# 1、项目的背景与目的

**使用商店、促销和竞争对手数据预测销售**

Rossmann在欧洲国家经营着3000多家日化用品超市。目前，Rossmann商店经理的任务是提前6周预测他们的日销售额。商店的销售受到许多因素的影响，包括促销、竞争、学校和国家假日、季节性和地域性。由于数以千计的管理者根据自己的特殊情况预测销售，结果的准确性可能会有很大的差异。

因此使用机器学习算法对销量进行预测，Rossmann要求预测德国1115家商店的6周日销售额。可靠的销售预测使商店经理能够制定有效的员工时间表，提高生产力和积极性。


这就是算法和零售、物流领域的一次深度融合，从而提前备货，减少库存、提升资金流转率，促进公司更加健康发展，为员工更合理的工作、休息提供合理安排，为工作效率提高保驾护航。

# 2、数据介绍

- **train.csv** - 包含销售情况的历史数据文件
- **test.csv** - 不包含销售情况的历史数据文件
- **sample_submission.csv** - 数据提交样本文件
- **store.csv** - 商店更多信息文件

| 字段 | 说明 |
| --- | --- |
| Store | 每个商店唯一的ID |
| Sales | 销售额 |
| Customers | 销售客户数 |
| Open | 商店是否营业 0=关闭，1=开业 |
| StateHoliday | 国家假日 |
| SchoolHoliday | 学校假期 |
| StoreType | 店铺类型: a, b, c, d |
| Assortment | 产品组合级别: a = 基本, b = 附加, c = 扩展 |
| CompetitionDistance | 距离最近的竞争对手距离(米) |
| CompetitionOpenSince[Month/Year] | 最近的竞争对手开业时间 |
| Promo | 指店铺当日是否在进行促销 |
| Promo2 | 指店铺是否在进行连续促销 0 = 未参与, 1 = 正在参与 |
| Promo2Since[Year/Week] | 商店开始参与Promo2的时间 |
| PromoInterval | 促销期 |

- StateHoliday：通常所有商店都在国家假日关门：a = 公共假日， b = 复活节假日，c = 圣诞节，0 = 无

# 3、数据加载

## 3.1、查看数据

```
#导入需要的库
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import xgboost as xgb
from time import time

#导入数据集
store=pd.read_csv('./data/store.csv')
train=pd.read_csv('./data/train.csv',dtype={'StateHoliday':np.string_}) # 指定数据类型
test=pd.read_csv('./data/test.csv',dtype={'StateHoliday':np.string_})

#观察下数据的基本情况
display(store.head(),train.head(),test.head())
```

- StateHoliday中数据为：a = 公共假日， b = 复活节假日，c = 圣诞节，0 = 无。既有字符串也有数字，需要指定类型
- 数据中存在空数据，一次需要处理空数据

## 3.2、空数据处理

### 3.2.1、训练数据处理

```
train.isnull().sum()
'''
Store            0
DayOfWeek        0
Date             0
Sales            0
Customers        0
Open             0
Promo            0
StateHoliday     0
SchoolHoliday    0
dtype: int64
'''
```

train数据无缺失，因此无需处理

### 3.2.2、测试数据处理

```
#test数据Open列有缺失
test.isnull().sum()
'''
Id                0
Store             0
DayOfWeek         0
Date              0
Open             11
Promo             0
StateHoliday      0
SchoolHoliday     0
dtype: int64
'''
#查看test缺失列都来自于622号店
test[test['Open'].isnull()]
#通过查看train里622号店的营业情况发现，622号店周一到周六都是营业的
train[train['Store']==622]
#所以我们认为缺失的部分是应该正常营业的，用1填充
test.fillna(1,inplace=True)
```

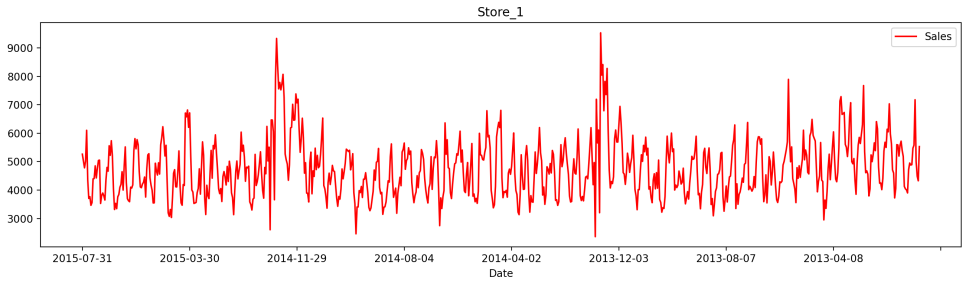
### 3.3.3、商店数据处理

```
#store列缺失值较多，但数量看来比较一致，看一下是否同步缺失
store.isnull().sum()
'''
Store                        0
StoreType                    0
Assortment                   0
CompetitionDistance          3
CompetitionOpenSinceMonth  354
CompetitionOpenSinceYear   354
```

```
Promo2                        0
Promo2SinceWeek             544
Promo2SinceYear             544
PromoInterval               544
dtype: int64
'''
#下面是观察store缺失的情况
v1='CompetitionDistance'
v2='CompetitionOpenSinceMonth'
v3='CompetitionOpenSinceYear'
v4='Promo2SinceWeek'
v5='Promo2SinceYear'
v6='PromoInterval'

# v2和v3是同时缺失
store[(store[a2].isnull())&(store[a3].isnull())].shape
'''
(354, 10)
'''
# V4、V5、V6也是同时缺失
store[(store[a4].isnull())&(store[a5].isnull())&(store[a6].isnull())].shape
'''
(544, 10)
'''
# V4、V5、V6列缺失是因为没有活动
set(store[(store[a4].isnull())&(store[a5].isnull())&(store[a6].isnull())]
['Promo2'])
'''
{0}
'''
# 下面对缺失数据进行填充
# 店铺竞争数据缺失，而且缺失的都是对应的。
# 原因不明，而且数量也比较多，如果用中值或均值来填充，有失偏颇。暂且填0，解释意义就是刚开业
# 店铺促销信息的缺失是因为没有参加促销活动，所以我们以0填充
store.fillna(0,inplace=True)
```

### 3.3.4、销量时间关系

```
#分析店铺销量随时间的变化
sales_data = train[train['Sales'] > 0]
sales_data.loc[strain['Store'] ==
1].plot(x='Date',y='Sales',title='Store_1',figsize=(16,4))
```




从图中可以看出店铺的销售额是有周期性变化的，一年中11,12月份销量相对较高，可能是季节（圣诞节）因素或者促销等原因。
此外从2014年6-9月份的销量来看，6,7月份的销售趋势与8,9月份类似，而我们需要预测的6周在2015年8,9月份，因此我们可以把2015年6,7月份最近6周的1115家店的数据留出作为测试数据，用于模型的优

化和验证!

## 4、合并数据

```
#我们只需要销售额大于0的数据
train=train[train['Sales'] > 0]
#把store基本信息合并到训练和测试数据集上
train = pd.merge(train,store,on='Store',how='left')
test = pd.merge(test,store,on='Store',how='left')

train.info()
'''
<class 'pandas.core.frame.DataFrame'>
Int64Index: 844338 entries, 0 to 844337
Data columns (total 18 columns):
 #   Column                        Non-Null Count   Dtype
---  ------                        --------------   -----
 0   Store                         844338 non-null  int64
 1   DayOfWeek                     844338 non-null  int64
 2   Date                          844338 non-null  object
 3   Sales                         844338 non-null  int64
 4   Customers                     844338 non-null  int64
 5   Open                          844338 non-null  int64
 6   Promo                         844338 non-null  int64
 7   StateHoliday                  844338 non-null  object
 8   SchoolHoliday                 844338 non-null  int64
 9   StoreType                     844338 non-null  object
 10  Assortment                    844338 non-null  object
 11  CompetitionDistance           844338 non-null  float64
 12  CompetitionOpenSinceMonth     844338 non-null  float64
 13  CompetitionOpenSinceYear      844338 non-null  float64
 14  Promo2                        844338 non-null  int64
 15  Promo2SinceWeek               844338 non-null  float64
 16  Promo2SinceYear               844338 non-null  float64
 17  PromoInterval                 844338 non-null  object
dtypes: float64(5), int64(8), object(5)
memory usage: 122.4+ MB
'''
```

## 5、特征工程

```
for data in [train,test]:
    # 将时间特征进行拆分和转化
    data['year']=data['Date'].apply(lambda x:x.split('-')[0]).astype(int)
    data['month']=data['Date'].apply(lambda x:x.split('-')[1]).astype(int)
    data['day']=data['Date'].apply(lambda x:x.split('-')[2]).astype(int)

    # 将'PromoInterval'特征转化为'IsPromoMonth'特征，表示某天某店铺是否处于促销月，1表示是，0表示否
    # 提示下：这里尽量不要用循环，用这种广播的形式，会快很多。循环可能会让你等的想哭
    month2str= {1:'Jan',2:'Feb',3:'Mar',4:'Apr',5:'May',6:'Jun',7:'Jul',8:'Aug',9:'Sep',10:'Oct',11:'Nov',12:'Dec'}
    data['monthstr'] = data['month'].map(month2str)
    convert = lambda x: 0 if x['PromoInterval'] == 0 else 1 if x['monthstr'] in x['PromoInterval'] else 0
```

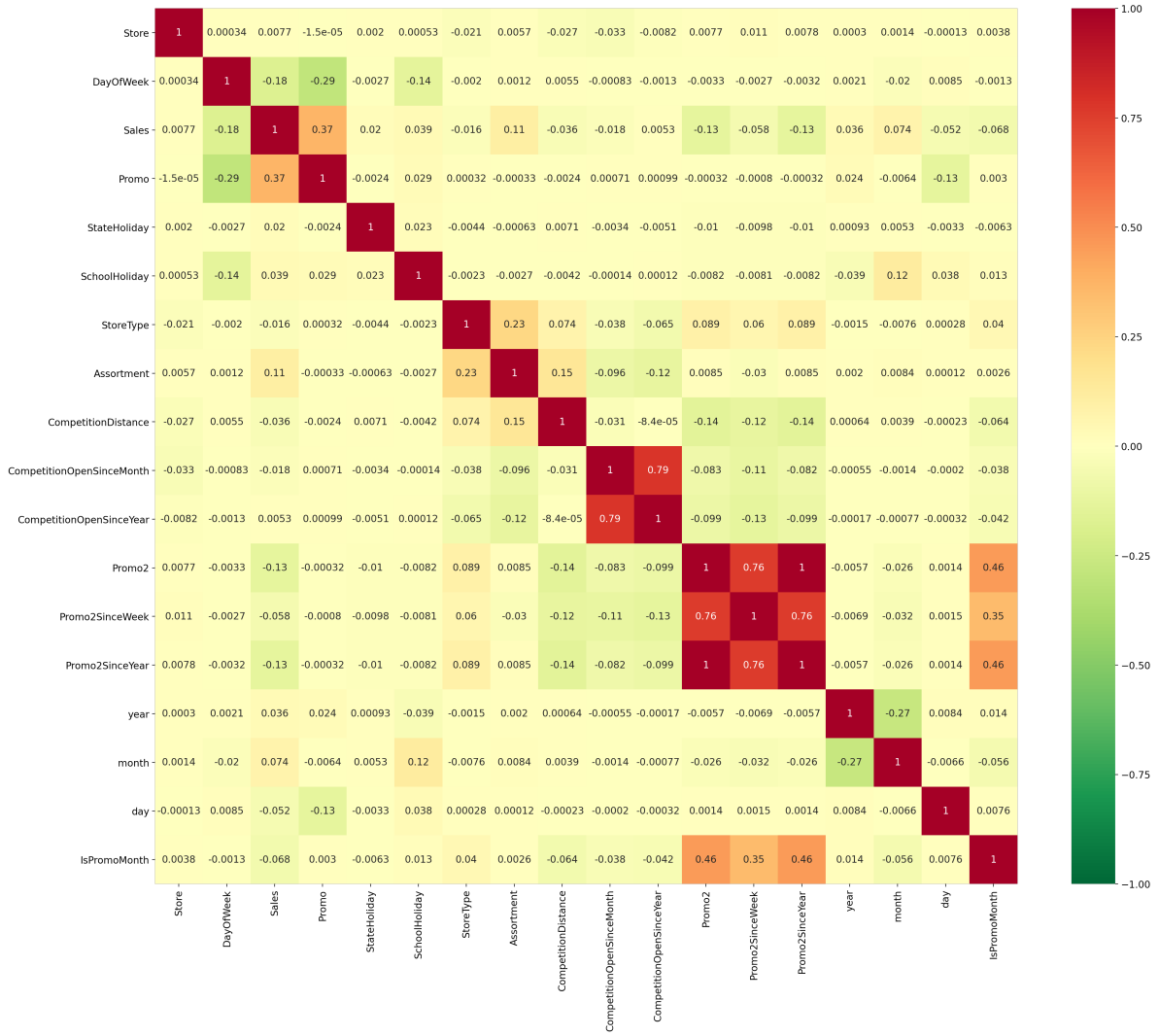
```
    data['IsPromoMonth']= data.apply(convert, axis=1)

    # 将存在其它字符表示分类的特征转化为数字
    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)
```

## 6、构建训练数据和测试数据

```
# 删掉训练和测试数据集中不需要的特征
df_train=train.drop(['Date','Customers','Open','PromoInterval','monthstr'],axis=
1)
df_test=test.drop(['Id','Date','Open','PromoInterval','monthstr'],axis=1)

# 如上所述，保留训练集中最近六周的数据用于后续模型的测试
X_train=df_train[6*7*1115:]
X_test=df_train[:6*7*1115]
```

## 7、数据属性间相关性系数

```
plt.figure(figsize=(24,20))
plt.rcParams['font.size'] = 12
sns.heatmap(df_train.corr(),cmap='RdYlGn_r',annot=True,vmin=-1,vmax=1)
```




# 8、提取模型训练的数据集

```
#拆分特征与标签，并将标签取对数处理
y_train = np.log1p(X_train['Sales'])
y_test = np.log1p(X_test['Sales'])

X_train = X_train.drop(['Sales'],axis=1)
X_test = X_test.drop(['Sales'],axis=1)
```

使用np.log1p进行对目标值处理，目标值更加正态化！

# 9、构建模型

## 9.1、定义评价函数

均方根百分比误差

$$RMSPE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(1-\frac{\hat{y}}{y})}$$

```
#定义评价函数，可以传入后面模型中替代模型本身的损失函数
def rmspe(y,yhat):
    return np.sqrt(np.mean((1 - yhat/y)**2))

# 起放大作用
def rmspe_xg(yhat,y):
    y=np.expm1(y.get_label())
    yhat=np.expm1(yhat)
    return 'rmspe',rmspe(y,yhat)
```

## 9.2、模型训练

```
#初始模型构建
#参数设定
params={'objective':'reg:linear',
        'booster':'gbtree',
        'eta':0.03,
        'max_depth':10,
        'subsample':0.9,
        'colsample_bytree':0.7,
        'silent':1,
        'seed':10}
num_boost_round= 6000
dtrain=xgb.DMatrix(X_train,y_train)
dvalid=xgb.DMatrix(X_test,y_test)
watchlist=[(dtrain,'train'),(dvalid,'eval')]

#模型训练
print('Train a XGBoost model')
start=time()
gbm=xgb.train(params,dtrain,num_boost_round,evals=watchlist,
              early_stopping_rounds=100,feval=rmspe_xg,verbose_eval=True)
end=time()
print('Train time is {:.2f} s.'.format(end-start))
# 数据集有点大，训练就花了7~8分钟。
gbm.save_model('./训练模型.json')
```

### 9.2.1、params参数说明：

- eta[默认是0.3] 和GBM中的learning rate参数类似。通过减少每一步的权重，可以提高模型的鲁棒性。典型值0.01-0.2
- max_depth [默认是3] 树的最大深度，这个值也是用来避免过拟合的3-10
- subsample[默认是1] 这个参数控制对于每棵树，随机采样的比例。减小这个参数的值算法会更加保守，避免过拟合。但是这个值设置的过小，它可能会导致欠拟合。典型值：0.5-1
- colsample_bytree[默认是1] 用来控制每颗树随机采样的列数的占比每一列是一个特征0.5-1（要依据特征个数来判断）
- objective[默认是reg:linear]这个参数定义需要被最小化的损失函数。最常用的值有：binary:logistic二分类的逻辑回归，返回预测的概率非类别。multi:softmax使用softmax的多分类器，返回预测的类别。在这种情况下，你还要多设置一个参数：num_class类别数目。

"reg:linear"：线性回归。
"reg:logistic"：逻辑回归。
"binary:logistic"：二分类的逻辑回归问题，输出为概率。
"binary:logitraw"：二分类的逻辑回归问题，输出的结果为wTx。
"count:poisson"：计数问题的poisson回归，输出结果为poisson分布。在poisson回归中，max_delta_step的缺省值为0.7。(used to safeguard optimization)
"multi:softmax"：让XGBoost采用softmax目标函数处理多分类问题，同时需要设置参数num_class（类别个数）
"multi:softprob"：和softmax一样，但是输出的是ndata * nclass的向量，可以将该向量reshape成ndata行nclass列的矩阵。没行数据表示样本所属于每个类别的概率。
"rank:pairwise"：set XGBoost to do ranking task by minimizing the pairwise loss

- seed[默认是0]随机数的种子，设置它可以复现随机数据的结果，也可以用于调整参数。
- booster[默认是gbtree ]选择每次迭代的模型，有两种选择：gbtree基于树的模型、gbliner线性模型
- silent[默认值=0]取0时表示打印出运行时信息，取1时表示以缄默方式运行，不打印运行时信息。

### 9.2.2、xgb.train()参数

- params：这是一个字典，里面包含着训练中的参数关键字和对应的值
- dtrain：训练的数据
- num_boost_round：这是指提升迭代的次数，也就是生成多少基模型
- evals：这是一个列表，用于对训练过程中进行评估列表中的元素。
- feval： 自定义评估函数
- early_stopping_rounds：早期停止次数 ，假设为100，验证集的误差迭代到一定程度在100次内不能再继续降低，就停止迭代。要求evals 里至少有 一个元素。
- verbose_eval (可以输入布尔型或数值型)：也要求evals 里至少有 一个元素。如果为True ,则对evals中元素的评估结果会输出在结果中；如果输入数字，假设为5，则每隔5个迭代输出一次
- learning_rates 每一次提升的学习率的列表
- xgb_model 在训练之前用于加载的xgb model

## 9.3、模型评估

```
# 采用保留数据集进行检测
print('validating')
X_test.sort_index(inplace=True)
y_test.sort_index(inplace=True)
yhat=gbm.predict(xgb.DMatrix(X_test))
error=rmspe(np.expm1(y_test),np.expm1(yhat))
print('RMSPE: {:.6f}'.format(error))

'''
validating
RMSPE: 0.129692
'''

# 构建保留数据集预测结果
res=pd.DataFrame(data=y_test)
res['Prediction']=yhat
res=pd.merge(X_test,res,left_index=True,right_index=True)
res['Ratio']=res['Prediction']/res['Sales']
res['Error']=abs(res['Ratio']-1)
```

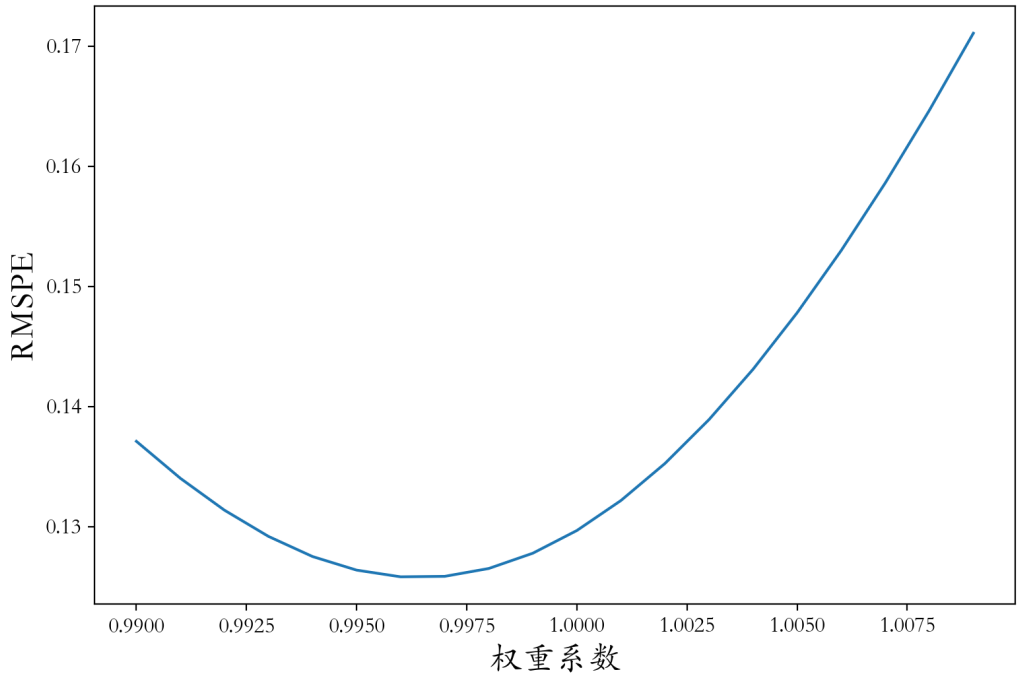
```
res['Weight']=res['Sales']/res['Prediction']
display(res.head())

# 分析保留数据集中任意三个店铺的预测结果
plt.rcParams['font.family'] = 'KaiTi SC'
col_1=['Sales','Prediction']
col_2=['Ratio']
L=np.random.randint(low=1,high=1115,size=3)
print('预测值和真实销量的比率 {}:全部商店'.format(res['Ratio'].mean()))
for i in L:
    s1=pd.DataFrame(res[res['Store']==i],columns=col_1)
    s2=pd.DataFrame(res[res['Store']==i],columns=col_2)
    s1.plot(title='预测数据和真实销量数据对比:商店 {}'.format(i),figsize=(12,4))
    s2.plot(title='预测值和真实销量的比率: 商店 {}'.format(i),figsize=(12,4))
    print('预测值和真实销量的比率 {}:商店 {}'.format(s2['Ratio'].mean(),i))
'''
预测值和真实销量的比率 1.0012098019730713:全部商店
预测值和真实销量的比率 1.004171751978279:商店 583
预测值和真实销量的比率 1.007007476542839:商店 567
预测值和真实销量的比率 1.0047990691924253:商店 589
'''

#分析偏差最大的10个预测结果
res.sort_values(['Error'],ascending=False,inplace=True)
res[:10]
```

从分析结果来看，初始模型已经可以比较好的预测保留数据集的销售趋势，但相对真实值，模型的预测值整体要偏高一些。从对偏差数据分析来看，偏差最大的3个数据也是明显偏高。因此，我们可以以保留数据集为标准对模型进行偏差校正。

## 9.4、模型优化

```
# 偏差整体校正优化
print('weight correction')
W=[(0.990+(i/1000)) for i in range(20)]
S=[]
for w in W:
    error=rmspe(np.expm1(y_test),np.expm1(yhat*w))
    print('RMSPE for {:.3f}:{:.6f}'.format(w,error))
    S.append(error)
Score=pd.Series(S,index=W)
plt.figure(figsize=(9,6))
Score.plot()
plt.xlabel('权重系数',fontsize = 18)
plt.ylabel('RMSPE',fontsize = 18)
best_score = Score[Score.values==Score.values.min()]
print('Best weight for Score:{}'.format(best_score))
'''
weight correction
RMSPE for 0.990:0.137119
RMSPE for 0.991:0.134037
RMSPE for 0.992:0.131384
RMSPE for 0.993:0.129201
RMSPE for 0.994:0.127526
RMSPE for 0.995:0.126396
RMSPE for 0.996:0.125839
RMSPE for 0.997:0.125879
```

```
RMSPE for 0.998:0.126531
RMSPE for 0.999:0.127803
RMSPE for 1.000:0.129692
RMSPE for 1.001:0.132188
RMSPE for 1.002:0.135274
RMSPE for 1.003:0.138926
RMSPE for 1.004:0.143117
RMSPE for 1.005:0.147817
RMSPE for 1.006:0.152995
RMSPE for 1.007:0.158619
RMSPE for 1.008:0.164658
RMSPE for 1.009:0.171083
Best weight for Score:0.996    0.125839
'''
```




当校正系数为0.996时，保留数据集的RMSPE得分最低：0.125839。相对于初始模型0.129692得分有很大的提升。
因为每个店铺都有自己的特点，而我们设计的模型对不同的店铺偏差并不完全相同，所以我们需要根据不同的店铺进行一个细致的校正。

## 9.5、全局优化

```
# 细致校正：以不同的店铺分组进行细致校正，每个店铺分别计算可以取得最佳RMSPE得分的校正系数
L=range(1115)
weights1 = [] # 验证数据的每个店铺的权重系数
weights2 = [] # 测试数据的每个店铺的权重系数
for i in L:
    s1 = pd.DataFrame(res[res['Store']==i+1],columns=col_1)
    s2 = pd.DataFrame(df_test[df_test['Store']==i+1])
    W1 = [(0.980+(i/1000)) for i in range(50)]
    S = []
    for w in W1:
        error = rmspe(np.expm1(s1['Sales']),np.expm1(s1['Prediction']*w))
        S.append(error)
```

```
    scores = pd.Series(S,index = W1)
    index = scores.argmin()
    best_weight = np.array(scores.index[index])
    weights1.extend(best_weight.repeat(len(s1)).tolist())
    weights2.extend(best_weight.repeat(len(s2)).tolist())

# 调整校正系数的排序
X_test = X_test.sort_values(by='Store')
X_test['weights1'] = weights1
X_test = X_test.sort_index()
weights1 = X_test['weights1'].values
X_test.drop(['weights1'],axis=1,inplace=True)

df_test = df_test.sort_values(by='Store')
df_test['weights2'] = weights2
df_test = df_test.sort_index()
weights2 = df_test['weights2'].values
df_test.drop(['weights2'],axis=1,inplace=True)

#计算校正后整体数据的RMSPE得分
yhat_new=yhat*weights1
error=rmspe(np.expm1(y_test),np.expm1(yhat_new))
print('RMSPE for weight corretion {:.6f}'.format(error))
```

RMSPE for weight corretion 0.118425，相对于整体校正的0.129692的得分又有不小的提高

## 9.6、模型预测

```
# 用初始和校正后的模型对训练数据集进行预测
print('对测试数据进行预测！')
test = xgb.DMatrix(df_test)
y_pred = gbm.predict(test)

# 初始模型
result=pd.DataFrame({'Id':np.arange(1,41089),'Sales':np.expm1(y_pred)})
result.to_csv('./Rossmann_submission1.csv',index=False)

# 整体校正模型
result=pd.DataFrame({'Id':np.arange(1,41089),'Sales':np.expm1(y_pred*0.996)})
result.to_csv('./Rossmann_submission2.csv',index=False)

# 细致校正模型
result=pd.DataFrame({'Id':np.arange(1,41089),'Sales':np.expm1(y_pred*weights2)})
result.to_csv('./Rossmann_submission3.csv',index=False)
```

上面构建的模型经过优化后，已经有着不错的表现。如果想继续提高预测的精度，可以在模型融合上试试。