Qbus6840 TUT2

添加噪声

为了避免因缺少generalization而导致关于时间的预测结果误差变大，需要在采集到的数据上添加一定的噪声作为regularization。

numpy.random 随机数库

`numpy.random.seed( int )`

设置一个数作为随机数的种子。

一般计算机的随机数都是伪随机数，以一个真随机数（种子）作为初始条件，然后用一定的算法不停迭代产生随机数。每一个随机种子中对应一个随机数序列，相同的随机种子产生固定的随机数结果，不同的随机种子产生不同的随机数结果，如果不设置seed，则每次会生成不同的随机数

`numpy.random.rand(d0,d1,...,dn)`

- rand 函数根据给定维度生成[0,1)之间的随机数 array，随机数包含0，不包含1
- dn 指矩阵的size

```
np.random.rand(n)
# 当没有参数时，返回单个随机值
```

```
np.random.rand(4,2)
```

```
array([[0.08881751, 0.76386783],
       [0.90482473, 0.28623335],
       [0.85578948, 0.91181243],
       [0.54528711, 0.20863432]])
```

`numpy.random.randn(d0,d1,...,dn)`

- randn 函数根据指定维度生成[0,1)的随机数 array，且**满足标准正态分布**(standard normal distribution)。
- dn 指矩阵的 size。

```
np.random.randn()
# 当没有参数时，返回单个随机值
```

```
np.random.randn(4,3,2)
```

```
array([[[-2.02067788, -0.17047864],
```

```
         [ 0.2431011 ,  2.04147134],
         [-1.03459526,  1.70765775]],

        [[ 0.16530504, -2.31204068],
         [-0.92086358,  0.48077415],
         [-0.2083771 , -1.65900628]],

        [[ 0.48263022, -0.18496153],
         [ 0.86799187, -1.13014091],
         [ 0.20610472,  0.92884775]],

        [[ 0.97521267,  0.46242446],
         [ 0.99368696,  1.10540577],
         [-0.42352212,  1.28309725]]])
```

`numpy.random.randint(low, high=None, size=None,dtype='l')`

- randint 函数返回随机整数，范围区间为[low,high);
- low为最小值，high为最大值;
- size为数组维度大小;
- dtype为数据类型，默认的数据类型是np.int;
- high没有填写时，默认生成随机数的范围是[0，low)

```
np.random.randint(1,5) # 返回1个[1,5)时间的随机整数
```

```
3
```

```
np.random.randint(-2,10,size=(2,3))
```

```
array([[ 9,  7, -1],
       [ 2, -2,  5]])
```

```
np.random.randint(-2,size=(2,3))
```

```
ValueError: low >= high
```

`numpy.random.choice(a, size=None, replace=True, p=None)`

- choice 函数从给定的一维数组中生成随机数
- a 为一维数组类似数据或整数；
- size 为数组维度；
- replace 生成的随机数能否有重复的数
- p 为数组中的数据出现的概率
- a 为整数时，对应的一维数组为 np.arange(a)

```
In [11]: np.random.choice(5)
Out[11]: 4
```

```
In [14]: np.random.choice(5,3)
Out[14]: array([1, 3, 3])
```

```
In [17]: np.random.choice(5,3,replace=False)
Out[17]: array([2, 0, 3])
```

```
In [15]: np.random.choice(5,2)
Out[15]: array([2, 0])
```

```
In [16]: np.random.choice(5,size=(3,2))
Out[16]: array([[2, 1],
       [0, 3],
       [1, 2]])
```

```
demo_list = ['lenovo', 'sansumg','moto','xiaomi', 'iphone']
In [22]: np.random.choice(demo_list,size=(3,3))
Out[22]:
array([['sansumg', 'moto', 'iphone'],
       ['xiaomi', 'iphone', 'lenovo'],
       ['sansumg', 'lenovo', 'iphone']], dtype='<U7')
```

- 参数p的长度与参数a的长度需要一致；
- 参数p为数据出现的概率，p里的数据之和应为1

```
n [23]: np.random.choice(demo_list,size=(3,3), p=[0.1,0.6,0.1,0.1,0.1])
Out[23]:
array([['moto', 'sansumg', 'sansumg'],
       ['sansumg', 'sansumg', 'moto'],
       ['xiaomi', 'sansumg', 'sansumg']], dtype='<U7')
```

Time-shift

`pandas.series.shift(periods=1, freq=None, axis=0, fill_value=None)`

shift 函数能将数据进行移动

- period : 类型为int，表示移动的幅度，可以是正数，也可以是负数，默认值是1，1表示数据移动一格。注意这里移动的都是数据，而索引是不移动的。
- freq : 只用于时间序列（TS），也就是 index 是时间格式。如果这个参数存在，那么数据会按照参数值移动时间索引，而数据没有发生变化
- axis : 数据移动方向。默认是0，上下移动，如果赋值为1，左右移动
- fill_value : 数据移动之后的填充值，移动之后如果没有填充值，数据赋值为NaN。(pandas 0.24.1

版本有这个参数,但 anaconda 目前 pandas 处于0.23.0,没有该参数)

```
df = pd.DataFrame({ 'Col1': [10, 20, 15, 30, 45],
                    'Col2': [13, 23, 18, 33, 48],
                    'Col3': [17, 27, 22, 37, 52]})
```

```
df.shift(periods=3)
    Col1    Col2    Col3
0   NaN NaN NaN
1   NaN NaN NaN
2   NaN NaN NaN
3   10.0    13.0    17.0
4   20.0    23.0    27.0
```

```
df.shift(periods=-1)
       Col1   Col2   Col3
0      20.0   23.0   27.0
0      15.0   18.0   22.0
2      30.0   33.0   37.0
3      45.0   48.0   52.0
4      NaN    NaN    NaN
```

```
df.shift(periods=1, axis=1)
    Col1    Col2    Col3
0   NaN 10.0    13.0
1   NaN 20.0    23.0
2   NaN 15.0    18.0
3   NaN 30.0    33.0
4   NaN 45.0    48.0
```

```
df.shift(periods=3, fill_value=0)
   Col1  Col2  Col3
0     0     0     0
1     0     0     0
2     0     0     0
3    10    13    17
4    20    23    27
```

```
df = pd.DataFrame(np.arange(16).reshape(4,4),columns=
['AA','BB','CC','DD'],index =pd.date_range('6/1/2012','6/4/2012'))

df

              AA    BB  CC  DD
2012-06-01  0   1   2   3
2012-06-02  4   5   6   7
2012-06-03  8   9   10  11
2012-06-04  12  13  14  15
```

```
import datetime
df.shift(freq=datetime.timedelta(1))
df

           AA   BB  CC  DD
2012-06-02  0   1   2   3
2012-06-03  4   5   6   7
2012-06-04  8   9   10  11
2012-06-05  12  13  14  15
```

```
df.shift(freq=datetime.timedelta(-2))
df

AA  BB  CC  DD
2012-05-30  0   1   2   3
2012-05-31  4   5   6   7
2012-06-01  8   9   10  11
2012-06-02  12  13  14  15
```

`pandas.series.tshift(periods=1)`

tshift 函数实现的功能和 shift 类似，不同的是，tshift 函数只用于 Time series，因为只改变 index 中的时间数据，不会导致数据 NaN 出现

- periods : 表示移动的幅度，可以是正数，也可以是负数，默认值是1，1表示数据移动一格

```
df.tshift(2)
           AA   BB  CC  DD
2012-06-03  0   1   2   3
2012-06-04  4   5   6   7
2012-06-05  8   9   10  11
2012-06-06  12  13  14  15
```

`matplotlib.pyplot.subplot(nrows=1, ncols=1, sharex=False, sharey=False)`

subplot 函数生成含多张子图的图表

- nrows, ncols : 生成的子表矩阵的行数和列数
- sharex,sharey : 是否共享x轴/y轴坐标,默认是 False

  返回
- fig : 图表（整张图 Figure） 对象
- ax : 子图（axes.Axes）对象， 个数根据 nrows 和 ncols 确定

```
x = np.linspace(0, 2*np.pi, 400)
y = np.sin(x**2)
```

```
# 只有一个子图
fig, ax = plt.subplots()
ax.plot(x, y)
ax.set_title('Simple plot')
```

```
# 有两个子图，并且共享 Y 轴坐标
f, (ax1, ax2) = plt.subplots(1, 2, sharey=True)
ax1.plot(x, y)
ax1.set_title('Sharing Y axis')
ax2.scatter(x, y)
```

`plt.legend( labels, loc)`

legend 函数在图表上添加图示

- labels 图示的内容， string类型
- loc 图示的位置

| Location String | Location Code | |---|---| 'best'|0 'upper right'|1 'upper left'|2 'lower left'|3 'lower right'|4 'right'|5 'center left'|6 'center right'|7 'lower center'|8 'upper center'|9 'center'|10

`plt.axvline(x=0, ymin=0, ymax=1)`

axvline函数 在子图表上添加垂直辅助线

- x : 垂直线在 x 轴的坐标
- ymin,ymax : 垂直线的长度，在 0 到 1之间，0是图的底部，1是图的顶部。

`plt.axhline(y=0, xmin=0, xmax=1)`

axhline 函数 在子图表上添加水平辅助线

- y : 水平线在 y 轴的坐标
- ymin,ymax : 水平线的长度，在 0 到 1之间，0是图的左部，1是图的右部。

数据提取

这部分的内容基本都是 Buss6002 Tutorial 第3、4周的知识，上过课的同学可以拿原来的课件复习下

`dataframe.info()`

info 函数 提供 DF 的一些基本信息

`dataframe.describe()`

describe 函数得到 dataframe 每一列的几个基本的统计值

```
s = pd.Series([1, 2, 3])
s.describe()
count    3.0
mean     2.0
std      1.0
min      1.0
25%      1.5
50%      2.0
75%      2.5
max      3.0
dtype: float64
```

数据查询

`Dataframe` 有很多筛选数据的功能，数据分析往往都是从对原始数据筛选这一步开始

筛选返回的仍然是`dataframe`类型

索引查询

- 我们想要筛选 `directmarketing`里`'AmountSpent'`列数据中大于0的行

  ```
  drinks = pd,read_csv("drinks.csv")
  ```

  ```
  euro_frame = drinks[drinks['continent'] == 'EU']
  ```

- 因为查询结果返回的是一个 dataframe 所以想要筛选 `directmarketing`里`'AmountSpent'`列数据中大于0的前10行，可以在之后加上索引，选择前十行数据

  ```
  drinks[drinks['continent'] == 'EU'][:10]
  ```

  注意条件判断里面等号是 == 不是一个等号

- 判断条件里也可以包括逻辑运算符（& | not ）

  想要筛选同时是欧洲国家,且年度服务量超过300的国家数据组成的 dataframe

  ```
  euro_wine_300_frame = drinks[(drinks['continent'] =='EU') &
  (drinks['wine_servings'] > 300)]
  ```

- 只需要A和B列数据，而D和C列数据都是用于筛选的

  想要筛选`'Age'`和`Gender`列数据，筛选条件是

`'AmountSpent'`列数据中大于1000 `'OwnHome'`是'Own'

```
  marketing[['Age','Gender']][ (marketing['AmountSpent' ] > 1000) &
(marketing['OwnHome'] == 'Own')]
```

条件查询

- 查询函数`dataframe.query('筛选条件')`

```
big_earners = marketing.query("Salary > 90000")
```

`dataFrame.sort_values(by = ['列一','列二',...], axis = 0, ascending = Ture, inplace=False)`

sort_values按值排序

- `by` 决定要依据哪一列（行）排序，

  如果是某一列 `by = '列名'`

  如果是很多列`by = ['列一','列二',...]`
- `axis` 决定是上下排序还是左右排序，默认为上下排序

  axis = 0 按 index 排序，上下排序 axis = 1 按 columns 排序, 左右排序
- `ascending` 决定是升序还是降序，默认是升序

  ascending = True 升序

  ascending = False 降序
- `inplace` 决定是否替代原数据， 默认为否

`dataFrame.fillna(value , inplace = False)`

fillna函数能填充将 NA 和 NaN（丢失的数据）

- value 用来填充缺失值的值
- inplace 是否修改当前的表格

```
drinks['continent'].fillna(value='NA', inplace=True)
```