毕业项目

Senga 优达学城 2020年3月11日

I. 问题的定义

项目概述

> 需要解决的问题涉及哪个领域？做这个项目的出发点？有哪些相关的数据集或输入数据？

本项目所解决的问题是**预测 Rossmann 未来的销售额**，该问题设计数据挖掘与金融领域。选择这个项目的出发点是因为对数据挖掘与金融感兴趣。使用了 kaggle 上所提供的 Rossmann 过去的销售记录作为输入数据。Rossmann在7个欧洲国家经营着3,000多家药店。目前，Rossmann商店经理的任务是预先提前六周预测他们的日常销售。商店销售受许多因素的影响。成千上万的个体经理根据他们独特的情况预测销售情况，结果的准确性可能会有很大差异。本次项目要求预测德国各地的1,115家商店6周中每周销售额。

> 问题的背景信息能够让完全没接触过这个问题的人充分了解这个问题吗？

问题的背景信息在 kaggle 上是比较具体的，对于输入数据的鱼问题的解释可以让完全没接触过这个问题的人充分了解这个问题

问题陈述

Rossmann Sales 是一个 regression 的 supervised learning，需要根据过去的销售数据提炼出 feature 然后进行预测。

预计使用 LGB 与 XGB 模型，每一步骤将会在分析部分具体展开：

- 首先要进行数据探索，探索 feature 的关系
- 数据预处理
- 对该模型进行调参优化
- 进行测试验证优化

评价指标

kaggle 给出的评价指标为 Root Mean Square Percentage Error (RMSPE)，

$$\mathrm{RMSPE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{y}_i}{y_i}\right)^2},$$

- 其中 y_i 一家 store 一天的销售额
- $\hat{y}_i$ 代表对应的 prediction
- 任何 0 sales 在评分中都会被忽略

用RMSPE作为评估销售预测的误差，在 RMSE 的基础上，令真实值与预测值之差再除以真实值，形成一个比例。

RMSPE 更贴近误差的概念。而相比于 MSE 和 RMSE， RMSPE 计算的是一个误差率，这样就避免了真实值之间大小的不同而对误差产生的影响。

RMSPE 的缺点是，如果预测值是真实值的两倍以上，则 RMSPE 取值可能就会非常大，失去了评估的意义。所以理论上， RMSPE的取值范围是 0 到正无穷

II. 分析

数据的探索

数据解释

name	meaning
Id	代表着 (Store, Date) 的 id
Store	一个商店的唯一 id
Sales	给定一天内的营业额
Customers	给定一天内的客户数量
Open	商店是否开门 0 = closed, 1 = open
StateHoliday	代表着国家假日，一般来说绝大多数商店都会在国家假日关门。注意学校在 public holiday 和周末都会关门。a = public holiday, b = Easter holiday, c = Christmas, 0 = None
SchoolHoliday	代表 (Store, Date) 是否有被学校放假所影响
StoreType	四种不同的商店类型: a, b, c, d
Assortment	商店的 assortment 等级: a = basic, b = extra, c =

	extended
CompetitionDistance	最近的竞争对手的距离，以米为单位
CompetitionOpenSince(Month/Year)	竞争对手开张的年/月
Promo	代表当日商店是否有打折
Promo2	持续性的打折 0 = store is not participating, 1 = store is participating
Promo2Since(Year/Week)	参与打折的年/周
PromoInterval	打折间隔 E.g. "Feb,May,Aug,Nov" means each round starts in February, May, August, November of any given year for that store

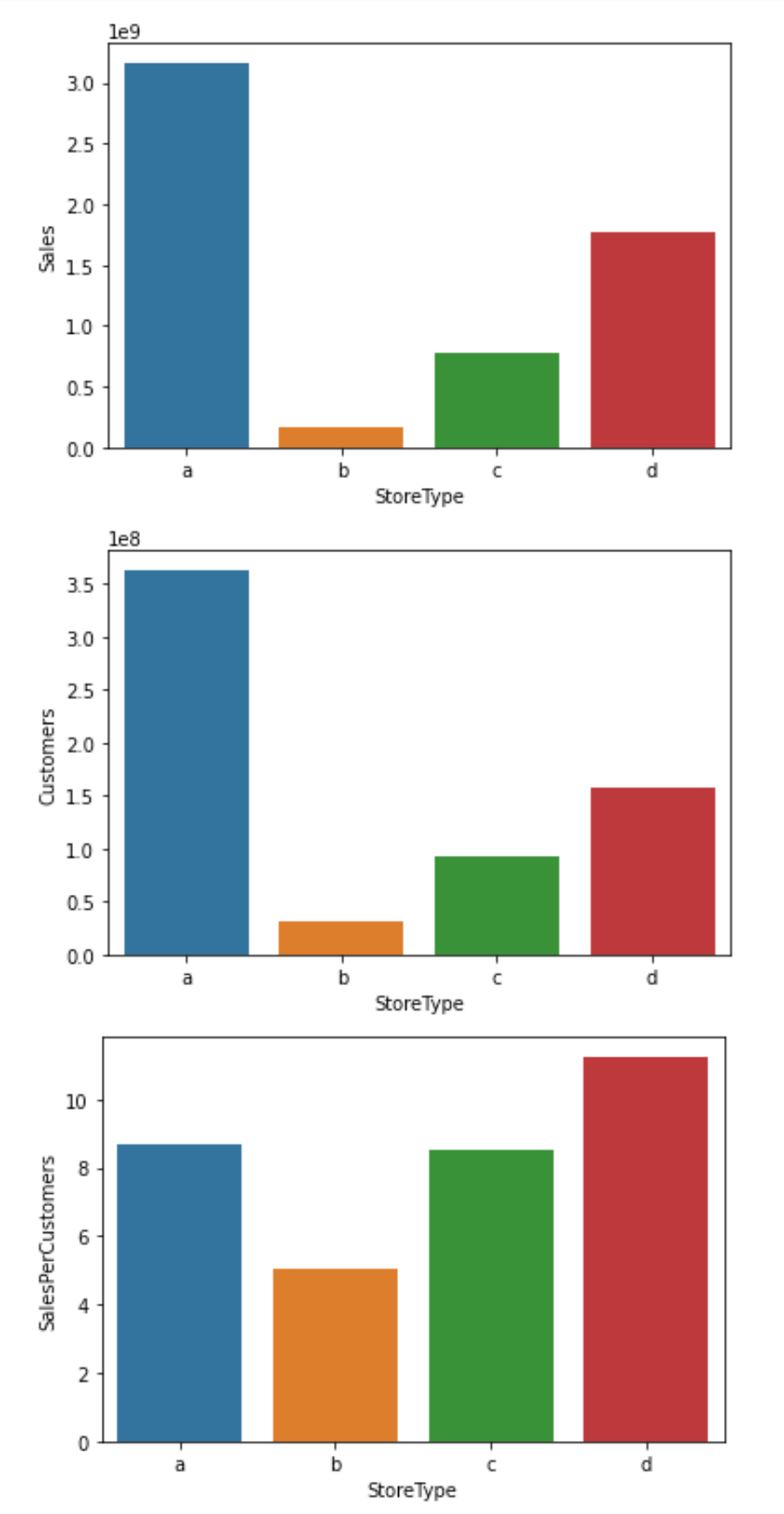
数据异常值

- 注意到在数据集中存在部分商店未开门却有营业额的情况，为了模型训练的准确性选择了商店开门且有营业额的数据作为训练集

探索性可视化

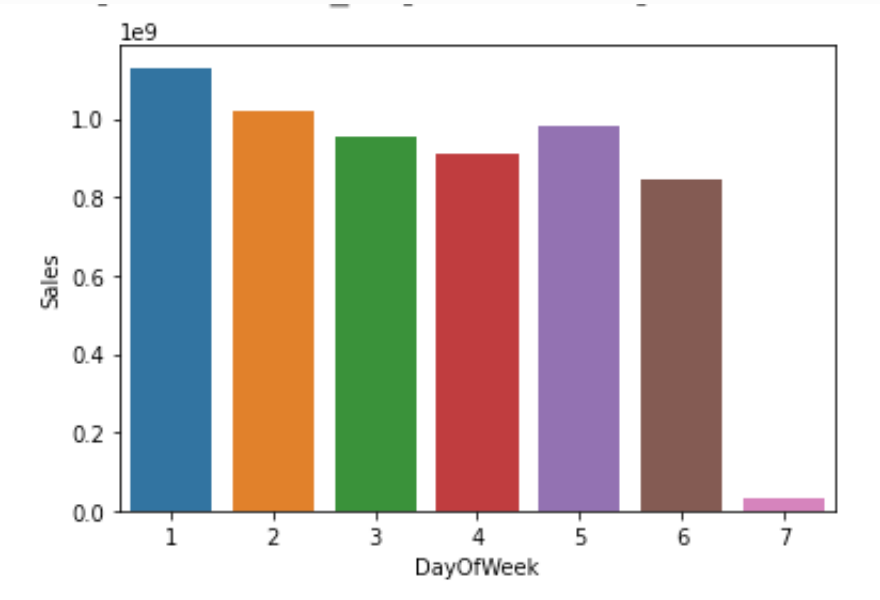
商店类型与营业额

增加了一个参数 `SalesPerCustomer`， 进行可视化后得出 a 的总营业额和客户数量最高，d的平均顾客画的钱最高，b类的商店在各个表现上均较差。

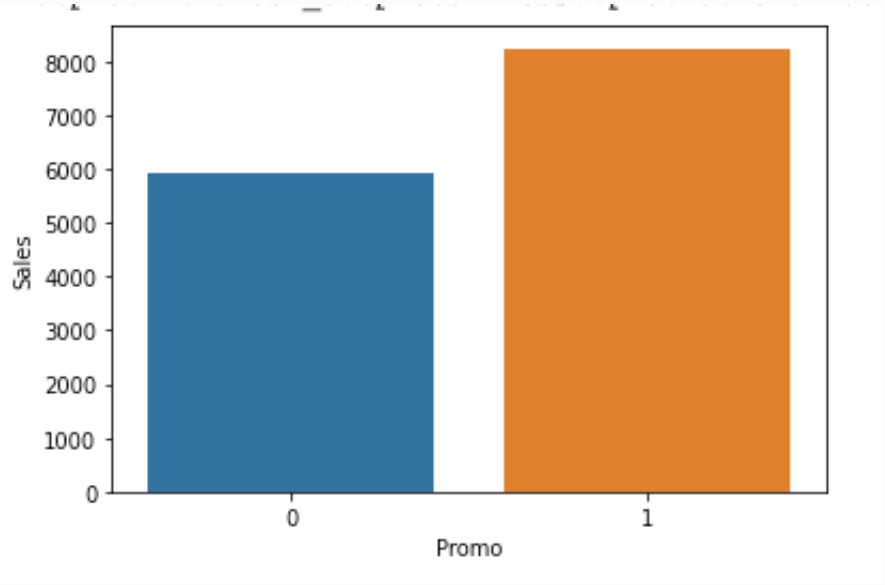


分析工作日与销售额

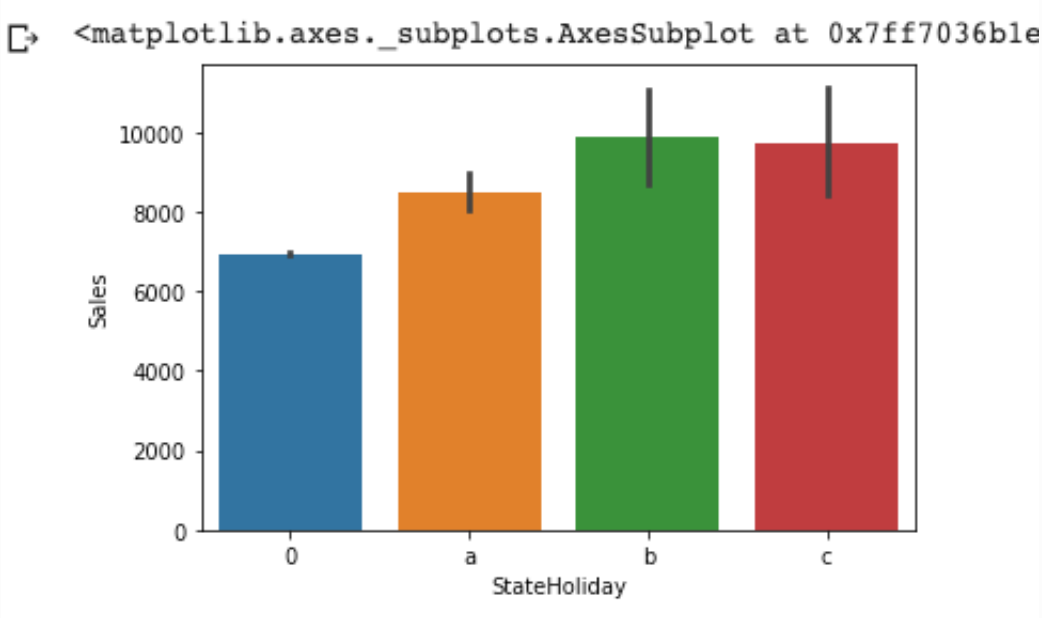
发现周日销售额最少，周一销售额最多。可知周几对于销售额来说是有参考价值的。


1e9
1.0
0.8
0.6
0.4
0.2
0.0
Sales
1
2
3
4
5
6
7
DayOfWeek

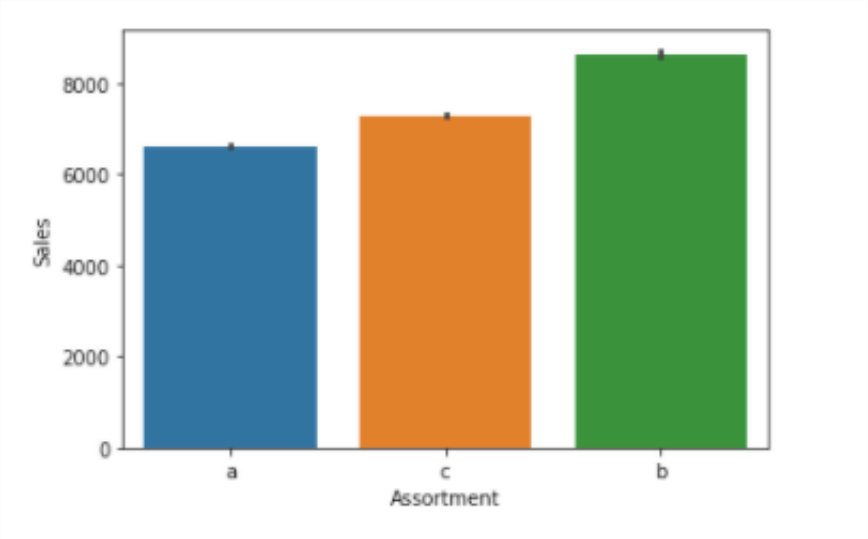
打折


8000
7000
6000
5000
4000
3000
2000
1000
0
Sales
0
1
Promo

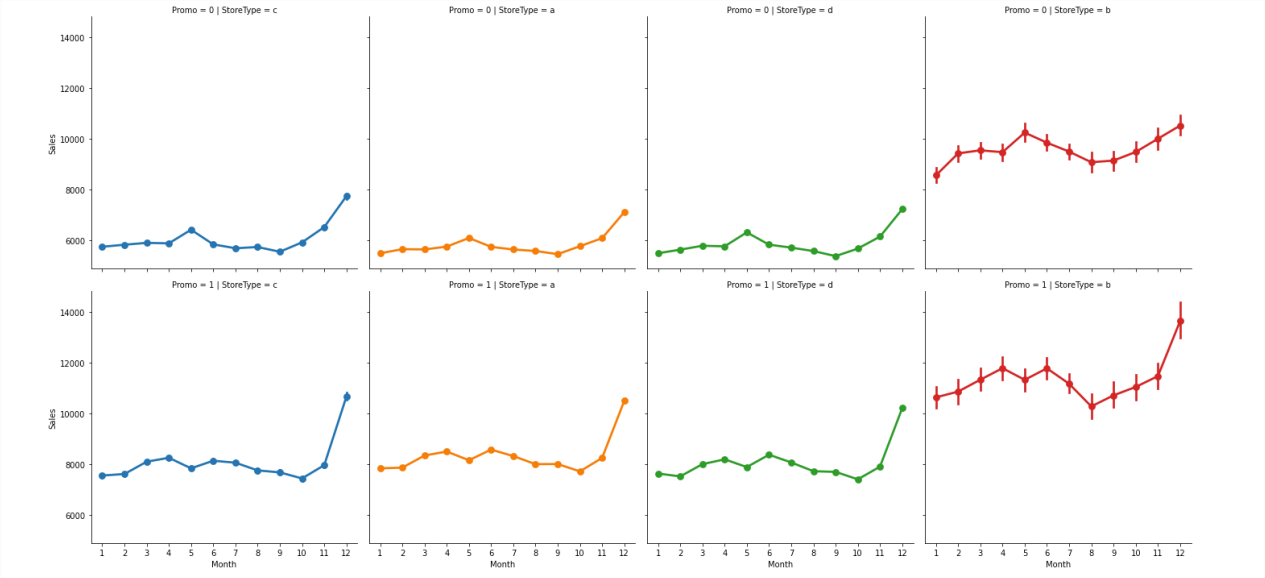
节假日

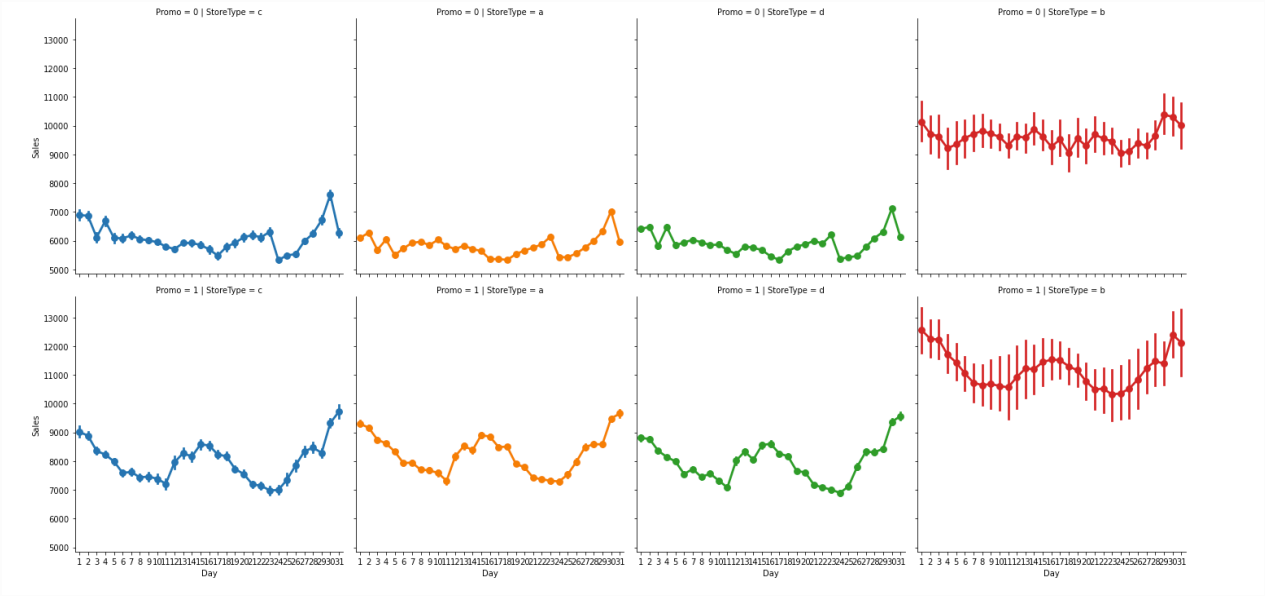


商店 AssortMent

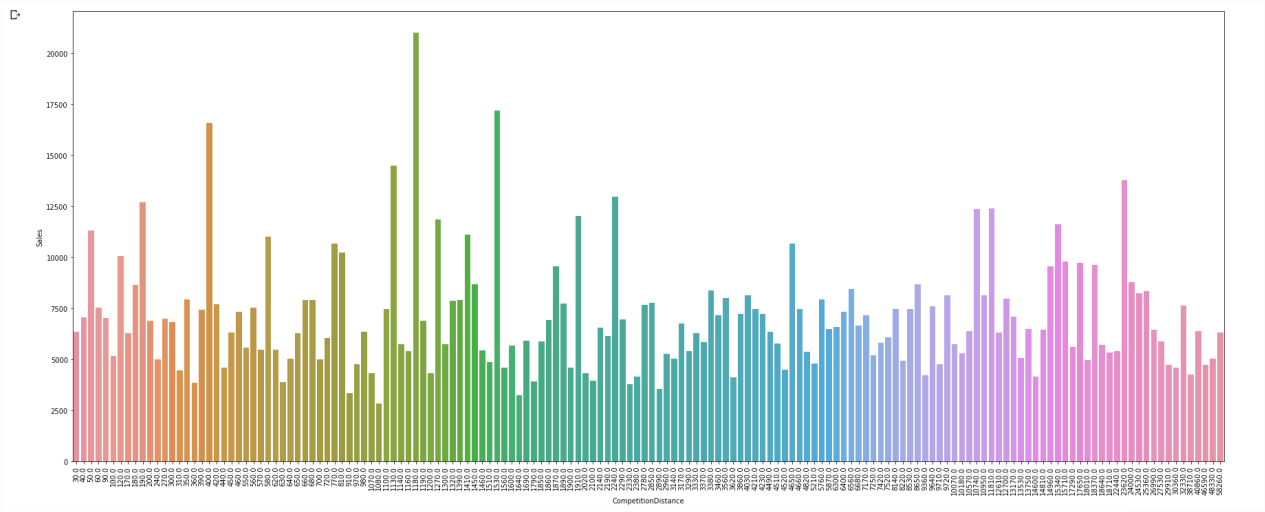


每月/日销量

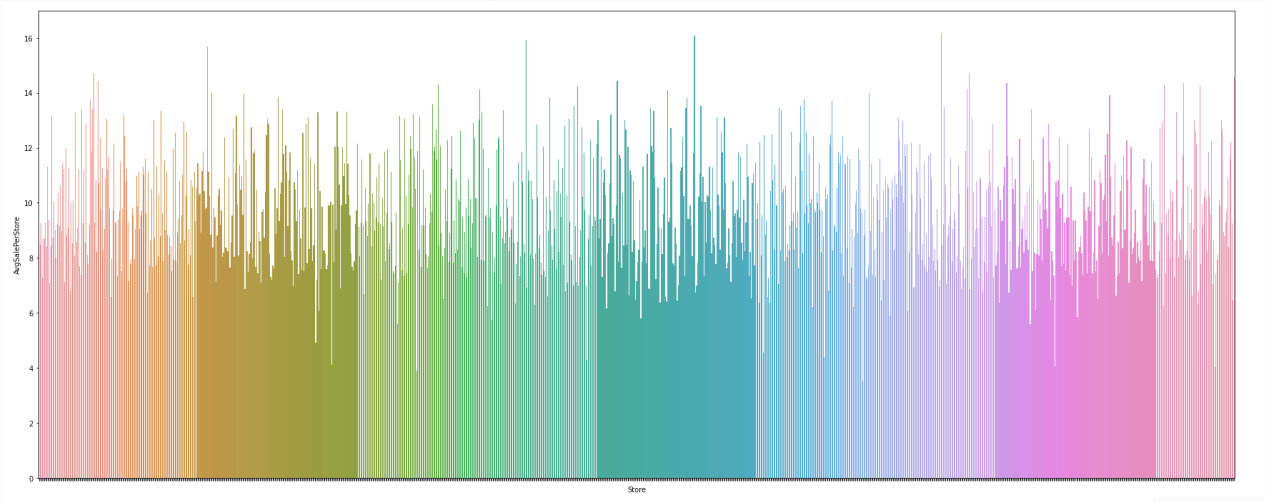




竞争对手距离与(平均)销售额的关系



商店销售额分析



算法和技术

实现模型的 training error 足够小，且算法的自由度不能过大。真实数据存在很多的噪声，需要在数据处理部分进行清除。

基准模型

模型的 trainning error 要足够小，需要处理实际数据中的噪声的现象。采用的是 GBDT(Gradient boosting) 模型：

Algorithm 1. The Algorithm Framework of GBDT

1. $F^*(x) = argminE_y(L(y, F(x))|x)$
2. $F_0(x) = f_0(x)$
3. $for \quad m = 1 \quad to \quad \mathrm{M}:$
4. $\quad \mathrm{g}_m(x) = -\frac{\partial E_y[L(y,F(x))|x]}{\partial F(x)}|F(x) = F_{m-1}(x) \quad //descent\ direction$
5. $\quad \rho_m = argminE_y[L(y, F_{m-1}(x) + \rho g_m(x)|x] \quad //step\ size$
6. $\quad f_m(x) = \rho_m g_m(x)$
7. $\quad F_m(x) = F_{m-1}(x) + \rho_m g_m(x)$
8. $end\ for$
9. $F^*(x) \approx F_M(x) = f_0(x) + \sum_{m=1}^{M} \rho_m g_m(x)$

XGBoost：

该算法思想是不断地添加树，不断地进行特征分裂来生长一棵树，每次添加一个树，其实是学习一个新函数，去拟合上次预测的残差。当我们训练完成得到 k 棵树，我们要预测一个样本的分数，其实就是根据这个样本的特征，在每棵树中会落到对应的一个叶子节点，每个叶子节点就对应一个分数，最后只需要将每棵树对应的分数加起来就是该样本的预测值。

- Add a new tree in each iteration
- Beginning of each iteration, calculate

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

- Use the statistics to greedily grow a tree $f_t(x)$

$$Obj = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$$

- Add $f_t(x)$ to the model $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$
  - Usually, instead we do $y^{(t)} = y^{(t-1)} + \epsilon f_t(x_i)$
  - ϵ is called step-size or shrinkage, usually set around 0.1
  - This means we do not do full optimization in each step and reserve chance for future rounds, it helps prevent overfitting

目标函数如下：

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

Training loss　　Complexity of the Trees

作为GBDT的高效实现，XGBoost是一个上限特别高的算法，因此在算法竞赛中比较受欢迎。简单来说，对比原算法GBDT，XGBoost主要从下面三个方面做了优化：

1. 算法本身的优化：

   在算法的弱学习器模型选择上，对比GBDT只支持决策树，还可以直接很多其他的弱学习器。

   - 在算法的损失函数上，除了本身的损失，还加上了正则化部分。
   - 在算法的优化方式上，GBDT的损失函数只对误差部分做负梯度（一阶泰勒）展开，而XGBoost损失函数对误差部分做二阶泰勒展开，更加准确。
2. 算法运行效率的优化：
   - 对每个弱学习器，比如决策树建立的过程做并行选择，找到合适的子树分裂特征和特征值。在并行选择之前，先对所有的特征的值进行排序分组，方便前面说的并行选择。
   - 对分组的特征，选择合适的分组大小，使用CPU缓存进行读取加速。将各个分组保存到多个硬盘以提高IO速度。
3. 算法健壮性的优化：
   - 对于缺失值的特征，通过枚举所有缺失值在当前节点是进入左子树还是右子树来决定缺失值的处理方式。
   - 算法本身加入了L1和L2正则化项，可以防止过拟合，泛化能力更强。

LightGBM

LightGBM 是一个梯度 boosting 框架，使用基于学习算法的决策树。它可以说是分布式的，高效的，有以下优势：更快的训练效率，低内存使用，更高的准确率，支持并行化学习，可处理大规模数据。

其他

另外还采用了 linear regression 和 decisiontree 作为基础模型进行比较，同时尝试了 LGB 算法，但是不知道出于什么原因效果不尽如人意。选择 XGboost 的原因也是因为在 google 的 colab 上可以直接使用 GPU 进行训练。

对于数据进行了商店每个星期日的销售额，商店类型对销售额的影响，连续月的销售额变化，节假日的销售额变化，对于每个商店种类的用户平均购买力进行分析。同时加入了以商店 id 为基准的信息分析。

基准目标

设置基准为 private leaderboard 的 top 10%，对于测试集rmpse为 0.11773

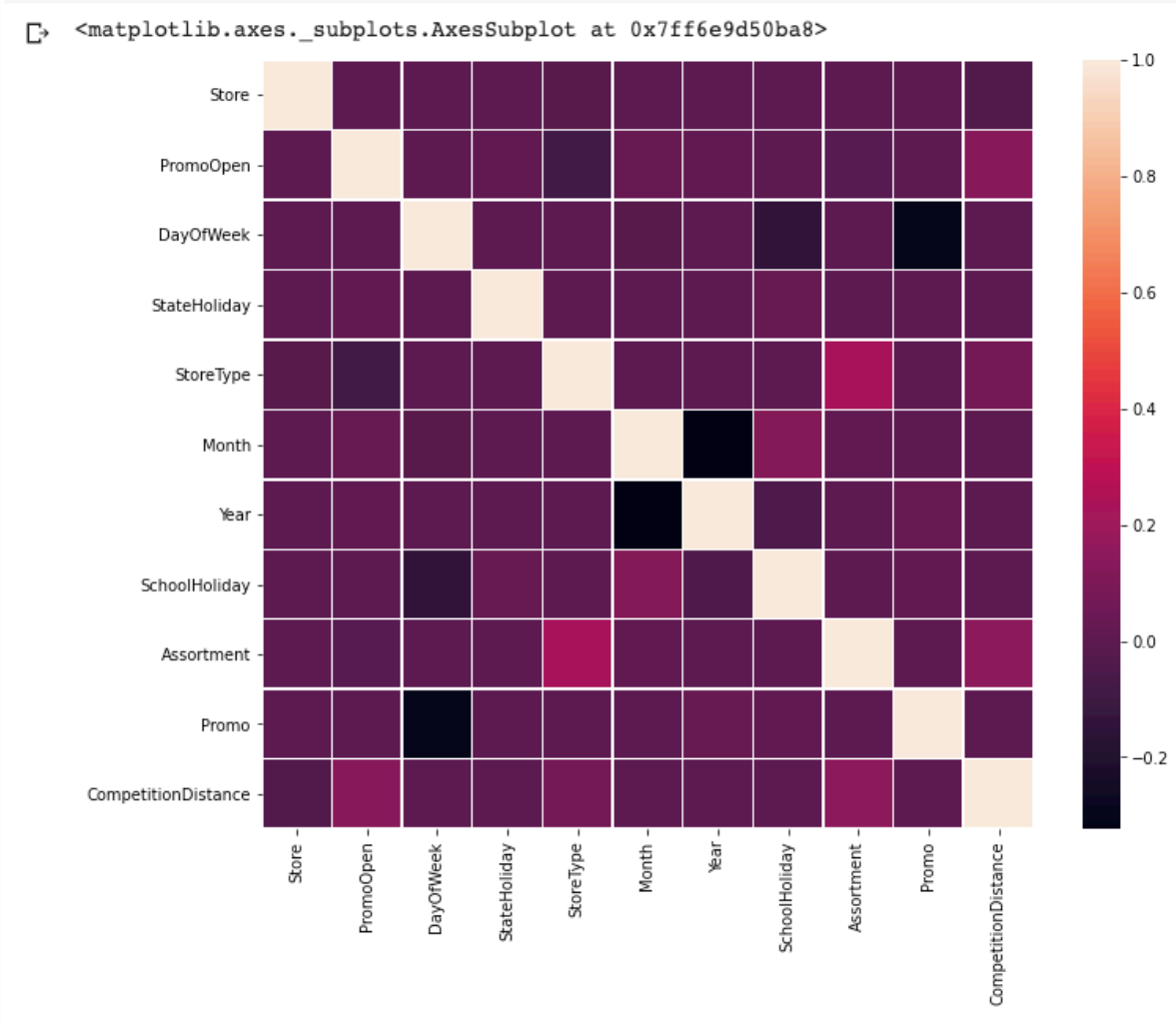
III. 方法

数据预处理

1. 填充缺失数据
    - test 中有部分 Open 没有值：填充默认为 1，关闭状态下的销售额预测没有意义。
    - store 数据中的 CompetitionDistance 部分缺失：使用中位数进行填充

	Store	StoreType	Assortment	CompetitionDistance	CompetitionOpenSinceMonth	CompetitionOpenSinceYear	P
290	291	d	a	NaN	NaN	NaN	
621	622	a	c	NaN	NaN	NaN	
878	879	d	a	NaN	NaN	NaN	

    - 将其余为空数据填充为 0
2. 将日期拆解
    - 将日期拆解成为年月日，这年的第几周与第几天的形式
3. 对于打折信息与竞争对手信息进行处理
    - 将打折信息处理为是否有打折
    - 将竞争对手信息处理为是否开张
4. 查看 heatmap



执行过程

1. 划分验证集

   由于预测是有时间性的，所以将所给的 train 集按照日期排序，然后划分最后一部分为验证集来验证模型准确度。

2. 选用 Feature

   通过数据探索部分暂时选定的 categorial features 为：

```
'Day', 'DayOfYear', 'Store', 'DayOfWeek', 'PromoOpen',
'CompetitionDistance', 'AvgSalePerStore', 'ShopAvgOpen',
'PromoShopSales', 'WeekOfYear', 'Promo', 'Year', 'Month',
'StoreType'
```

3. 初始算法比较

Model	Initial RMSPE
LinearRegression	0.385
DecisionTree	0.322

XGB	0.345
LGB	0.317

4. gridsearch 的调参数

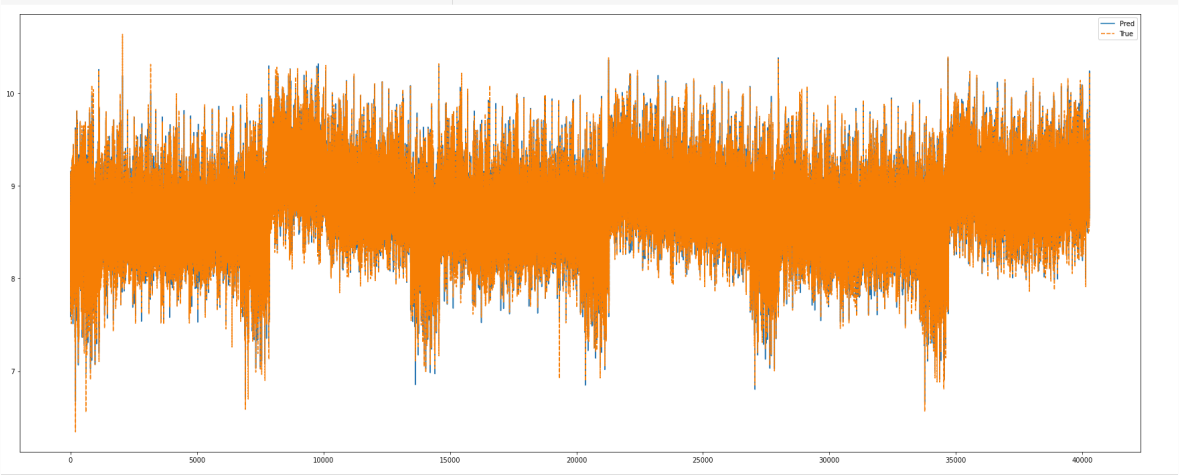
- 项目实现过程中使用了 LGB 与 XGBoost 算法
- 在基准模型的基础上使用了 GridSearch 进行控制变量地参数调整，最后选择了

```
LGBMRegressor(bagging_fraction=0.9, boosting_type='gbdt',
class_weight=None,
              colsample_bytree=0.8, feature_fraction=0.8,
              importance_type='split', learning_rate=0.1,
max_depth=-1,
              min_child_samples=20, min_child_weight=0.01,
min_split_gain=0.0,
              n_estimators=7263, n_jobs=-1, num_leaves=144,
              objective='regression', random_state=49,
reg_alpha=0.01,
              reg_lambda=0.001, silent=True, subsample=0.8,
              subsample_for_bin=200000, subsample_freq=1)
```

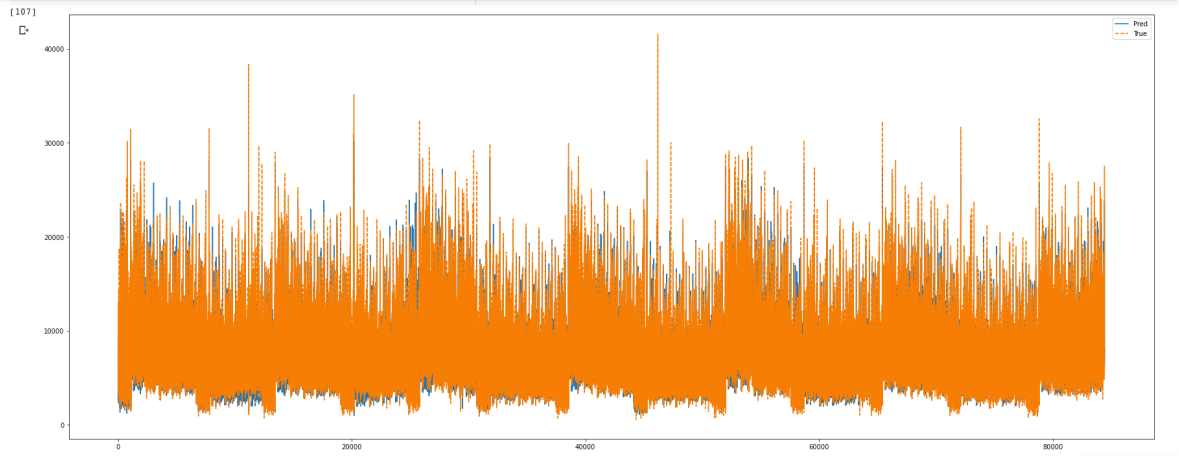
```
XGBRegressor(base_score=0.5, booster='gbtree',
colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.7,
gamma=0,
             importance_type='gain', learning_rate=0.03,
max_delta_step=0,
             max_depth=10, min_child_weight=1, missing=None,
n_estimators=2108,
             n_jobs=1, nthread=None,
objective='reg:squarederror',
             random_state=0, reg_alpha=0, reg_lambda=1,
scale_pos_weight=1,
             seed=None, silent=None, subsample=0.9,
tree_method='gpu_hist',
             verbosity=1)
```

5. 进行预测

XGB 预测结果



LGB 预测结果



完善

增加一些以商店 id 为基准的 feature

观察发现以商店 id 为基准下有很多数据是有参考价值的，因此加入 feature 中，实验结果进步很大，在 feature importance 中可发现占比还是比较大的。

IV. 结果

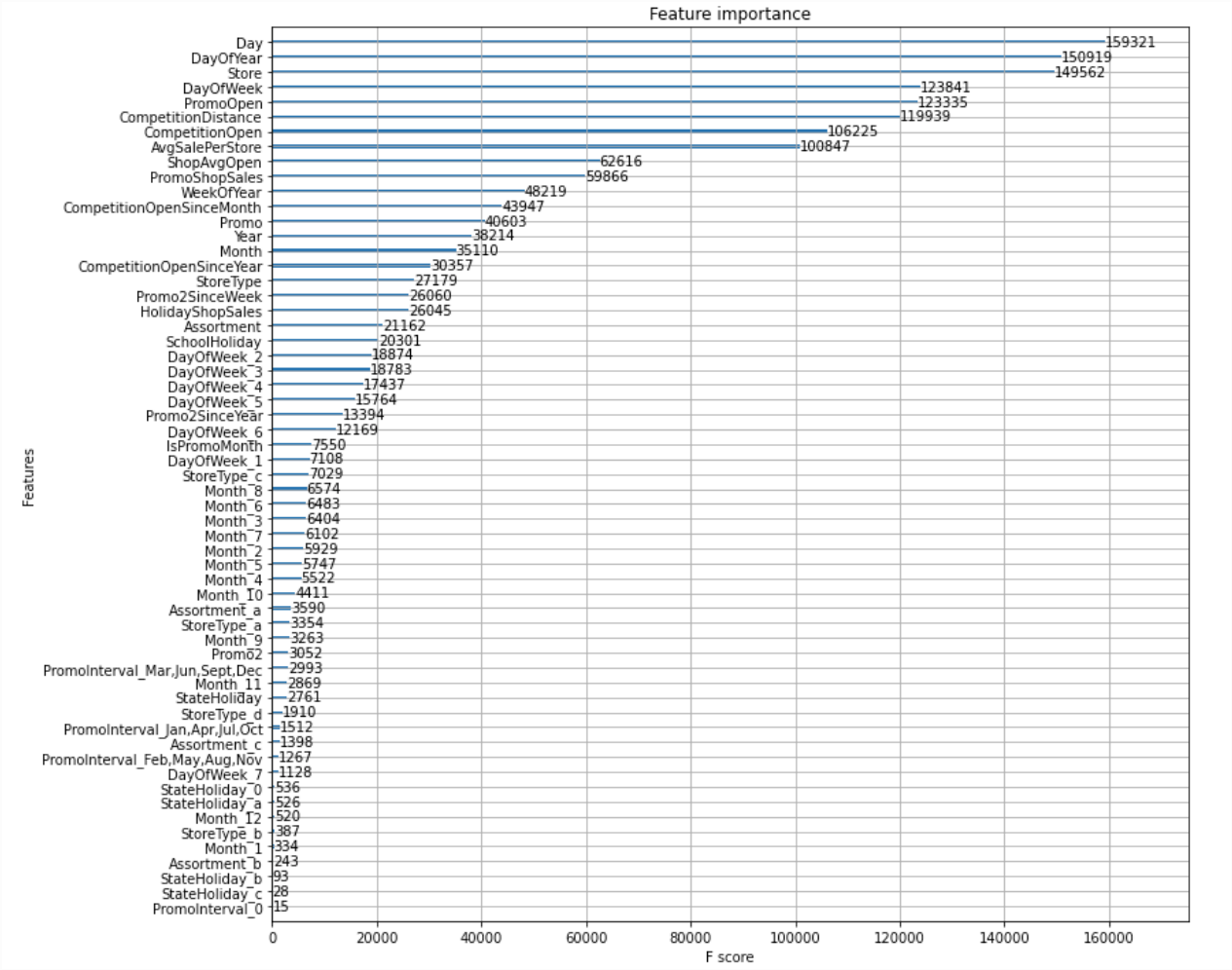
模型的评价与验证

XGB

Final Train Results

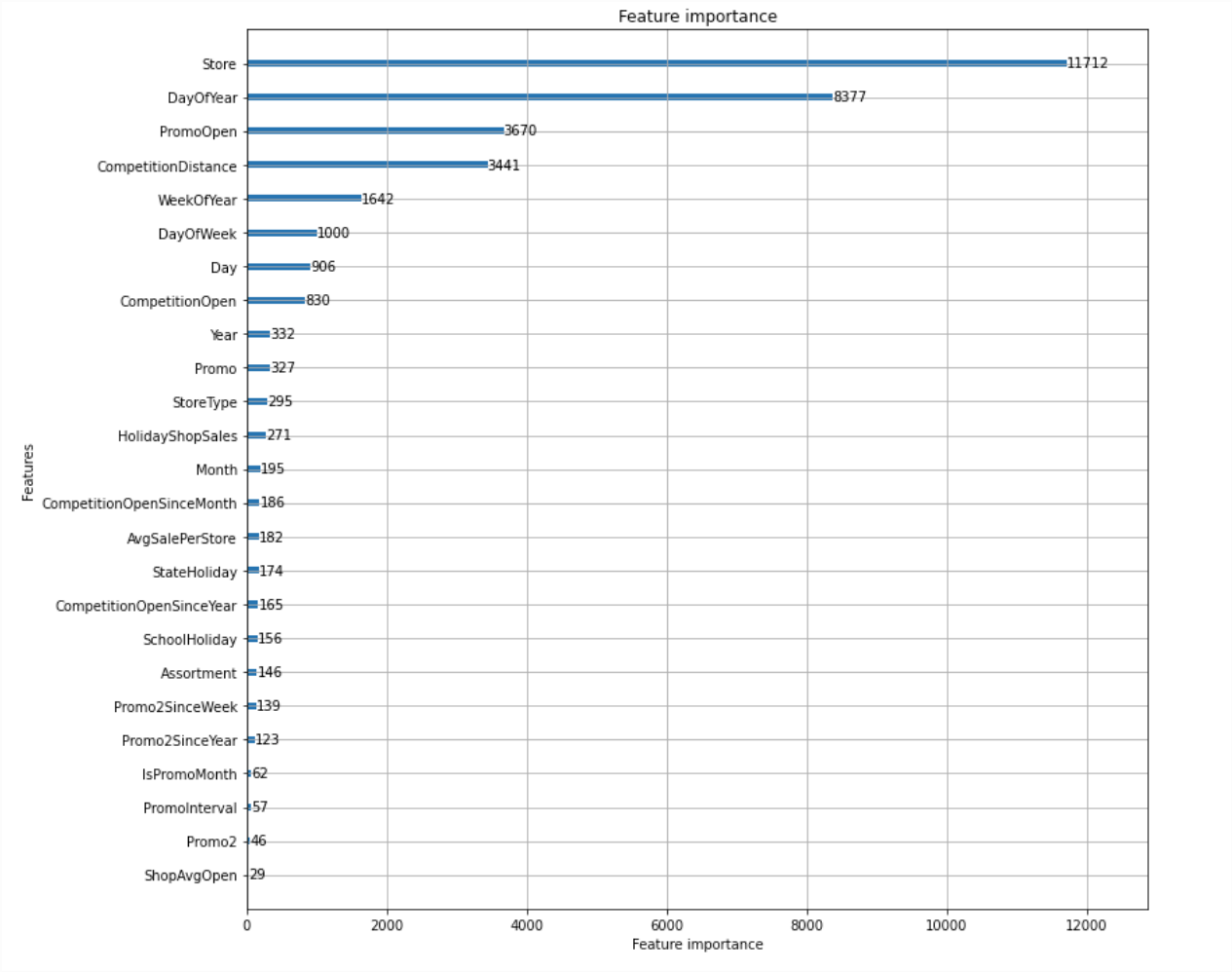
```
Will train until validation_0-rmspe hasn't improved in 100 rounds.
[100]   validation_0-rmse:0.444758      validation_0-rmspe:0.353499
[200]   validation_0-rmse:0.161195      validation_0-rmspe:0.200256
[300]   validation_0-rmse:0.129785      validation_0-rmspe:0.168512
[400]   validation_0-rmse:0.113178      validation_0-rmspe:0.150021
[500]   validation_0-rmse:0.102567      validation_0-rmspe:0.134289
[600]   validation_0-rmse:0.095952      validation_0-rmspe:0.125939
[700]   validation_0-rmse:0.0907        validation_0-rmspe:0.118542
[800]   validation_0-rmse:0.087042      validation_0-rmspe:0.11431
[900]   validation_0-rmse:0.084001      validation_0-rmspe:0.11025
[1000]  validation_0-rmse:0.081624      validation_0-rmspe:0.102826
[1100]  validation_0-rmse:0.07938       validation_0-rmspe:0.095269
[1200]  validation_0-rmse:0.077511      validation_0-rmspe:0.091722
[1300]  validation_0-rmse:0.07593       validation_0-rmspe:0.089288
[1400]  validation_0-rmse:0.074443      validation_0-rmspe:0.085823
[1500]  validation_0-rmse:0.073116      validation_0-rmspe:0.082742
[1600]  validation_0-rmse:0.071865      validation_0-rmspe:0.080582
[1700]  validation_0-rmse:0.070716      validation_0-rmspe:0.078107
[1800]  validation_0-rmse:0.069649      validation_0-rmspe:0.07615
[1900]  validation_0-rmse:0.068678      validation_0-rmspe:0.07454
[2000]  validation_0-rmse:0.067746      validation_0-rmspe:0.07297
[2100]  validation_0-rmse:0.066841      validation_0-rmspe:0.071779
[2107]  validation_0-rmse:0.066789      validation_0-rmspe:0.071725
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.7, gamma=0,
             importance_type='gain', learning_rate=0.03, max_delta_step=0,
             max_depth=10, min_child_weight=1, missing=None, n_estimators=2108,
             n_jobs=1, nthread=None, objective='reg:squarederror',
             random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
             seed=None, silent=None, subsample=0.9, tree_method='gpu_hist',
             verbosity=1)
```

XGB 的 fature important



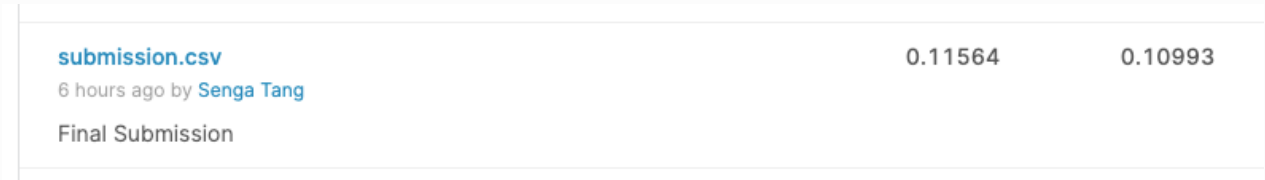
LGB

```
/usr/local/lib/python3.6/dist-packages/lightgbm/basic.py:1209: UserWarning: categorical_feature in Dataset is overridden.
New categorical_feature is ['AvgSalePerStore', 'CompetitionDistance', 'Day', 'DayOfWeek', 'DayOfYear', 'Month', 'Promo', 'PromoOpen', '
  'New categorical_feature is {}'.format(sorted(list(categorical_feature))))
Training until validation scores don't improve for 500 rounds.
[100]   training's l2: 421324   training's rmspe: 0.0960331     valid_1's l2: 1.4993e+06        valid_1's rmspe: 0.176452
[200]   training's l2: 304763   training's rmspe: 0.0833977     valid_1's l2: 1.46827e+06       valid_1's rmspe: 0.175488
[300]   training's l2: 249353   training's rmspe: 0.0768318     valid_1's l2: 1.46885e+06       valid_1's rmspe: 0.175744
[400]   training's l2: 210520   training's rmspe: 0.0716866     valid_1's l2: 1.46861e+06       valid_1's rmspe: 0.176083
[500]   training's l2: 182384   training's rmspe: 0.0675414     valid_1's l2: 1.47127e+06       valid_1's rmspe: 0.176426
[600]   training's l2: 160609   training's rmspe: 0.0639655     valid_1's l2: 1.47379e+06       valid_1's rmspe: 0.176672
[700]   training's l2: 142915   training's rmspe: 0.0608218     valid_1's l2: 1.47529e+06       valid_1's rmspe: 0.176822
Early stopping, best iteration is:
[241]   training's l2: 278682   training's rmspe: 0.0804436     valid_1's l2: 1.46553e+06       valid_1's rmspe: 0.175409
LGBMRegressor(bagging_fraction=0.9, boosting_type='gbdt', class_weight=None,
              colsample_bytree=0.8, feature_fraction=0.8,
              importance_type='split', learning_rate=0.1, max_depth=-1,
              min_child_samples=20, min_child_weight=0.01, min_split_gain=0.0,
              n_estimators=7263, n_jobs=-1, num_leaves=144,
              objective='regression', random_state=49, reg_alpha=0.01,
              reg_lambda=0.001, silent=True, subsample=0.8,
              subsample_for_bin=200000, subsample_freq=1)
```

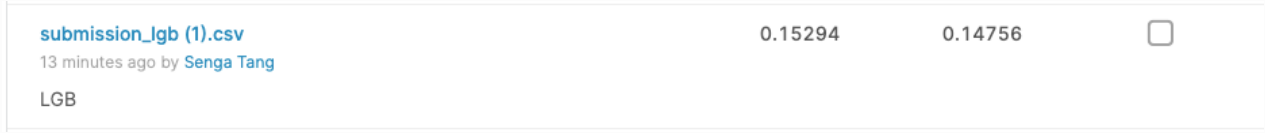


合理性分析

XGB



LGB

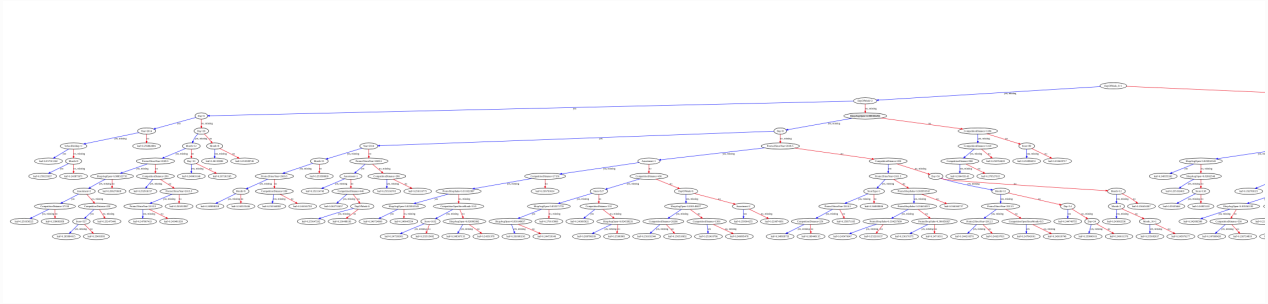


其中 XGB 符合项目要求，且表现较好

V. 项目结论

结果可视化

结果可视化见附件 xgb.dot.pdf 对树结构的可视化，缩略图截图如下：



对项目的思考

- 对于这些销售额的预测来说，数据处理占很大一部分的比重，将 raw data 和处理过的 data 放入模型中训练可以得出非常不同的结果。数据预处理非常重要。
- 刚开始做的的时候不知道从何下手，代码也是乱七八糟，应该早点开始的其实。

需要作出的改进

- 可以尝试使用神经网络或者 Entity Embedding 来实现