Pandas（续） C06

胡俊峰 2020/03/23
北京大学信息科学技术学院

主要内容

- Pandas索引对象与高维索引
- 推荐算法与搜索排名
- Python交互界面与Tkinter软件包（郭一诺助教）

The Pandas Index Object

- 不可修改的数组
- 有序
- 支持可重复 key

Index as ordered set（支持表关联计算的基础）

```
indA = pd.Index([1, 3, 5, 7, 9])
indB = pd.Index([2, 3, 5, 7, 11])
```

```
indA & indB  # intersection
```

```
Int64Index([3, 5, 7], dtype='int64')
```

```
indA | indB  # union
```

```
Int64Index([1, 2, 3, 5, 7, 9, 11], dtype='int64')
```

```
indA ^ indB  # symmetric difference
```

```
Int64Index([1, 2, 9, 11], dtype='int64')
```

Indexers: loc, iloc, and ix 按位置索引

These slicing and indexing conventions can be a source of confusion. For example, if your `Series` has an explicit integer index, an indexing operation such as `data[1]` will use the explicit indices, while a slicing operation like `data[1:3]` will use the implicit Python-style index.

```
data = pd.Series(['a', 'b', 'c'], index=[1, 3, 5])
data
```

```
1    a
3    b
5    c
dtype: object
```

```
# explicit index when indexing
data[1]
```

```
'a'
```

```
print(data.loc[1])
print(data.iloc[1])
```

```
a
b
```

```
# implicit index when slicing
data[1:3]
```

```
3    b
5    c
dtype: object
```

Data Selection in DataFrame

```
area = pd.Series({'California': 423967, 'Texas': 695662,
                  'New York': 141297, 'Florida': 170312,
                  'Illinois': 149995})
pop = pd.Series({'California': 38332521, 'Texas': 26448193,
                 'New York': 19651127, 'Florida': 19552860,
                 'Illinois': 12882135})
data = pd.DataFrame({'area':area, 'pop':pop})
data
```

	area	pop
California	423967	38332521
Florida	170312	19552860
Illinois	149995	12882135
New York	141297	19651127
Texas	695662	26448193

```
data['area']
```

```
California    423967
Florida       170312
Illinois      149995
New York      141297
Texas         695662
Name: area, dtype: int64
```

用法：字段名 类比 属性

Equivalently, we can use attribute-style access with column names that are strings:

```
data.area
```

```
California    423967
Florida       170312
Illinois      149995
New York      141297
Texas         695662
Name: area, dtype: int64
```

```python
data['density'] = data['pop'] / data['area']
data
```

	area	pop	density
California	423967	38332521	90.413926
Florida	170312	19552860	114.806121
Illinois	149995	12882135	85.883763
New York	141297	19651127	139.076746
Texas	695662	26448193	38.018740

行列互换

```
s = data.T
print(s)
s['California']
```

```
         California         Texas      New York       Florida      Illinois
area     4.239670e+05  6.956620e+05  1.412970e+05  1.703120e+05  1.499950e+05
pop      3.833252e+07  2.644819e+07  1.965113e+07  1.955286e+07  1.288214e+07
density  9.041393e+01  3.801874e+01  1.390767e+02  1.148061e+02  8.588376e+01
```

```
area       4.239670e+05
pop        3.833252e+07
density    9.041393e+01
Name: California, dtype: float64
```

筛选，赋值：

```
data.loc[data.density > 100, ['pop', 'density']]
```

	pop	density
Florida	19552860	114.806121
New York	19651127	139.076746

Any of these indexing conventions may also be used to set or modify values; this is done in the standard way that you might be accustomed to from working with NumPy:

```
data.iloc[0, 2] = 90
data
```

	area	pop	density
California	423967	38332521	90.000000
Texas	695662	26448193	38.018740
New York	141297	19651127	139.076746
Florida	170312	19552860	114.806121
Illinois	149995	12882135	85.883763

Working with NumPy ufunc

```
df = pd.DataFrame(rng.randint(0, 10, (3,4)),
                  columns=['A', 'B', 'C', 'D'])
df
```

	A	B	C	D
0	9	2	6	7
1	4	3	7	7
2	2	5	4	1

```
np.sin(df * np.pi / 4)
```

采用Numpy的Ufunc —— 广播机制

	A	B	C	D
0	7.071068e-01	1.000000	-1.000000e+00	-0.707107
1	1.224647e-16	0.707107	-7.071068e-01	-0.707107
2	1.000000e+00	-0.707107	1.224647e-16	0.707107

Dataframe之间的运算自动进行索引对齐-补足（out join）

Out[22]:

	A	B
0	2	4
1	18	6

In [23]:

```
B = pd.DataFrame(rng.randint(0, 10, (3, 3)),
                 columns=list('BAC'))
B
```

Out[23]:

	B	A	C
0	4	8	6
1	1	3	8
2	1	9	8

In [24]:

```
A + B
```

Out[24]:

	A	B	C
0	10.0	8.0	NaN
1	21.0	7.0	NaN
2	NaN	NaN	NaN

The following table lists Python operators and their equivalent Pandas object methods:

Python Operator	Pandas Method(s)
`+`	`add()`
`-`	`sub()`, `subtract()`
`*`	`mul()`, `multiply()`
`/`	`truediv()`, `div()`, `divide()`
`//`	`floordiv()`
`%`	`mod()`
`**`	`pow()`

Frame 与 series 计算，按行broadcasting

```
A = rng.randint(10, size=(3, 4))
A
```

```
array([[9, 4, 1, 3],
       [6, 7, 2, 0],
       [3, 1, 7, 3]])
```

```
df = pd.DataFrame(A, columns=list('QRST'))
df - df.iloc[0]
```

	Q	R	S	T
0	0	0	0	0
1	-3	3	1	-3
2	-6	-3	6	0

```
df.subtract(df['R'], axis=0)
```

	Q	R	S	T
0	5	0	-3	-1
1	-1	0	-5	-7
2	2	0	6	2

运算过程中类型自适应转换

The following table lists the upcasting conventions in Pandas when NA values are introduced:

Typeclass	Conversion When Storing NAs	NA Sentinel Value
`floating`	No change	`np.nan`
`object`	No change	`None` or `np.nan`
`integer`	Cast to `float64`	`np.nan`
`boolean`	Cast to `object`	`None` or `np.nan`

Keep in mind that in Pandas, string data is always stored with an `object` dtype.

Detecting null values

Pandas data structures have two useful methods for detecting null data: `isnull()` and `notnull()`. Either one will return a Boolean mask over the data. For example:

```
data = pd.Series([1, np.nan, 'hello', None])
```

```
data.isnull()
```

```
0    False
1     True
2    False
3     True
dtype: bool
```

As mentioned in Data Indexing and Selection, Boolean masks can be used directly as a `Series` or `DataFrame` index:

```
data[data.notnull()]
```

```
0        1
2    hello
dtype: object
```

We can fill NA entries with a single value, such as zero:

```
data.fillna(0)
```

```
a    1.0
b    0.0
c    2.0
d    0.0
e    3.0
dtype: float64
```

We can specify a forward-fill to propagate the previous value forward:

```
# forward-fill
data.fillna(method='ffill')
```

```
a    1.0
b    1.0
c    2.0
d    2.0
e    3.0
dtype: float64
```

层次-组合 索引（ Hierarchical-Indexing ）

```python
index = [('California', 2000), ('California', 2010),
         ('New York', 2000), ('New York', 2010),
         ('Texas', 2000), ('Texas', 2010)]
populations = [33871648, 37253956,
               18976457, 19378102,
               20851820, 25145561]
pop = pd.Series(populations, index=index)
pop
```

```
(California, 2000)    33871648
(California, 2010)    37253956
(New York, 2000)      18976457
(New York, 2010)      19378102
(Texas, 2000)         20851820
(Texas, 2010)         25145561
dtype: int64
```

类似二维表切片

```python
pop[:, 2010]
```

```
California    37253956
New York      19378102
Texas         25145561
dtype: int64
```

MultiIndex VS extra dimension

```python
#unstack() method will quickly convert a multiply indexed Series
#into a conventionally indexed DataFrame:
pop_df = pop.unstack()
pop_df
```

	2000	2010
California	33871648	37253956
New York	18976457	19378102
Texas	20851820	25145561

```python
#unstack() method will quickly convert a multiply indexed Series into a conventi
pop_df.stack()
```

```
California  2000    33871648
            2010    37253956
New York    2000    18976457
            2010    19378102
Texas       2000    20851820
            2010    25145561
dtype: int64
```

```python
pop_df = pd.DataFrame({'total': pop,
                       'under18': [9267089, 9284094,
                                   4687374, 4318033,
                                   5906301, 6879014]})
pop_df
```

		total	under18
California	2000	33871648	9267089
	2010	37253956	9284094
New York	2000	18976457	4687374
	2010	19378102	4318033
Texas	2000	20851820	5906301
	2010	25145561	6879014

```python
f_u18 = pop_df['under18'] / pop_df['total']
f_u18.unstack()
```

	2000	2010
California	0.273594	0.249211
New York	0.247010	0.222831
Texas	0.283251	0.273568

Methods of MultiIndex Creation

The most straightforward way to construct a multiply indexed `Series` or `DataFrame` is to simply pass a list of two or more index arrays to the constructor. For example:

```
df = pd.DataFrame(np.random.rand(4, 2),
                  index=[['a', 'a', 'b', 'b'], [1, 2, 1, 2]],
                  columns=['data1', 'data2'])
df
```

		data1	data2
a	1	0.554233	0.356072
	2	0.925244	0.219474
b	1	0.441759	0.610054
	2	0.171495	0.886688

Similarly, if you pass a dictionary with appropriate tuples as keys, Pandas will automatically recognize this and use a `MultiIndex` by default:

```
data = {('California', 2000): 33871648,
        ('California', 2010): 37253956,
        ('Texas', 2000): 20851820,
        ('Texas', 2010): 25145561,
        ('New York', 2000): 18976457,
        ('New York', 2010): 19378102}
pd.Series(data)
```

```
California  2000    33871648
            2010    37253956
New York    2000    18976457
            2010    19378102
Texas       2000    20851820
            2010    25145561
dtype: int64
```

MultiIndex constructor

```
1  pd.MultiIndex.from_arrays([['a', 'a', 'b', 'b'], [1, 2, 1, 2]])
```

```
MultiIndex(levels=[['a', 'b'], [1, 2]],
           labels=[[0, 0, 1, 1], [0, 1, 0, 1]])
```

You can construct it from a list of tuples giving the multiple index values of each point:

```
1  pd.MultiIndex.from_tuples([('a', 1), ('a', 2), ('b', 1), ('b', 2)])
```

```
MultiIndex(levels=[['a', 'b'], [1, 2]],
           labels=[[0, 0, 1, 1], [0, 1, 0, 1]])
```

You can even construct it from a Cartesian product of single indices:

```
1  pd.MultiIndex.from_product([['a', 'b'], [1, 2]])
```

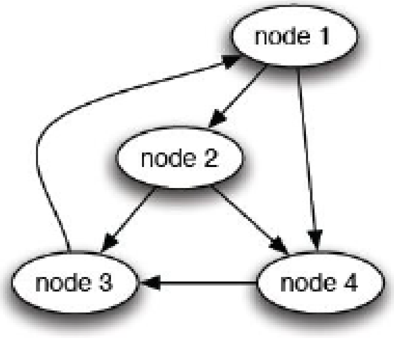
```
MultiIndex(levels=[['a', 'b'], [1, 2]],
           labels=[[0, 0, 1, 1], [0, 1, 0, 1]])
```

推荐算法与搜索排名

- PCA降维与特征空间
- HITS算法与搜索排名
- SVD与隐含语义挖掘
- 协同过滤算法

HITS算法(Hyperlink-induced Topic Search)

- 首先，通过关键词在文档中是否出现，得到相关网页集合及其链接关系。
- 每个网页有两个值：a(权威值，入度高）及h（hub，中枢值，出度高）
- 通过反复迭代计算，得出得分最高的网页。



$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 6 \\ 4 \\ 3 \end{bmatrix} = \begin{bmatrix} 9 \\ 7 \\ 2 \\ 4 \end{bmatrix}$$

定义n个网页之间链接关系的邻接矩阵为M,即M_{ij}为1$\Longleftrightarrow$从网页i到j有链接，则网页i的中枢值为：

$$a_i \longleftarrow M_{1i}h_1 + M_{2i}h_2 + \dots M_{ni}h_n \tag{3}$$

$$h_i \longleftarrow M_{i1}a_1 + M_{i2}a_2 + \dots M_{in}a_n \tag{4}$$

h^k=$(h_1, h_2, \dots h_n)^T$,a^k=$(a_1, a_2, \dots a_n)^T$为运行了k次的时候，n个网页的中枢，权威向量，则转换规则为：（先更新a^k)

$$a^k = M^T h^{k-1} \tag{5}$$

$$h^k = M a^k \tag{6}$$

h^0, a^0的元素都为1 迭代可得：$a^1 = M^T h^0, h^1 = M M^T h^0, \ldots$

$$a^k = (M^T M)^{k-1} h^0 \tag{7}$$

$$h^k = (M M^T)^k h^0 \tag{8}$$

定理：n*n的实对称矩阵有n个特征值，且不同特征值的特征向量彼此正交（即特征向量构成线性空间的一组基底）

设$(M^TM)^{k-1}$的特征值为$c_1, c_2, \ldots c_n$，且$c_1 > c_2 > , \ldots > c_n$，对应的特征向量为$z_1, z_2, \ldots z_n$，而$h^0$在基底下的表示为

$$h_0 = q_1 z_1 + q_2 z_2 + \ldots + q_n z_n \tag{9}$$

则

$$\begin{aligned} h^k &= (MM^T)^k h^0 & (10)\\ &= (MM^T)^k (q_1 z_1 + q_2 z_2 + \ldots q_n z_n) & (11)\\ &= q_1 (MM^T)^k z_1 + q_2 (MM^T)^k z_2 + \ldots + q_n (MM^T)^k z_n & (12)\\ &= q_1 c_1^k z_1 + q_2 c_2^k z_2 + \ldots q_n c_n^k z_n & (13) \end{aligned}$$

如果要收敛，需要对每项正规化。则

$$
\begin{aligned}
h^k &= = \frac{(MM^T)^k h^0}{\|(MM^T)^k h^0\|} && (14)\\
&= \frac{q_1 c_1^k z_1 + q_2 c_2^k z_2 + \ldots q_n c_n^k z_n}{\|q_1 c_1^k z_1 + q_2 c_2^k z_2 + \ldots q_n c_n^k z_n\|} && (15)\\
&= \frac{q_1 z_1 + q_2 (\frac{c_2}{c_1})^k z_2 + \ldots q_n (\frac{c_2}{c_1})^k z_n}{\|q_1 z_1 + q_2 (\frac{c_2}{c_1})^k z_2 + \ldots q_n (\frac{c_2}{c_1})^k z_n\|} && (16)\\
&= \frac{q_1 z_1}{\|q_1 z_1\|} && (17)
\end{aligned}
$$

故h_k收敛，同理可得a_k收敛

奇异值分解

- $N \times M$矩阵X，把每一行当成一个点。设$r = rank(X)$。
- $\vec{v}_1 = argmax_{|\vec{v}|=1}|X\vec{v}|$
- $\vec{v}_2 = argmax_{|\vec{v}|=1,\vec{v}\perp\vec{v}_1}|X\vec{v}|$
- ……
- $\vec{v}_r = argmax_{|\vec{v}|=1,\vec{v}\perp\vec{v}_1,\vec{v}\perp\vec{v}_2\ldots,,\vec{v}\perp\vec{v}_{r-1}}|X\vec{v}|$

- 令$\sigma_i = |X\vec{v}_i|, \overrightarrow{u_i} = \frac{1}{\sigma_i}X\vec{v}_i, 1 \le i \le r$
- 则$X = \sum_{i=1}^{r} \sigma_i \vec{u}_i \vec{v}_i^T = \begin{bmatrix} | & & | \\ \vec{u}_1 & \cdots & \vec{u}_r \\ | & & | \end{bmatrix} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix} \begin{bmatrix} -\vec{v}_1^T - \\ \cdots \\ -\vec{v}_r^T \end{bmatrix} = UDV^T$

奇异值分解

- $N \times M$矩阵X，把每一行当成一个点。设$r = rank(X)$。
- $X = \sum_{i=1}^{r} \sigma_i \vec{u}_i \vec{v}_i^T = \begin{bmatrix} | & & | \\ \vec{u}_1 & \dots & \vec{u}_r \\ | & & | \end{bmatrix} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix} \begin{bmatrix} -\vec{v}_1^T - \\ \dots \\ -\vec{v}_r^T - \end{bmatrix} = UDV^T$
- U, D, V分别为$N \times r, r \times r, M \times r$矩阵
- $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r$称为奇异值

奇异值分解与主成分分析

- $X = UDV^T$
- 考虑X的变换$Y = XV = UD$(维度重组)

- 则$C_Y = \frac{1}{N}Y^TY = \frac{1}{N} = \frac{1}{N}D^TU^TUD = \frac{1}{N}D^2$是个对角矩阵
- 各个维度之间不相关
- 对角元的大小——这一维自身的方差
- 按方差大小排列，得到第1,2,…, r个主成分
- 方差小的几个主成分可以认为是噪声，将其丢弃——降维

降维的意义

- 减少数据冗余度
- 一定程度上减少了噪声

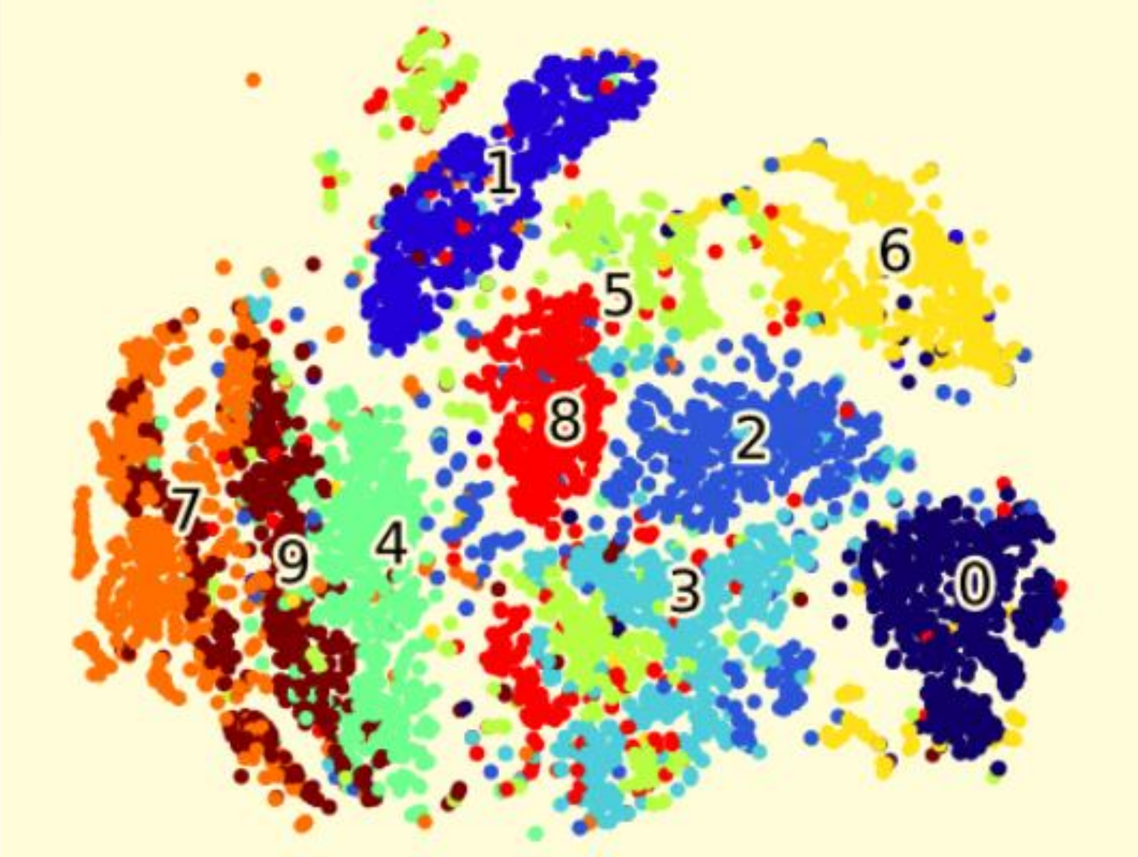
一个直接的应用

- 隐含语义挖掘（LSI 或称 LSA）

T-SNE T分布随机近邻嵌入

1. PCA和LDA都是线性的，t-sne是一种非线性的降维算法，非常适用于将高维数据降维到2维和3维，进行可视化。
2. SNE构建一个高维对象之间的概率分布，使得相似的对象有更高的概率被选择，而不相似的对象有较低的概率被选择。
3. SNE在低维空间里在构建这些点的概率分布，使得这两个概率分布之间尽可能的相似。

```python
from sklearn.manifold import TSNE
from matplotlib import patheffects as PathEffects
tsne = TSNE(n_components=2)
tsne_results = tsne.fit_transform(X_std)

def scatter(x, colors):

    # We create a scatter plot.
    f = plt.figure(figsize=(8, 8))
    ax = plt.subplot(aspect='equal')
    sc = ax.scatter(x[:,0], x[:,1], lw=0, s=40,
                    c = Target.values, cmap='jet',)
    plt.xlim(-25, 25)
    plt.ylim(-25, 25)
    ax.axis('off')
    ax.axis('tight')

    # We add the labels for each digit.
    txts = []
    for i in range(10):
        # Position of each label.
        xtext, ytext = np.median(x[colors == i, :], axis=0)
        txt = ax.text(xtext, ytext, str(i), fontsize=24)
        txt.set_path_effects([
            PathEffects.Stroke(linewidth=5, foreground="w"),
            PathEffects.Normal()])
        txts.append(txt)

    return f, ax, sc, txts


scatter(tsne_results, Target.values)
```



作业群里布置：