结点的样本重组起来, 得到:

$$\mathcal{L}^{(t)} = \sum_{j=1}^{T} \left[\left(\sum_{i \in I_j} g_i \right) \omega_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) \omega_j^2 \right] + \gamma T$$

通过上式的改写, 可以将目标函数改写成关于叶子结点分数 ω 的一个一元二次函数, 求解最优的 和目标函数值就变得很简单了.

3. 最后是 lightGBM. 这种算法是在 XGBoost 的基础上进行改进, 加入了 GOSS 和 EFB 算法, 从而大大提高了速度. 所以我们这里简要分析一下为什么加入它们之后可以大大提高速度. 首先是 GOSS, GOSS 通过随机去除一些梯度小于阈值的样本, 起到了减少训练样本的作用, 可以一定程度地提高训练速度. 同时, Ke [3] 在论文中通过定理 3.2 的分析

$$\epsilon(d) \leq C_{a,b}^2 \ln\frac{1}{\delta} \cdot \max\{\frac{1}{n_l^j(d)}, \frac{1}{n_r^j(d)}\} + 2DC_{a,b}\sqrt{\frac{\ln\frac{1}{\delta}}{n}}$$

得到误差项取决于

$$\sqrt{\frac{1}{n}}$$

所以, 当 $n \to \infty$, 该误差

$$\epsilon \to 0$$

即两者误差很小. 所以 GOSS 方法达到了在几乎不损失精度的情况下, 加快了训练速度的效果.

如果两个特征是互斥的, 那么可以将两个特征合为一个特征. EFB 就利用了这个原理. 加入 EFB 将原训练集中稀疏而且互斥的特征变成一个特征, 减少了特征数量,

$$O(data \times feature)$$

转变为

$$O(data \times bundle)$$

可以加快训练速度.

各算法时间复杂度对比:

算法	最好情况	最坏情况	平均情况
CART	$\Theta(pN\log^2 N)$	$O(pN^2\log N)$	$\Theta(pN\log^2 N)$
Bagging	$\Theta(Mp\tilde{N}\log^2\tilde{N})$	$O(Mp\tilde{N}^2\log\tilde{N})$	$\Theta(Mp\tilde{N}\log^2\tilde{N})$
随机森林	$\Theta(MK\tilde{N}\log^2\tilde{N})$	$O(MK\tilde{N}^2\log\tilde{N})$	$\Theta(MK\tilde{N}\log^2\tilde{N})$
ETs	$\Theta(MKN\log N)$	$\Theta(MKN^2)$	$\Theta(MKN\log N)$
PERT	$\Theta(MN\log N)$	$\Theta(MN^2)$	$\Theta(MN\log N)$

9 数值结果

实验环境: windows + Intel Core i5 + Python 3.7.6; Ubuntu 18.04.4 LTS + Intel Core i5 + Python 3.6.9

1. 观察数据: 利用部分网约车的 GPS 信息, 绘制其行车路线图:

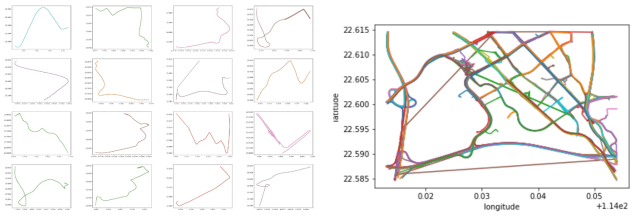

图 3: 部分网约车行车路线图

2. 数据预处理: 利用部分网约车的 GPS 和速度信息, 在地图上绘制拥堵状况热力图, 其中速度越小对应的数值越大, 即色温越高的地方越拥堵:

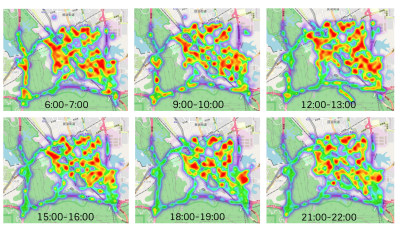

图 4: 拥堵状况热力图

3. 绘制了三个模型 (lightgbm, xgboost, random_forest) 在各个路口验证集上的 mae 误差如下图:

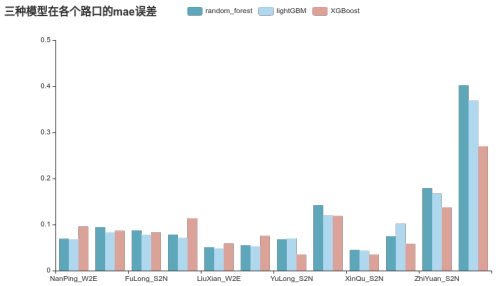

图 5: 模型在各个路口验证集上的 mae 误差

10 讨论

在讨论部分中, 我们主要研究了:

1. 用 XGBoost 在验证集上进行测试, 发现用六个时间段预测之后的三个时间段, 尽管分别训练了三个模型, 但是效果仍是越来越差, 这是因为第三个时间段相较前两个与输入的六个时间段的相关性变小了, 效果自然不如之前的:

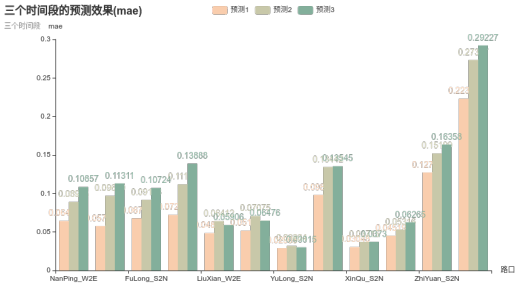

图 6: 验证集上三个时间段的预测效果

2. 在 TTI 回归问题中降维、聚类、集成有效性的问题. 首先是 KMeans 聚类. 最初我们根据交通拥堵指数问题实际情况, 考虑 TTI 可能会在不同时间段有不同分布模型, 所以我们对输入的向量先进行了聚类, 然后在每一类中进行回归预测. 但是, 效果并不理想, 最终我们放弃了先进行聚类的想法. 实验结果见下表格. 我们希望找到该问题不适合采用 Kmeans 聚类的原因.

簇数	验证集误差
1	0.118500
2	0.147335
3	0.121700
5	0.131066

然后我们考虑使用降维进行数据可视化, 我们对回归问题中每一个路口输入的 24 维向量进行 PCA 降维, 降到 2 维, 然后以此作为 x,y 轴, 绘制出图像如图 7.

容易看出, 在每个路口的图像中, 图形都近似团成一簇, 只有少量点散落在簇外, 直观上并没有多簇的情况. 再结合 kmeans 的原理, 我们认为 kmeans 会把原先的一团分成几部分, 反而破坏了模型的泛化能力, 所以导致加入聚类效果不佳. 同时由于降维之后我们发现有很少的点散落

在簇外, 我们考虑了异常点的情况. 由于 PCA 降维有利有弊, 既可以通过取较大的特征值对应的特征向量来去除噪音, 也会丢失一部分特征信息, 所以我们用降维前后的数据分别训练 2 种 xgboost 模型, 然后根据问题实际特点将它们的预测结果加权相加, 我们希望通过这种方法能够尽可能保留有效特征又去除一些噪音影响.

最后, 我们根据各个模型在各路口预测结果的 mae 误差图 (图 5), 可以看出这三个模型在不同路口总体表现相似, 但是在不同路口三种模型各有优劣. 所以, 我们希望按照不同权重将它们结合起来, 以得到更高的泛化能力. 经过实验, 最终确定权重见伪代码. 通过这一步, 可以将泛化误差从 0.1144 降低到 0.1070.

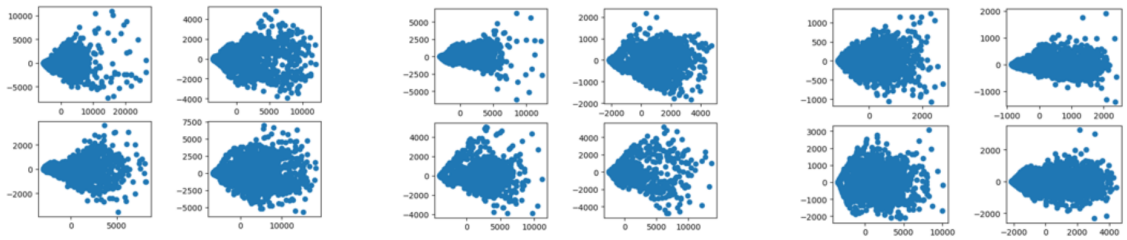

图 7: PCA

11 结论

初赛排名:

22	nzhxjlzcu	0.107	2020/06/18

经过实验, 我们得到以下结论:

1. 尽管针对每个路口的每个样本, 针对三个时间段训练了不同的模型, 但预测效果是 model1 > model2 > model3.
2. 集成学习的效果要好于单一模型, 这体现在实验过程中, 吕志存曾尝试用 lstm 进行预测, 但效果比随机森林和 XGBoost 差很多.
3. 将多个模型预测的结果取加权和会比随机森林 (RF)、XBGoost、LightGBM、XGBoost+PCA 效果更好. 最终确定的权值为: 0.405*xgb+0.45*(xgb+PCA)+0.1*RF+0.045*lightgbm. 在测试集上得到了最好的结果 0.1070.
4. 此外我们还有一些想法, 因为时间限制暂时没能着手实践:
   - 我们整理出的每个样本可以表示为 (f1, f2, f3, f4, f5, f6)->(t7, t8, t9), 其中 1 9 代表了时间顺序, 在本实验中, 我们根据前 6 个特征, 直接用三个模型分别对后三个 TTI 进行预测, 我们更希望能用第一个模型的预测结果加入到第二个模型的输入中去, 将第二个模型的预测结果加入到第三个模型的输入中去. 这样也许能得到更好的结果.
   - 我们发现致远路的预测效果很差, 我们希望针对致远路单独进行调试, 也许能得到更好的结果.

- 我们发现被融合的 4 个模型在 12 个路口的表现各不相同, 我们希望能针对 12 个路口分别调整 4 个模型的权值, 也许能得到更好的结果.

12 致谢

感谢助教老师提供了针对数据竞赛的阅读材料和撰写论文的建议.
感谢为我们提供了优秀的开源模型的团队们.

参考文献

[1] Leo Breiman. Random forests. *Machine Learning, 45(1):5-32*, 2001.

[2] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 785–794, 2016.

[3] Thomas Finley Guolin Ke, Qi Meng. Lightgbm: A highly efficient gradient boosting decision tree. *In Advances in Neural Information Processing Systems*, page 3149–3157, 2017.

[4] Gilles Louppe. Understanding random forest from theory to practice. *Machine Learning*, 2015.

13 组员贡献

- 181220067 张含笑: GPS 轨迹绘制和观察, 部分数据清洗, 数据集划分,XGBoost 回归, 报告部分内容撰写
- 181220038 吕志存: 部分数据清洗,RF 回归,LightGBM 回归, 集成学习参数调整, 报告部分内容撰写

14 附录

我们使用 git 来管理代码,github 项目地址: `https://github.com/Michelia-zhx/Huaweicloud_Competition_Traffic`

```
/
└── algorithm ..... 不同算法的应用于本次 TTI 预测问题的代码
    ├── train_array ..... 读入训练集的中间结果
    ├── gen_train_npy.py ..... 产生中间结果, 暂时存到 train_array 中
    ├── lightgbm.py ..... 使用 lightGBM 处理回归问题
    │   ├── train(train_df, test_df, params) ..... 使用 lightgbm 进行训练
    │   └── predict(road_id, timestamp, train_TTI, test_gps, models) ..... 使用
    │       训练出的 lightgbm 模型进行预测
    ├── lstm.py ..... 使用 LSTM 模型处理回归问题
    ├── random_forest2.py ..... 使用随机森林处理回归问题
    │   └── train(df,rf1) ..... 使用随机森林在训练集上训练
```

- predict(road_id, timestamp, train_TTI, test_gps, lst) . 利用 lst 中的模型对 road_id 在 timestamp 时的 TTI 进行预测
- with_cluster_xgboost_alg.py
  - gen_train() ..... 把数据经过特征工程变为 24 维的输入向量
  - gen_test() ..... 把测试数据经过特征工程变为 24 维的输入向量
  - train() ..... 训练模型
  - evaluate() ..... 使用 model 在 X 上进行评价
  - predict() ..... 使用 model 对测试集进行预测
- xgboost_alg.py ..... 使用 XGBoost 处理回归问题
  - train(train_X, train_y, eval_X, eval_y, road_index) . 对 r 训练集 oad_index 路口,train_X,train_y 作为, eval_X, eval_y 作为验证集, 进行训练
  - gen_test(model, pred_df) ..... 进行特征工程, 并对测试集产生预测结果
  - evaluate(model, X, y) ..... 评价 model 在 X 上的 mae 误差
- data_preprocessing ..... 数据预处理的代码
  - count_car.py ..... 通过对每 10 分钟经过各个路口的网约车计数, 得到以十分钟为粒度的车流量数据
  - gen_testset.py ..... 产生每个路口的测试集
  - gen_trainset.py ..... 产生每个路口的训练集
- datasets ..... 将训练数据和测试数据按照路口划分之后保存的 csv 结果
- photo ..... 根据 20191201-20191220.csv 中部分网约车的 GPS 信息, 绘制出的 GPS 图形
  - draw_road.py ..... 绘制出 20191201-20191220.csv 中一部分网约车 GPS 图形, 以经度作为横轴, 维度作为纵轴
- processed_test_data ..... 根据测试集上网约车 GPS 整理出的各个 10 分钟内每个路口的车流量
- processed_train_data ..... 根据训练集上网约车 GPS 整理出的各个时刻每个路口的车流量
- traffic ..... 部分训练集和测试集数据
- README.md ..... 项目背景和题目要求