1.定义

1.1 项目概述

项目是针对 rosman 的历年销售额，进行分析建模以便可以预测未来的销售额。

Rossmann 是欧洲的一家连锁药店。 在这个源自 Kaggle 比赛 Rossmann Store Sales 中，我们需要根据 Rossmann 药妆店的信息（比如促销，竞争对手，节假日）以及在过去的销售情况，来预测 Rossmann 未来的销售额。

解决该问题涉及回归算法领域，数据集使用的 rosman 提供的销售数据以及门店信息数据。

1.2 问题陈述

项目选择的数据是来此 rosman 提供的真实数据，项目的挑战在于需要从众多的特征数据中整理分析出影响预测未开销售数据的重要特征。

问题分为两个部分，第一个部分是数据挖掘分析，第二个部分是模型的学习调参。

整个问题是一个回归问题，输入的数据由特征数据构成，目标是特征数据的预测销售额。

需要对项目的输入数据进行特征处理，获取相关的数据来训练回归模型。通过最后的模型训练，可以有效的对未来的数据进行预测。

这是一个有监督的回归问题，这里你使用的 XGBOOST 建模求解。

1.3 评价指标

对于回归问题，评价指标是 RMSPE，均方根百分比误差（Root Mean Square Percentage Error ）。

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。

。

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

2.分析

Xgboost 介绍

集成学习是很有用的学习方法，可以有效的增加机器学习的准确度并且可以防止过拟合。
主流的三种方式：bagging， boosting， stacking
这里主要运用 boosting 方法，而最为流行的就是 gradient boosting。
Xgboost 和 GBDT 的相同点：
每一棵树学的是之前所有树结论和的残差，这个残差就是一个加预测值后能得真实值的累加量.

三、XGBoost 原理

XGBoost 目标函数定义为：

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

Training loss　　Complexity of the Trees

$$where\ \Omega(f) = \gamma T + \frac{1}{2}\lambda ||w||^2$$

目标函数由两部分构成，第一部分用来衡量预测分数和真实分数的差距，另一部分则是正则化项。正则化项同样包含两部分，T 表示叶子结点的个数，w 表示叶子节点的分数。γ 可以控制叶子结点的个数，λ 可以控制叶子节点的分数不会过大，防止过拟合。
正如上文所说，新生成的树是要拟合上次预测的残差的，即当生成 t 棵树后，预测分数可以写成：

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$$

同时，可以将目标函数改写成：

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y_i}^{(t-1)} + f_t(\mathbf{x}_i)) + \Omega(f_t)$$

很明显，我们接下来就是要去找到一个 f_t 能够最小化目标函数。XGBoost 的想法是利用其在 f_t=0 处的泰勒二阶展开近似它。所以，目标函数近似为：

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n}[l(y_i, \hat{y}^{(t-1)}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2}h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

其中 g_i 为一阶导数，h_i 为二阶导数：

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

由于前 t-1 棵树的预测分数与 y 的残差对目标函数优化不影响，可以直接去掉。简化目标函数为：

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n}[g_i f_t(\mathbf{x}_i) + \frac{1}{2}h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

上式是将每个样本的损失函数值加起来，我们知道，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，过程如下图：

$$\begin{aligned} Obj^{(t)} &\simeq \sum_{i=1}^{n}\left[g_i f_t(x_i) + \tfrac{1}{2}h_i f_t^2(x_i)\right] + \Omega(f_t) \\ &= \sum_{i=1}^{n}\left[g_i w_{q(x_i)} + \tfrac{1}{2}h_i w^2_{q(x_i)}\right] + \gamma T + \lambda\tfrac{1}{2}\sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T}\left[(\sum_{i\in I_j} g_i)w_j + \tfrac{1}{2}(\sum_{i\in I_j} h_i + \lambda)w_j^2\right] + \gamma T \end{aligned}$$

因此通过上式的改写，我们可以将目标函数改写成关于叶子结点分数 w 的一个一元二次函数，求解最优的 w 和目标函数值就变得很简单了，直接使用顶点公式即可。因此，最优的 w 和目标函数公式为

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

Xgboost 和 GBDT 的区别：

(1)传统 GBDT 以 CART 作为基分类器，xgboost 还支持线性分类器，这个时候 xgboost 相当于带 L1 和 L2 正则化项的逻辑斯蒂回归（分类问题）或者线性回归（回归问题）。节点分裂的方式不同，gbdt 是用的 gini 系数，xgboost 是经过优化推导后的

(2)传统 GBDT 在优化时只用到一阶导数信息，xgboost 则对代价函数进行了二阶泰勒展开，同时用到了一阶和二阶导数。为什么 xgboost 要用泰勒展开，优势在哪里？xgboost 使用了一

阶和二阶偏导, 二阶导数有利于梯度下降的更快更准. 使用泰勒展开取得函数做自变量的二阶导数形式, 可以在不选定损失函数具体形式的情况下, 仅仅依靠输入数据的值就可以进行叶子分裂优化计算, 本质上也就把损失函数的选取和模型算法优化/参数选择分开了. 这种去耦合增加了 xgboost 的适用性, 使得它按需选取损失函数, 可以用于分类, 也可以用于回归。
(3)Xgboost 在代价函数里加入了正则项，用于控制模型的复杂度，降低了过拟合的可能性。正则项里包含了树的叶子节点个数、每个叶子节点上输出的 score 的 L2 模的平方和
(4)Xgboost 工具支持并行。boosting 不是一种串行的结构吗?怎么并行的？注意 xgboost 的并行不是 tree 粒度的并行，xgboost 也是一次迭代完才能进行下一次迭代的（第 t 次迭代的代价函数里包含了前面 t-1 次迭代的预测值）。xgboost 的并行是在特征粒度上的。我们知道，决策树的学习最耗时的一个步骤就是对特征的值进行排序（因为要确定最佳分割点），xgboost 在训练之前，预先对数据进行了排序，然后保存为 block 结构，后面的迭代中重复地使用这个结构，大大减小计算量。这个 block 结构也使得并行成为了可能，在进行节点的分裂时，需要计算每个特征的增益，最终选增益最大的那个特征去做分裂，那么各个特征的增益计算就可以开多线程进行

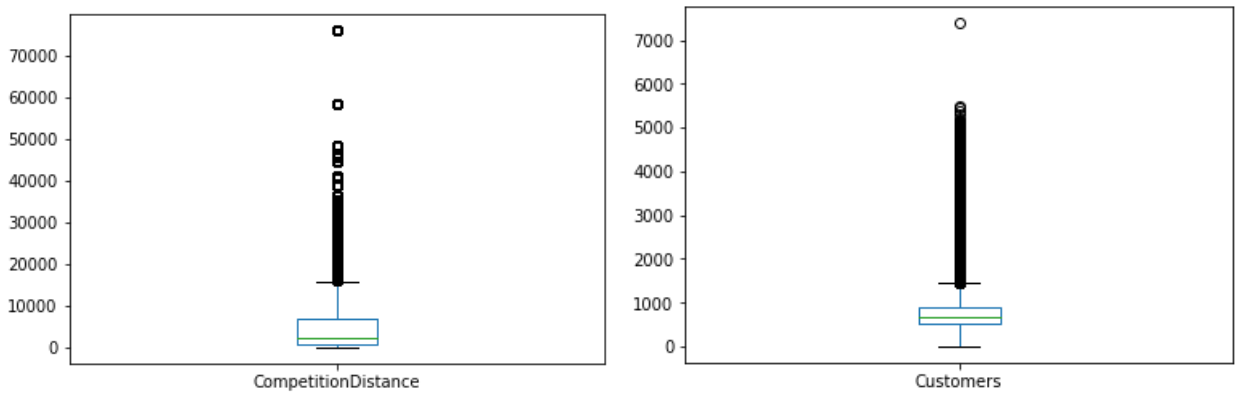
数据可视化

Rosman 数据集分为训练数据集，店铺信息数据集，测试数据集。训练数据集中包括了销售相关的外部环境因素特征。比如销售当天的客户数，是否是国定节假日，是否是学校 oen 日等。
店铺相关数据是代表了销售所在的店铺的相关信息，比如商店销售物品的等级划分，促销的实行情况以及竞争对手距离商店位置的相关信息。
通过这些数据可以有效的进行预测学习。
该训练数据集包括两类，历史销售数据以及每个商店的个体信息，我们可以通过结合这两个数据集来训练预测。
对两类数据合并后有如下的训练特征：

```
['Store', 'DayOfWeek', 'Date', 'Sales', 'Customers', 'Open', 'Promo',
       'StateHoliday', 'SchoolHoliday', 'StoreType', 'Assortment',
       'CompetitionDistance', 'CompetitionOpenSinceMonth',
       'CompetitionOpenSinceYear', 'Promo2', 'Promo2SinceWeek',
       'Promo2SinceYear', 'PromoInterval']
```

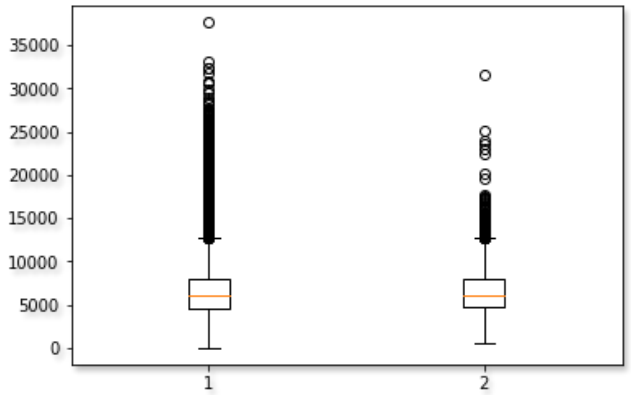
分析异常值：

通过对 customer，CompetitionDistance 存在异常值

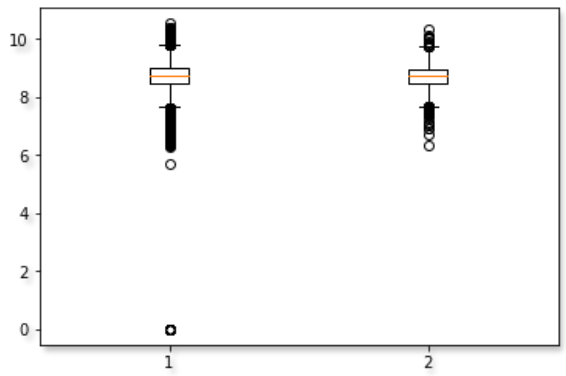


我们可以对异常值进行删除，但是考虑到我们这次用的算法是 xgboost，所以不需要对异常值做太多的处理

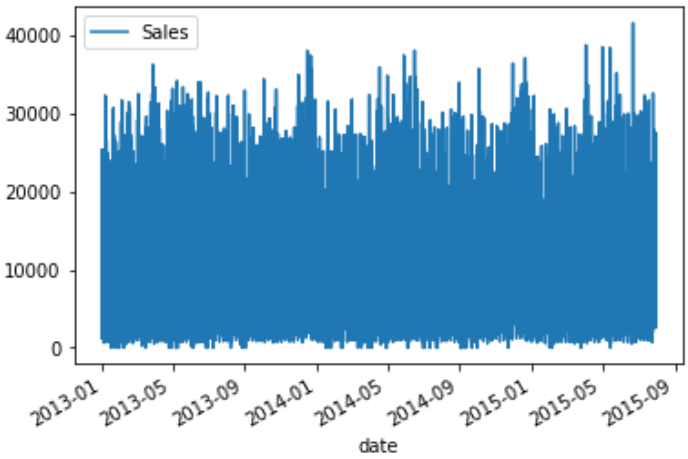
目标数据是 Sales 特征，对这个特征的盒线图显示，数据存在比较大的异常偏差。



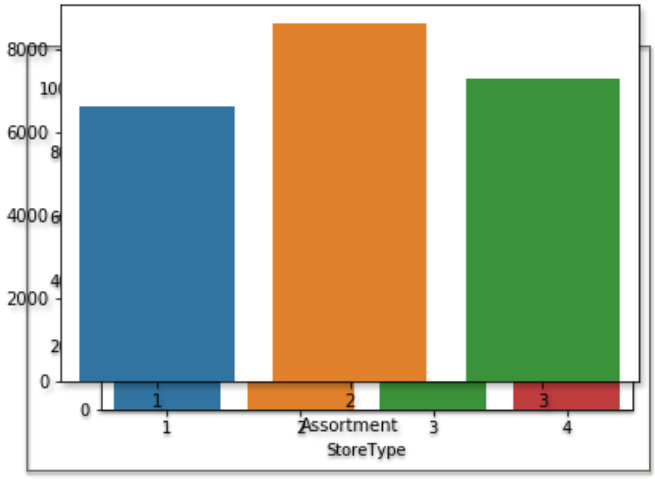
所以对标签数据进行平滑处理，使其服从正态分布。使用 log 平滑处理。



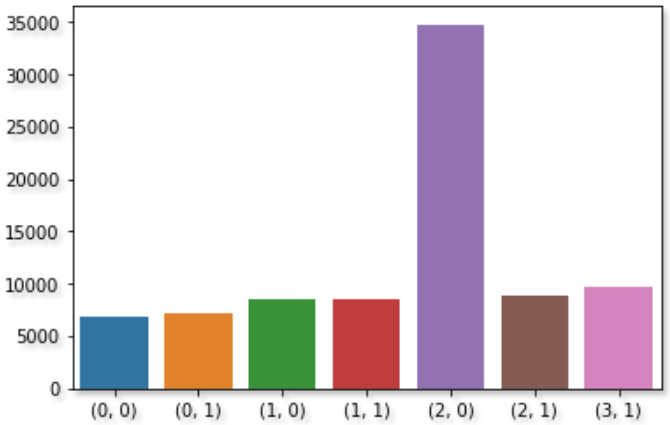
Sales 也是这次项目的预测标签，查看下从 13 年到 15 年的 sales 数据趋势



StoreType 对于销售额的影响很大，从柱形图就可以看出不同 type 的平均销售额是有差别的

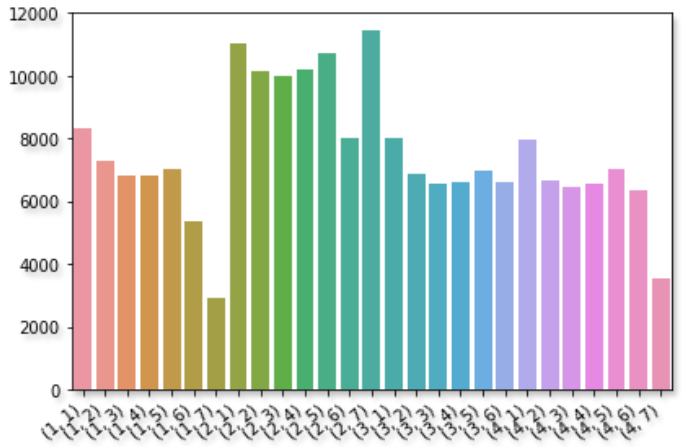


发现在不同的节假日的情况下，学校的开关也对销售额有很大的影响



从直觉上来说 Assortment 也会影响销售额

DayOfWeek 对于销售额也影响



通过绘制图获取的 feature 特征包括

['DayOfWeek', 'Sales', 'Promo', 'StateHoliday', 'SchoolHoliday',
'StoreType', 'Assortment', 'CompetitionDistance',
'CompetitionOpenSinceMonth', 'CompetitionOpenSinceYear', 'Promo2',
'Promo2SinceWeek', ‘Promo2SinceYear']

在数据集中有很多标签数据，统一进行 label encode

```
mappings = {'0':0, 'a':1, 'b':2, 'c':3, 'd':4}
data.StoreType.replace(mappings, inplace=True)
data.Assortment.replace(mappings, inplace=True)
data.StateHoliday.replace(mappings, inplace=True)
```

使其适合模型的训练运算

算法和技术

算法：

回归算法

根据前面的分析可知，销售额预测本质上是一个回归问题。对于回归的算法有很多，比如线性回归，多项式回归，支持向量机， CART 回归树等。这些算法都是机器学习的算法。 通过将所需要训练的数据特征转化为特征向量，输出的是预测的最终销售数据。

线性回归顾名思义是求得线性预测的算发，然而对于有这么多特征的数据来说线性划分并不可靠，容易欠拟合。

多项式回归能够对于多特征进行有效的拟合预测，可以有效的对特征数据进行分割，模拟。

CART 是一种二分递归分割的技术，分割方法采用基于最小距离的基尼指数估计函数，将当前的样本集分为两个子样本集，使得生成的的每个非叶子节点都有两个分支。因此，CART 算法生成的决策树是结构简洁的二叉树。回归树是针对目标变量是连续性的变量，通过选取最优分割特征的某个值，然后数据根据大于或者小于这个值进行划分进行树分裂最终生成回归树。

增强算法

处理具体问题时，单一的回归树肯定是不够用的。可以利用集成学习中的 boosting 框架，对回归树进行改良升级，得到的新模型就是提升树（Boosting Decision Tree），在进一步，可以得到梯度提升树（Gradient Boosting Decision Tree，GBDT），再进一步可以升级到 XGBoost。

提升方法（Boosting），是一种可以用来减小监督式学习中偏差的机器学习算法。面对的问题是迈可·肯斯（Michael Kearns）提出的：一组“弱学习者”的集合能否生成一个“强学习者”？弱学习者一般是指一个分类器，它的结果只比随机分类好一点点；强学习者指分类器的结果非常接近真值。

大多数提升算法包括由迭代使用弱学习分类器组成，并将其结果加入一个最终的成强学习分类器。加入的过程中，通常根据它们的分类准确率给予不同的权重。加和弱学习者之后，数据通常会被重新加权，来强化对之前分类错误数据点的分类。

一个经典的提升算法例子是 AdaBoost。一些最近的例子包括 LPBoost、TotalBoost、BrownBoost、MadaBoost 及 LogitBoost。许多提升方法可以在 AnyBoost 框架下解释为在函数空间利用一个凸的误差函数作梯度下降。

技术：

在这个项目中主要运用的是 xgboost 算法。xgboost 一直在竞赛江湖里被传为神器，比如时不时某个 kaggle/天池比赛中，某人用 xgboost 于千军万马中斩获冠军。

因为这个问题是回归问题，xgboost 内部使用的 CART tree，这种结构可以处理分类回归问题。而且 xgboost 的特点是它能够自动利用 CPU 的多线程进行并行，同时在算法上加以改进提高了精度。也是一种集成方法。集成方法的有点就是采用很多弱学习器最后合并成强学习器，效果会比较好。

Xgboost 在学习的过程中， 或不断的通过上一次学习的模型的残差进一步学习，最终将损失缩小。

基准指标：

对于项目，rosman 规定使用 RMSPE 进行指标评估，根据 Kaggle 竞赛的分数排名，RMSPE 的得分在 0.11 左右是比较好的成绩。

具体方法

数据预处理

考虑到数据集中存在很多的 label 标签，不利于算法的运行，所以需要对其进行 labelEncoding。

```
mappings = {'0':0, 'a':1, 'b':2, 'c':3, 'd':4}
data.StoreType.replace(mappings, inplace=True)
data.Assortment.replace(mappings, inplace=True)
data.StateHoliday.replace(mappings, inplace=True)
```

目标数据存在很大的离群值，所以需要进行平滑处理。让目标数据的分布比较平衡。能够更好的学习模型。

训练集经过随机的打乱，一部分用于交叉验证，即在训练中衡量模型的预测性能。

对于该项目，将所销售训练集和 store 的信息数据集合并在一起，1.2%作为验证数据，剩余的作为训练数据。最终训练集（834259 样本），验证集（10133 样本）。

执行过程

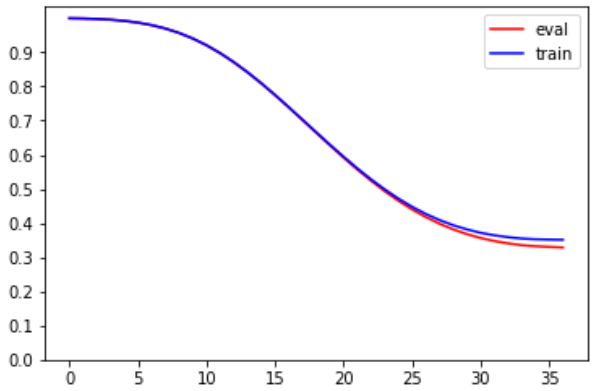
对于时间序列数据，如果考虑到时间因素，应该会对模型带来提升，所以讲时间数据加入特征集

```
#    data['Year'] = data.Date.dt.year
#    data['Month'] = data.Date.dt.month
#    data['Day'] = data.Date.dt.day
```

初始版本对所有的特征进行向量化之后，传入 xgboost。随机设置分类数的深度学习率等参数，

初始版本	训练集	验证集	
"eta": 0.1,	0.351623	0.352851	

"max_depth": 5, "min_child_weight":1, "gamma":0, "subsample": 0.7, "colsample_bytree": 0.7,			




```
[85]    eval-rmse:0.301548    train-rmse:0.300529    eval-rmspe:0.324166    tra
in-rmspe:0.358648
[86]    eval-rmse:0.301375    train-rmse:0.300338    eval-rmspe:0.323935    tra
in-rmspe:0.358438
Stopping. Best iteration:
[36]    eval-rmse:0.374072    train-rmse:0.373598    eval-rmspe:0.328553    tra
in-rmspe:0.35109

Train use time: 130.53800010681152
```

数据训练到 492 次迭代的时候停止了，并且通过 cv 返回的数据得知目前只需要 492 棵决策树的数目

完善

模型没有进行过任何改进，结果明显不够好，训练的拟合度不够。

进行参数调优，首先对 max_depth 和 min_weight 参数调优，因为它们对最终结果有很大的影响。首先，我们先大范围地粗调参数，然后再小范围地微调。使用栅格搜索(grid search)对参数进行调优。

首先第一步是对 max_deepth 进行调参

```
param_test1 = {
 'max_depth':range(10,13,1),
}
params_fit = {
    'eval_metric':rmspe_xg,
    'early_stopping_rounds':20,
    'eval_set':eval_set
```

}

nums_round = 492

gsearch1 = GridSearchCV(estimator = xgb.XGBRegressor(max_depth=12,learning_rate = 0.1,n_jobs =4, n_estimators=nums_round,subsample=0.8,colsample_bytree=0.8,nthread= 4,scale_pos_weight=1),param_grid = param_test1,cv=3,verbose=2,scoring=make_scorer(c ustomer_rmspe_xg,greater_is_better=False),fit_params = params_fit)

gsearch1.fit(X_train, Y_train)

获得最佳的参数是{'max_depth': 10}

然后针对 subsample 进行调参

```
param_test2 = {
 'subsample': [0.8,0.9],
}
nums_round = 492
gsearch2 = GridSearchCV(estimator = xgb.XGBRegressor(max_depth=10,learning_r
ate =0.1,n_jobs =4, n_estimators=nums_round,subsample=0.8,colsample_bytree=
0.8,nthread=4,scale_pos_weight=1),
                        param_grid = param_test2,cv=3,verbose=2,scoring=make
_scorer(customer_rmspe_xg,greater_is_better=False),fit_params = params_fit)
gsearch2.fit(X_train, Y_train)
```

获得最佳的参数是{' subsample’ : 0.8}

再对学习率 eta 进行调参

```
param_test3 = {
 'eta': [0.08,0.1,0,12,0,15,0,2],
}
nums_round = 492
gsearch3 = GridSearchCV(estimator = xgb.XGBRegressor(max_depth=10,learning_r
ate =0.1,n_jobs =4, n_estimators=nums_round,subsample=0.8,colsample_bytree=
0.8,nthread=4,scale_pos_weight=1),
                        param_grid = param_test3,verbose=2,scoring=make_scor
er(customer_rmspe_xg,greater_is_better=False),fit_params = params_fit)
gsearch3.fit(X_train, Y_train)
```

获得最佳的参数是{'eta ‘: 0.1}

最后针对参数继续对 gamma 进行优化，

param_test4 = {

'gamma':[i/10.0 for i in range(0,5)]

}

nums_round = 492

gsearch4 = GridSearchCV(estimator = xgb.XGBRegressor(max_depth=10,learning_rate = 0.1,n_jobs =4, n_estimators=nums_round,subsample=0.8,colsample_bytree=0.8,nthread= 4,scale_pos_weight=1),

param_grid = param_test4,verbose=2,scoring=make_scorer(customer_rmspe_xg,greater_is_better=False),fit_params = params_fit)

gsearch4.fit(X_train, Y_train)

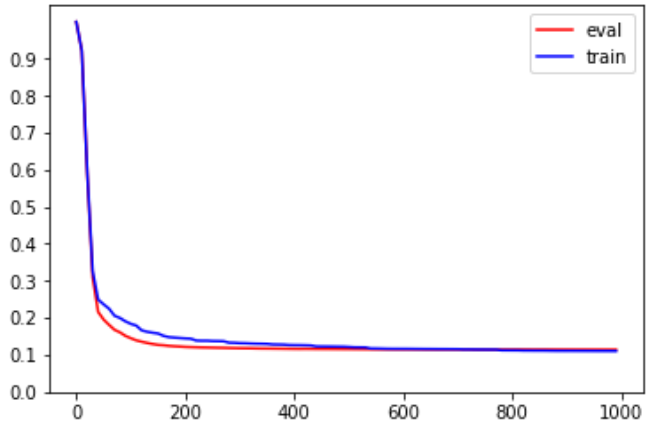
获得的最佳 gamma `{'gamma': 0.4}`

使用训练得到的参数，组合后再次对训练数据进行训练

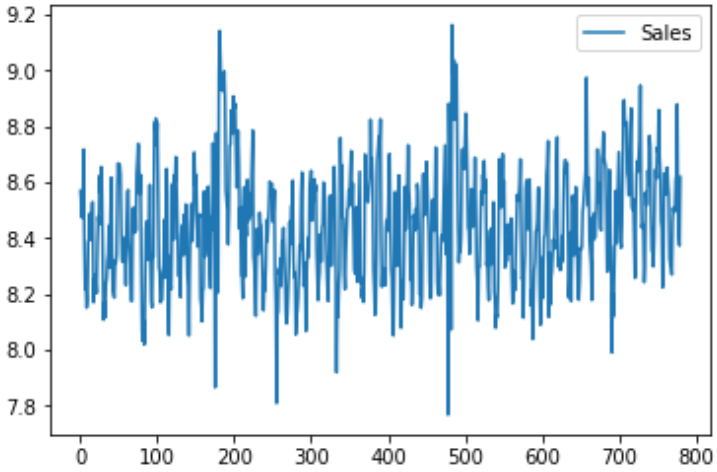
```
estimator = xgb.XGBRegressor(
max_depth=10,
learning_rate =0.1,
n_jobs =4,
gamma = 0.4,
n_estimators=num_trees,
subsample=0.8,
colsample_bytree=0.7,
nthread=4,
scale_pos_weight=1)
```

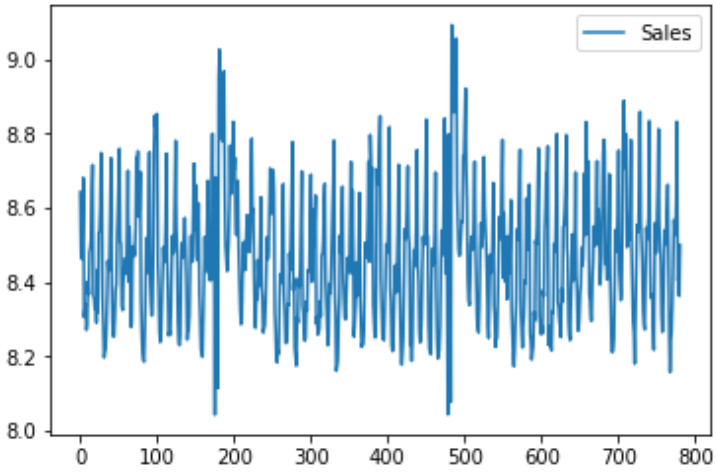
estimator.fit(X_train, Y_train,eval_metric=rmspe_xg, eval_set=eval_set, verbose=True,early_stopping_rounds=20)

最终获得验证数据 `eval-rmspe: 0.11574`

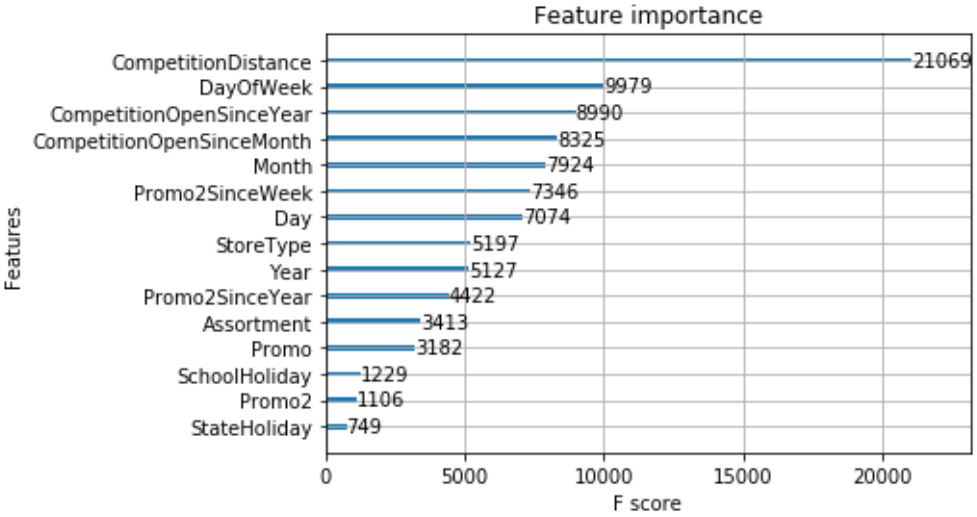


通过对比原始商店 1 的真实数据和模型预测数据





我们可以看出预测值和真实值还是比较接近的。



可以看出最终的特征重要性上时间是一个很重要的特征，以后可以针对时间特征构造一些新的自定义时间特征，应该有助于优化结果。

结果

使用该训练模型对测试数据进行预测，最终的到的 kaggle 分数在 0.11605。从最终得分上来说预测的结果没有特别好。从上图中也可以看到，在训练的过程中，训练数据存在偏差，应该对特征数据进行更具体的分析和提取应该可以进一步改善问题。

xgboost_submission.csv 18 days ago by dboyaoao add submission details	0.13387	0.11605

总体来说，现在这个模型表现比初始状态还是有所改进。

结论

Rosman 项目通过分析数据，可视化，对数据进一步获取了解，然后运用回归处理技术。有很多回归技术可以用，但是 xgboost 是目前最为流行的，速度快强大，容易使用上手。

效果不是特别理想，应该还有更有效的时间特征可以提取。与 kaggle 最好的成绩想差 0.2 左右，还是有很多改进的空间。一个后续的改进是希望能通过对特征数据进行进一步的提取，获取到更多可以表示销售情况的特征。毕竟模型的参数的优化所能带来的提升是有限的。

参考文献

https://zhuanlan.zhihu.com/p/54334329

https://www.kaggle.com/c/rossmann-store-sales/discussion/18024

https://blog.csdn.net/han_xiaoyang/article/details/52665396

https://zhuanlan.zhihu.com/p/34679467

https://zhuanlan.zhihu.com/p/40129825