时间序列分析

下文时我对简书的文章的自己进行在提取

基础知识

鲁棒性的介绍

鲁棒性是指某一个自动系统在系统发生故障时仍然能较好的完成预定工作的能力，我们来参考两个个机器人系统，我们希望机器人从A点像B点行进，如果中途机器人倒下，一号机器人只能停下，但二号机器人可以自行的爬起继续像B点前进，我们就认为一号机器人不具有鲁棒性，而二号有较好的鲁棒性。

它是在异常和危险情况下系统生存的关键，是指系统在一定(结构、大小)的参数摄动下，维持某些性能的特性。例如，计算机软件在输入错误、磁盘故障、网络过载或有意攻击情况下，能否不死机、不崩溃，就是该软件的鲁棒性。

时间序列的引入

支付数据、客流量数据、交通数据等，具有明显的周期性。从预测的角度说，周期性是核心，只要抓住了周期性，任务就完成了一大半。所以，我们需要学习时间序列来分析数据

首先，我们先学习基本规则法

基本规则法

预测的核心任务就是尽可能准确的提取到周期

	周一	周二	周三	周四	周五	周六	周日	周均值
第一周	20	10	70	50	250	200	100	100
第二周	26	18	66	50	180	140	80	80
第三周	15	8	67	60	270	160	120	100

先除以周均值，得到比例；**再**按列取中位数，这样就能得到一组带有鲁棒性的周期因子

如下

	周一	周二	周三	周四	周五	周六	周日
第一周	0.2	0.1	0.7	0.5	2.5	2	1
第二周	0.325	0.225	0.825	0.625	2.25	1.75	1
第三周	0.15	0.08	0.67	0.6	2.7	1.6	1.2
中位数	0.2	0.1	0.7	0.6	2.5	1.75	1

而预测下周的数据时，我们只需要将周期因子乘以一个base，就可以对下一周做预测

	周一	周二	周三	周四	周五	周六	周日
中位数	0.2	0.1	0.7	0.6	2.5	1.75	1
预测 (base=100)	20	10	70	60	250	175	100

可以见的，如果需要，可以将数据衍生更多，得到更准确的数据

对周期因子的优化

可以看出直接提取中位数来当作周期因子，十分简单而且页满足对周期因子的鲁棒性的要求。但是，并非所有的数据都能够适合提取中位数来做预测的，所以，在实际情况下，我们可以提取均值并和中位数融合形成周期因子（比例可参照测试集的表现来确定）。同样也可以根据与预测周的时间距离来赋予不同的权重。

针对base的优化

我们直接拿数据最后一周的平均值拿来做base（如此做的目的因为预测数据与最近的数据最相关），但这个方法也不是最好的，许最后三天或最后五天的均值能更好的反映最新的情况。但是，我们不能直接对最后三天客流量取均值（最后三天是周末，这样取的base就偏大了）。需要去掉周期性因素后，再取平均。具体做法，就是用客流量除以周期因子。

	周一	周二	周三	周四	周五	周六	周日
第三周	15	8	67	60	270	160	120
中位数	0.2	0.1	0.7	0.6	2.5	1.75	1
去周期以后的客流量	75	80	95.7	100	108	91.4	120

这样我们就可以取最后三天的平均，(108+91.4+120)/3=106.5，作为base。具体取多少天的，也要通过测试集的表现来确定。当然也可以按某些函数形式来给每天赋予不同的权重。

其他影响因素

在IJCAI-17的这个赛题里面，天气是非常重要的一个影响因素。可以提取残差，然后用残差训练一个天气的模型。推荐使用xgboost。其他影响因素可以如法炮制。

题目：（在口碑平台上，我们将客户流量定义为“单位时间内在商家使用支付宝消费的用户人次”。在这个问题中，我们将提供用户的浏览和支付历史，以及商家相关信息，并希望参赛选手可以以此预测所有商家在接下来14天内，每天的客户流量。我们鼓励参赛选手使用类似天气等额外的数据，并希望参赛选手能够将数据源共享在论坛中。）

代码理解

```
import pandas as pd
import numpy as np
import datetime
# Load the balance data
def load_data(path: str = '') -> pd.DataFrame:
    data_balance = pd.read_csv(path)
    data_balance = add_timestamp(data_balance)
    return data_balance.reset_index(drop=True)
# add tiemstamp to dataset
def add_timestamp(data: pd.DataFrame, time_index: str = 'report_date') ->
pd.DataFrame:
    data_balance = data.copy()
```

```
    data_balance['date'] = pd.to_datetime(data_balance[time_index],
format="%Y%m%d")
    data_balance['day'] = data_balance['date'].dt.day
    data_balance['month'] = data_balance['date'].dt.month
    data_balance['year'] = data_balance['date'].dt.year
    data_balance['week'] = data_balance['date'].dt.isocalendar().week
    data_balance['weekday'] = data_balance['date'].dt.weekday
    return data_balance.reset_index(drop=True)
# total amount
def get_total_balance(data: pd.DataFrame, date: str = '2014-03-31') ->
pd.DataFrame:
    df_tmp = data.copy()
    df_tmp = df_tmp.groupby(['date'])[['total_purchase_amt',
'total_redeem_amt']].sum()
    df_tmp.reset_index(inplace=True)  # 删除数据后使用reset_index()重置索引
    return df_tmp[(df_tmp['date'] >= date)].reset_index(drop=True)
# Generate the test data
def generate_test_data(data: pd.DataFrame) -> pd.DataFrame:
    total_balance = data.copy()
    start = datetime.datetime(2014, 9, 1)
    testdata = []
    while start != datetime.datetime(2014, 10, 15):
        temp = [start, np.nan, np.nan]
        testdata.append(temp)
        start += datetime.timedelta(days=1)
    testdata = pd.DataFrame(testdata)
    testdata.columns = total_balance.columns
    total_balance = pd.concat([total_balance, testdata], axis=0)  # 数据的拼接
    total_balance = total_balance.reset_index(drop=True)  # reset_index()重置索引
    return total_balance.reset_index(drop=True)
```

首先先导入数据，然后对时间数据进行预处理（加入时间戳），得到经过比对后的数据，生成测试集

```
# 定义生成时间序列规则预测结果的方法
def generate_base(df: pd.DataFrame, month_index: int) -> pd.DataFrame:
    # 选中固定时间段的数据集
    total_balance = df.copy()
    total_balance = total_balance[['date', 'total_purchase_amt',
'total_redeem_amt']]
    total_balance = total_balance[(total_balance['date'].dt.date >=
datetime.date(2014, 3, 1)) & (
                    total_balance['date'].dt.date < datetime.date(2014,
month_index, 1))]
    # 加入时间戳
    total_balance['weekday'] = total_balance['date'].dt.weekday
    total_balance['day'] = total_balance['date'].dt.day
    total_balance['week'] = total_balance['date'].dt.isocalendar().week
    total_balance['month'] = total_balance['date'].dt.month
    # 统计翌日因子
    mean_of_each_weekday = total_balance[['weekday'] + ['total_purchase_amt',
'total_redeem_amt']].groupby('weekday', as_index=False).mean()
    for name in ['total_purchase_amt', 'total_redeem_amt']:
        mean_of_each_weekday = mean_of_each_weekday.rename(columns={name: name +
'_weekdaymean'})
    mean_of_each_weekday['total_purchase_amt_weekdaymean'] /=
np.mean(total_balance['total_purchase_amt'])
```

```
    mean_of_each_weekday['total_redeem_amt_weekdaymean'] /=
np.mean(total_balance['total_redeem_amt'])
    # 合并统计结果到原数据集
    total_balance = pd.merge(total_balance, mean_of_each_weekday, on='weekday',
how='left')
    # 分别统计翌日在(1~31)号出现的频次
    weekday_count = total_balance[['day', 'weekday', 'date']].groupby(['day',
'weekday'], as_index=False).count()
    weekday_count = pd.merge(weekday_count, mean_of_each_weekday, on='weekday')
    # 依据频次对翌日因子进行加权，获得日期因子
    weekday_count['total_purchase_amt_weekdaymean'] *= weekday_count['date'] /
len(np.unique(total_balance['month']))
    weekday_count['total_redeem_amt_weekdaymean'] *= weekday_count['date'] /
len(np.unique(total_balance['month']))
    day_rate = weekday_count.drop(['weekday', 'date'], axis=1).groupby('day',
as_index=False).sum()
    # 将训练集中所有日期的均值剔除日期残差得到base
    day_mean = total_balance[['day'] + ['total_purchase_amt',
'total_redeem_amt']].groupby('day', as_index=False).mean()
    day_pre = pd.merge(day_mean, day_rate, on='day', how='left')
    day_pre['total_purchase_amt'] /= day_pre['total_purchase_amt_weekdaymean']
    day_pre['total_redeem_amt'] /= day_pre['total_redeem_amt_weekdaymean']
    # 生成测试集数据
    for index, row in day_pre.iterrows():
        if month_index in (2, 4, 6, 9) and row['day'] == 31:
            break
        day_pre.loc[index, 'date'] = datetime.datetime(2014, month_index,
int(row['day']))
    # 基于base与翌日因子获得最后的预测结果
    day_pre['weekday'] = day_pre.date.dt.weekday
    day_pre = day_pre[['date', 'weekday'] + ['total_purchase_amt',
'total_redeem_amt']]
    day_pre = pd.merge(day_pre, mean_of_each_weekday, on='weekday')
    day_pre['total_purchase_amt'] *= day_pre['total_purchase_amt_weekdaymean']
    day_pre['total_redeem_amt'] *= day_pre['total_redeem_amt_weekdaymean']
    day_pre = day_pre.sort_values('date')[['date'] + ['total_purchase_amt',
'total_redeem_amt']]
    return day_pre
```