Python与数据科学导论-09

—— Pandas进阶，推荐算法

胡俊峰 北京大学

2023/03/20

主要内容：

- 序列数据类型概览
- Dataframe的表连接关系计算
- dataframe数据分析
- 协同过滤算法

序列数据类型：构造器

Series

Constructor

`Series`([data, index, dtype, name, copy, ...])	One-dimensional ndarray with axis labels (including time series).

Conversion

`Series.astype`(dtype[, copy, errors])	Cast a pandas object to a specified dtype `dtype`.
`Series.convert_dtypes`([infer_objects, ...])	Convert columns to best possible dtypes using dtypes supporting `pd.NA`.
`Series.infer_objects`()	Attempt to infer better dtypes for object columns.

数据访问方式：

Indexing, iteration

`Series.get`(key[, default])	Get item from object for given key (ex: DataFrame column).
`Series.at`	Access a single value for a row/column label pair.
`Series.iat`	Access a single value for a row/column pair by integer position.
`Series.loc`	Access a group of rows and columns by label(s) or a boolean array.
`Series.iloc`	Purely integer-location based indexing for selection by position.
`Series.__iter__`()	Return an iterator of the values.

数据访问方式：

Indexing, iteration

`Series.get`(key[, default])	Get item from object for given key (ex: DataFrame column).
`Series.at`	Access a single value for a row/column label pair.
`Series.iat`	Access a single value for a row/column pair by integer position.
`Series.loc`	Access a group of rows and columns by label(s) or a boolean array.
`Series.iloc`	Purely integer-location based indexing for selection by position.
`Series.__iter__`()	Return an iterator of the values.

显示序列的特征与属性：

Attributes

Axes

`Series.index`	The index (axis labels) of the Series.
`Series.array`	The ExtensionArray of the data backing this Series or Index.
`Series.values`	Return Series as ndarray or ndarray-like depending on the dtype.
`Series.dtype`	Return the dtype object of the underlying data.

逐元素的（二元）运算：

Binary operator functions

`Series.add`(other[, level, fill_value, axis])	Return Addition of series and other, element-wise (binary operator *add*).
`Series.sub`(other[, level, fill_value, axis])	Return Subtraction of series and other, element-wise (binary operator *sub*).
`Series.mul`(other[, level, fill_value, axis])	Return Multiplication of series and other, element-wise (binary operator *mul*).
`Series.div`(other[, level, fill_value, axis])	Return Floating division of series and other, element-wise (binary operator *truediv*).
`Series.ge`(other[, level, fill_value, axis])	Return Greater than or equal to of series and other, element-wise (binary operator *ge*).
`Series.ne`(other[, level, fill_value, axis])	Return Not equal to of series and other, element-wise (binary operator *ne*).

Function application, GroupBy & window

`Series.apply`(func[, convert_dtype, args])	Invoke function on values of Series.
`Series.agg`([func, axis])	Aggregate using one or more operations over the specified axis.
`Series.aggregate`([func, axis])	Aggregate using one or more operations over the specified axis.
`Series.transform`(func[, axis])	Call `func` on self producing a Series with the same axis shape as self.
`Series.map`(arg[, na_action])	Map values of Series according to an input mapping or function.
`Series.groupby`([by, axis, level, as_index, ...])	Group Series using a mapper or by a Series of columns.
`Series.rolling`(window[, min_periods, ...])	Provide rolling window calculations.

序列数据分析常用的一些特征指标：

`Series.any` (*[, axis, bool_only, skipna, level])	Return whether any element is True, potentially over an axis.
`Series.autocorr` ([lag])	Compute the lag-N autocorrelation.
`Series.between` (left, right[, inclusive])	Return boolean Series equivalent to left <= series <= right.
`Series.clip` ([lower, upper, axis, inplace])	Trim values at input threshold(s).
`Series.corr` (other[, method, min_periods])	Compute correlation with *other* Series, excluding missing values.
`Series.count` ([level])	Return number of non-NA/null observations in the Series.
`Series.cov` (other[, min_periods, ddof])	Compute covariance with Series, excluding missing values.

Reindexing / selection / label manipulation

`Series.reindex`(*args, **kwargs)	Conform Series to new index with optional filling logic.
`Series.reindex_like`(other[, method, copy, ...])	Return an object with matching indices as other object.
`Series.rename`([index, axis, copy, inplace, ...])	Alter Series index labels or name.
`Series.rename_axis`([mapper, inplace])	Set the name of the axis for the index or columns.
`Series.reset_index`([level, drop, name, ...])	Generate a new DataFrame or Series with the index reset.

pandas.Series.reindex

Show Source

`Series.reindex(*args, **kwargs)` [source]

Conform Series to new index with optional filling logic.

Places NA/NaN in locations having no value in the previous index. A new object is produced unless the new index is equivalent to the current one and `copy=False`.

Parameters: **index** : *array-like, optional*

New labels / index to conform to, should be specified using keywords. Preferably an Index object to avoid duplicating data.

method : *{None, 'backfill'/'bfill', 'pad'/'ffill', 'nearest'}*

Method to use for filling holes in reindexed DataFrame. Please note: this is only applicable to DataFrames/Series with a monotonically increasing/decreasing index.

- None (default): don't fill gaps
- pad / ffill: Propagate last valid observation forward to next valid.
- backfill / bfill: Use next valid observation to fill gap.
- nearest: Use nearest valid observations to fill gap.

copy : *bool, default True*

Return a new object, even if the passed indexes are the same.

level : *int or name*

Broadcast across a level, matching Index values on the passed MultiIndex level.

```
>>> index = ['Firefox', 'Chrome', 'Safari', 'IE10', 'Konqueror']
>>> df = pd.DataFrame({'http_status': [200, 200, 404, 404, 301],
...                    'response_time': [0.04, 0.02, 0.07, 0.08, 1.0]},
...                   index=index)
>>> df
            http_status  response_time
Firefox             200           0.04
Chrome              200           0.02
Safari              404           0.07
IE10                404           0.08
Konqueror           301           1.00
```

Create a new index and reindex the dataframe. By default values in the new index that do not have corresponding records in the dataframe are assigned `NaN`.

```
>>> new_index = ['Safari', 'Iceweasel', 'Comodo Dragon', 'IE10',
...              'Chrome']
>>> df.reindex(new_index)
               http_status  response_time
Safari               404.0           0.07
Iceweasel              NaN            NaN
Comodo Dragon          NaN            NaN
IE10                 404.0           0.08
Chrome               200.0           0.02
```

We can fill in the missing values by passing a value to the keyword `fill_value`. Because the

pandas.Series.reset_index

Show Source

```
Series.reset_index(level=None, *, drop=False, name=_NoDefault.no_default,
inplace=False, allow_duplicates=False)                                [source]
```

Generate a new DataFrame or Series with the index reset.

This is useful when the index needs to be treated as a column, or when the index is meaningless and needs to be reset to the default before another operation.

Parameters: **level** : ***int, str, tuple, or list, default optional***

For a Series with a MultiIndex, only remo
index. Removes all levels by default.

drop : ***bool, default False***

Just reset the index, without inserting it

name : ***object, optional***

The name to use for the column contain
`self.name` by default. This argument is

inplace : ***bool, default False***

Modify the Series in place (do not creat

allow_duplicates : ***bool, default False***

Allow duplicate column labels to be cre

```
>>> s = pd.Series([1, 2, 3, 4], name='foo',
...               index=pd.Index(['a', 'b', 'c', 'd'], name='idx'))
```

Generate a DataFrame with default index.

```
>>> s.reset_index()
  idx  foo
0   a    1
1   b    2
2   c    3
3   d    4
```

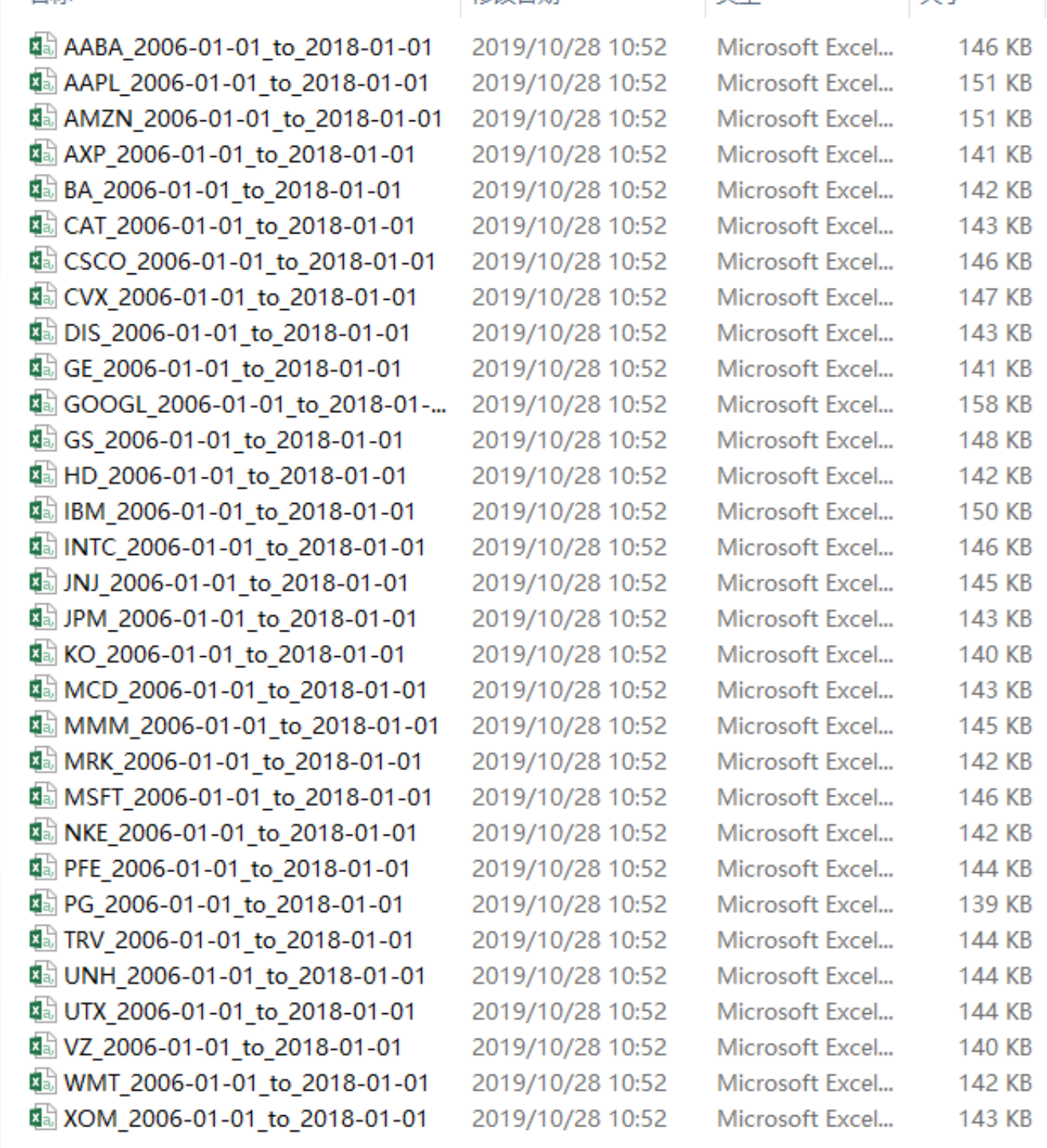
小结一下：

- Serial类型的数据本体是数组
- 索引的本质是数据的语义标签，理论上与数据等长（可以有空数据）
  - 在索引标记的支持下，可以对数据进行分组-聚合计算
- Serial类型支持合法的逐元素的计算和外部函数计算
- pandas对Serial数据整体还开发提供了较丰富的统计信息，趋势信息
- pandas对时间序列数据提供了一些方法进行支持
  - 比较适合金融量化分析需要的功能

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.reset_index.html#pandas.Series.reset_index

股市散户投资策略：

- 已知10年的数据
  - 以及已经有的历史数据
- 10万资金（0.1%交易税）
- 2年后能剩多少钱？



AABA_2006-01-01_to_2018-01-01	2019/10/28 10:52	Microsoft Excel...	146 KB
AAPL_2006-01-01_to_2018-01-01	2019/10/28 10:52	Microsoft Excel...	151 KB
AMZN_2006-01-01_to_2018-01-01	2019/10/28 10:52	Microsoft Excel...	151 KB
AXP_2006-01-01_to_2018-01-01	2019/10/28 10:52	Microsoft Excel...	141 KB
BA_2006-01-01_to_2018-01-01	2019/10/28 10:52	Microsoft Excel...	142 KB
CAT_2006-01-01_to_2018-01-01	2019/10/28 10:52	Microsoft Excel...	143 KB
CSCO_2006-01-01_to_2018-01-01	2019/10/28 10:52	Microsoft Excel...	146 KB
CVX_2006-01-01_to_2018-01-01	2019/10/28 10:52	Microsoft Excel...	147 KB
DIS_2006-01-01_to_2018-01-01	2019/10/28 10:52	Microsoft Excel...	143 KB
GE_2006-01-01_to_2018-01-01	2019/10/28 10:52	Microsoft Excel...	141 KB
GOOGL_2006-01-01_to_2018-01-...	2019/10/28 10:52	Microsoft Excel...	158 KB
GS_2006-01-01_to_2018-01-01	2019/10/28 10:52	Microsoft Excel...	148 KB
HD_2006-01-01_to_2018-01-01	2019/10/28 10:52	Microsoft Excel...	142 KB
IBM_2006-01-01_to_2018-01-01	2019/10/28 10:52	Microsoft Excel...	150 KB
INTC_2006-01-01_to_2018-01-01	2019/10/28 10:52	Microsoft Excel...	146 KB
JNJ_2006-01-01_to_2018-01-01	2019/10/28 10:52	Microsoft Excel...	145 KB
JPM_2006-01-01_to_2018-01-01	2019/10/28 10:52	Microsoft Excel...	143 KB
KO_2006-01-01_to_2018-01-01	2019/10/28 10:52	Microsoft Excel...	140 KB
MCD_2006-01-01_to_2018-01-01	2019/10/28 10:52	Microsoft Excel...	143 KB
MMM_2006-01-01_to_2018-01-01	2019/10/28 10:52	Microsoft Excel...	145 KB
MRK_2006-01-01_to_2018-01-01	2019/10/28 10:52	Microsoft Excel...	142 KB
MSFT_2006-01-01_to_2018-01-01	2019/10/28 10:52	Microsoft Excel...	146 KB
NKE_2006-01-01_to_2018-01-01	2019/10/28 10:52	Microsoft Excel...	142 KB
PFE_2006-01-01_to_2018-01-01	2019/10/28 10:52	Microsoft Excel...	144 KB
PG_2006-01-01_to_2018-01-01	2019/10/28 10:52	Microsoft Excel...	139 KB
TRV_2006-01-01_to_2018-01-01	2019/10/28 10:52	Microsoft Excel...	144 KB
UNH_2006-01-01_to_2018-01-01	2019/10/28 10:52	Microsoft Excel...	144 KB
UTX_2006-01-01_to_2018-01-01	2019/10/28 10:52	Microsoft Excel...	144 KB
VZ_2006-01-01_to_2018-01-01	2019/10/28 10:52	Microsoft Excel...	140 KB
WMT_2006-01-01_to_2018-01-01	2019/10/28 10:52	Microsoft Excel...	142 KB
XOM_2006-01-01_to_2018-01-01	2019/10/28 10:52	Microsoft Excel...	143 KB

pandas.DataFrame.groupby — pandas 1.5.3 documentation

https://pandas.pydata.org/pandas-docs/stable/...

Web Group DataFrame using a mapper or by a Series of columns. A **groupby** operation involves some combination of splitting the object, applying a function, and combining the results. ...

pandas.DataFrame.transfo...

pandas.DataFrame.transform# DataFrame. transform (func, axis = 0, * args, ** ...

User Guide

To support column-specific aggregation with control over the output column ...

pandas.DataFrame.copy

pandas.DataFrame.copy# DataFrame. copy (deep = True) [source] # Make a copy of ...

pandas.DataFrame.aggreg...

pandas.DataFrame.aggregate# DataFrame. aggregate (func = None, axis = 0, * args, ...

pandas.DataFrame.gt

See also. DataFrame.eq. Compare DataFrames for equality elementwise. ...

pandas.DataFrame.count

pandas.DataFrame.count# DataFrame. count (axis = 0, level = None, ...

pandas.DataFrame.get

pandas.DataFrame.get# DataFrame. get (key, default = None) [source] # Get item ...

pandas.core.groupby.DataFr...

pandas.core.groupby.DataFrameGroupBy.de
DataFrameGroupBy. describe (** kwargs) ...

pandas.DataFrame.sum

min_count int, default 0. The required number of valid values to perform the ...

pandas.core.groupby.Data...

pandas.core.groupby.DataFrameGroupBy.ag
DataFrameGroupBy.agg (arg, *args, ...

https://pandas.pydata.org/pandas-docs/stable/reference/frame.html

DataFrame

Constructor

`DataFrame`([data, index, columns, dtype, copy])	Two-dimensional, size-mutable, potentially heterogeneous tabular data.

Attributes and underlying data

Axes

`DataFrame.index`	The index (row labels) of the DataFrame.
`DataFrame.columns`	The column labels of the DataFrame.
`DataFrame.dtypes`	Return the dtypes in the DataFrame.
`DataFrame.info`([verbose, buf, max_cols, ...])	Print a concise summary of a DataFrame.
`DataFrame.select_dtypes`([include, exclude])	Return a subset of the DataFrame's columns based on the column dtypes.

```
# hierarchical indices and columns
index = pd.MultiIndex.from_product([[2013, 2014], [1, 2, 3, 4]],
                                   names=['year', 'season'])  # 行、列都是2维索引
columns = pd.MultiIndex.from_product([['Bob1', 'Bob2', 'Bob3'], ['WR', 'HD']],
                                     names=['name', 'type'])

# mock some data
data = np.random.randint(1,91, 48)  # 1-91之间的整数，48个
data = data.reshape(8,6)
# create the DataFrame
wh_data = pd.DataFrame(data, index=index, columns=columns)
wh_data
```

	name	Bob1		Bob2		Bob3	
	type	WR	HD	WR	HD	WR	HD
year	season						
2013	1	55	29	27	74	75	48
	2	9	86	54	5	11	14
	3	18	22	79	42	39	73
	4	77	77	52	77	65	17
2014	1	45	30	9	43	16	18

多维索引表操作

直接定位的索引访问方式：

```
wh_data.iloc[:3,:3]  # 可以按二维表的iloc直接切片
```

	name	Bob		Guido
	type	WR	HD	WR
year	season			
2013	1	39	14	2
	2	46	20	21
	3	50	1	74

.loc可以实现最外层切片访问

```
wh_data.loc[2014:,:'Bob2']    # 右面也是闭区间
# wh_data[2014:,:'Bob2']  # TypeError: unhashable type: 'slice'
```

	name	Bob1		Bob2	
	type	WR	HD	WR	HD
year	season				
2014	1	27	85	69	78
	2	2	26	27	16
	3	17	9	42	41
	4	71	33	43	66

使用indexSlice对象实现多重索引下的切片

```
# wh_data.loc[2014:,(:'WR')]  # invalid syntax
idx = pd.IndexSlice
wh_data.loc[idx[2014:],idx[ :,'WR']]    # 加入index切片对象辅助实现
```

	name	Bob1	Bob2	Bob3
	type	WR	WR	WR
year	season			
2014	1	27	69	50
	2	2	27	56
	3	17	42	82
	4	71	43	59

按索引排序操作：（基本了解就行）

```
# data = data.sort_index()
try:
    data['a':'b']       # 区间访问有问题
except KeyError as e:   # 没有按键值形成物理序
    print(type(e))
    print(e)
```

```
<class 'pandas.errors.UnsortedIndexError'>
'MultiIndex slicing requires the index to be lexsorted: slicing on levels [0], lexso
rt depth 0'
```

```
data = data.sort_index()   # index排个序
data['a':'b']              # 左右都是闭区间
#data[:,1:3]                # 这个还是不行
```

```
char  int
a     1      0.150613
      2      0.725752
      3      0.647706
      4      0.224172
b     1      0.795473
      2      0.289532
```

表连接及组合Combining Datasets: Merge and Join

- *one-to-one*
- *many-to-one*
- *many-to-many* joins.

One-to-one joins :

```
df1 = pd.DataFrame({'employee': ['Bob', 'Jake', 'Lisa', 'Sue'],
                    'group': ['Accounting', 'Engineering', 'Engineering', 'HR']})
df2 = pd.DataFrame({'employee': ['Lisa', 'Bob', 'Jake', 'Sue'],
                    'hire_date': [2004, 2008, 2012, 2014]})
display('df1', 'df2')
```

df1

	employee	group
0	Bob	Accounting
1	Jake	Engineering
2	Lisa	Engineering
3	Sue	HR

df2

	employee	hire_date
0	Lisa	2004
1	Bob	2008
2	Jake	2012
3	Sue	2014

```
df3 = pd.merge(df1, df2)
df3
```

两个表有一个同名字段

自动按公共字段为轴进行合并

	employee	group	hire_date
0	Bob	Accounting	2008
1	Jake	Engineering	2012
2	Lisa	Engineering	2004
3	Sue	HR	2014

指定特定字段进行关联 merge-on

```
display('df1', 'df2', "pd.merge(df1, df2, on='employee')")
```

df1

	employee	group
0	Bob	Accounting
1	Jake	Engineering
2	Lisa	Engineering
3	Sue	HR

df2

	employee	hire_date
0	Lisa	2004
1	Bob	2008
2	Jake	2012
3	Sue	2014

pd.merge(df1, df2, on='employee')

	employee	group	hire_date
0	Bob	Accounting	2008
1	Jake	Engineering	2012
2	Lisa	Engineering	2004
3	Sue	HR	2014

left_on,right_on可以指定不同名字段关联

```
df3 = pd.DataFrame({'name': ['Bob', 'Jake', 'Lisa', 'Sue'],
                    'salary': [70000, 80000, 120000, 90000]})
display('df1', 'df3', 'pd.merge(df1, df3, left_on="employee", right_on="name")')
```

df1

	employee	group
0	Bob	Accounting
1	Jake	Engineering
2	Lisa	Engineering
3	Sue	HR

df3

	name	salary
0	Bob	70000
1	Jake	80000
2	Lisa	120000
3	Sue	90000

pd.merge(df1, df3, left_on="employee", right_on="name")

	employee	group	name	salary
0	Bob	Accounting	Bob	70000
1	Jake	Engineering	Jake	80000
2	Lisa	Engineering	Lisa	120000
3	Sue	HR	Sue	90000

The result has a redundant column that we can drop if desired–for example, by using the `drop()` method of `DataFrame`s:

```
pd.merge(df1, df3, left_on="employee", right_on="name").drop('name', axis=1)
```

Many-to-one joins 有重键值与对应的唯一键值进行**join**，单值对应的记录展开为多个：

```
df4 = pd.DataFrame({'group': ['Accounting', 'Engineering', 'HR'],
                    'supervisor': ['Carly', 'Guido', 'Steve']})
display('df3', 'df4', 'pd.merge(df3, df4)')
```

df3

	employee	group	hire_date
0	Bob	Accounting	2008
1	Jake	Engineering	2012
2	Lisa	Engineering	2004
3	Sue	HR	2014

df4

	group	supervisor
0	Accounting	Carly
1	Engineering	Guido
2	HR	Steve

pd.merge(df3, df4)

	employee	group	hire_date	supervisor
0	Bob	Accounting	2008	Carly
1	Jake	Engineering	2012	Guido
2	Lisa	Engineering	2004	Guido
3	Sue	HR	2014	Steve

Many-to-many joins：
多对多关联操作可以认为是一对多操作的复用

```
df5 = pd.DataFrame({'group': ['Accounting', 'Accounting',
                              'Engineering', 'Engineering', 'HR', 'HR'],
                    'skills': ['math', 'spreadsheets', 'coding', 'linux',
                               'spreadsheets', 'organization']})
display('df1', 'df5', "pd.merge(df1, df5)")
```

df1

	employee	group
0	Bob	Accounting
1	Jake	Engineering
2	Lisa	Engineering
3	Sue	HR

df5

	group	skills
0	Accounting	math
1	Accounting	spreadsheets
2	Engineering	coding
3	Engineering	linux
4	HR	spreadsheets
5	HR	organization

pd.merge(df1, df5)

	employee	group	skills
0	Bob	Accounting	math
1	Bob	Accounting	spreadsheets
2	Jake	Engineering	coding
3	Jake	Engineering	linux
4	Lisa	Engineering	coding
5	Lisa	Engineering	linux
6	Sue	HR	spreadsheets
7	Sue	HR	organization

Inner-join 求索引交集

```
df6 = pd.DataFrame({'name': ['Peter', 'Paul', 'Mary'],
                    'food': ['fish', 'beans', 'bread']},
                   columns=['name', 'food'])
df7 = pd.DataFrame({'name': ['Mary', 'Joseph'],
                    'drink': ['wine', 'beer']},
                   columns=['name', 'drink'])
display('df6', 'df7', 'pd.merge(df6, df7)')
```

```
pd.merge(df6, df7, how='inner')
```

	name	food	drink
0	Mary	bread	wine

df6

	name	food
0	Peter	fish
1	Paul	beans
2	Mary	bread

df7

	name	drink
0	Mary	wine
1	Joseph	beer

pd.merge(df6, df7)

	name	food	drink
0	Mary	bread	wine

outer-join是索引的并集（left join，right join...）

```
display('df6', 'df7', "pd.merge(df6, df7, how='outer')")
```

df6

	name	food
0	Peter	fish
1	Paul	beans
2	Mary	bread

df7

	name	drink
0	Mary	wine
1	Joseph	beer

```
pd.merge(df6, df7, how='outer')
```

	name	food	drink
0	Peter	fish	NaN
1	Paul	beans	NaN
2	Mary	bread	wine
3	Joseph	NaN	beer

Data Aggregations on Multi-Indices

- Pandas has built-in data aggregation methods,
- such as mean(), sum(), and max().
- For hierarchically indexed data, these can be passed a level parameter that controls which **subset of the data the aggregate is computed on**.

- Group by certain Key

```
pd.options.display.max_rows = 6 # 最多显示6行

exam_data = {
    'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael', 'Matthew',
    'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],
    'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
    'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']
}
labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

# 以labels为行索引，生成一个exam_data的DataFrame
df = pd.DataFrame(exam_data, index = labels)
df
```

	name	score	attempts	qualify
a	Anastasia	12.5	1	yes
b	Dima	9.0	3	no
c	Katherine	16.5	2	yes
...	...	...	...	...
h	Laura	NaN	1	no
i	Kevin	8.0	2	no
j	Jonas	19.0	1	yes

10 rows × 4 columns

生成一个成绩表

输出数据表的结构信息、基本统计信息

```
# df的基本信息
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 10 entries, a to j
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   name      10 non-null     object
 1   score     8 non-null      float64
 2   attempts  10 non-null     int64
 3   qualify   10 non-null     object
dtypes: float64(1), int64(1), object(2)
memory usage: 400.0+ bytes
```

```
# df的统计信息 describe
print(df.describe())
```

```
            score   attempts
count    8.000000  10.000000
mean    13.562500   1.900000
std      4.693746   0.875595
min      8.000000   1.000000
25%      9.000000   1.000000
50%     13.500000   2.000000
75%     17.125000   2.750000
max     20.000000   3.000000
```

```
df = pd.DataFrame({'A':['bob','john','bob','jeff','bob','jeff','bob','john'],
                   'B':['one','one','two','three','two','two','one','three'],
                   'C':[3,1,4,1,5,9,2,6],
                   'D':[1,2,3,4,5,6,7,8]})   # 给出4栏数据
df
```

	A	B	C	D
0	bob	one	3	1
1	john	one	1	2
2	bob	two	4	3
3	jeff	three	1	4
4	bob	two	5	5
5	jeff	two	9	6
6	bob	one	2	7
7	john	three	6	8

```
g = df.groupby('A').count()   # 此时可以看作是按字段‘A’的key
g
```

A	B	C	D
bob	4	4	4
jeff	2	2	2
john	2	2	2

```
g = df.groupby('A').sum()    # 可加字段参与运算
g
```

A	C	D
bob	14	16
jeff	10	10
john	7	10

Pandas apply方法

```
d = grouped.apply(lambda x:x.head(2)) # 留前两条
d
```

		A	B	C	D
A					
bob	0	bob	one	3	1
	2	bob	two	4	3
jeff	3	jeff	three	1	4
	5	jeff	two	9	6
john	1	john	one	1	2
	7	john	three	6	8

```python
import pandas as pd

df = pd.DataFrame({'A':['bob','john','bob','jeff','bob','jeff','bob','john'],
                   'B':['one','one','two','three','two','two','one','three'],
                   'C':[3,1,4,1,5,9,2,6],
                   'D':[1,2,3,4,5,6,7,8]})  # 给出4栏数据

grouped = df.groupby('A')  # 按 属性A的值进行分组

for name,group in grouped:
    print(name)   # 唯一的属性值
    print(group)
```

```
bob
     A    B  C  D
0  bob  one  3  1
2  bob  two  4  3
4  bob  two  5  5
6  bob  one  2  7
jeff
      A      B  C  D
3  jeff  three  1  4
5  jeff    two  9  6
john
      A      B  C  D
1  john    one  1  2
7  john  three  6  8
```

应用例子：给定文档集，统计词表的TF*IDF

- 按关键词集合实现文档检索
- 利用文档的词向量表达计算文档相似度或实现话题聚类？
- 利用词的文档分布计算词义相似度或实现词义聚类？

词袋子方案下的文档词向量 Incidence Vector

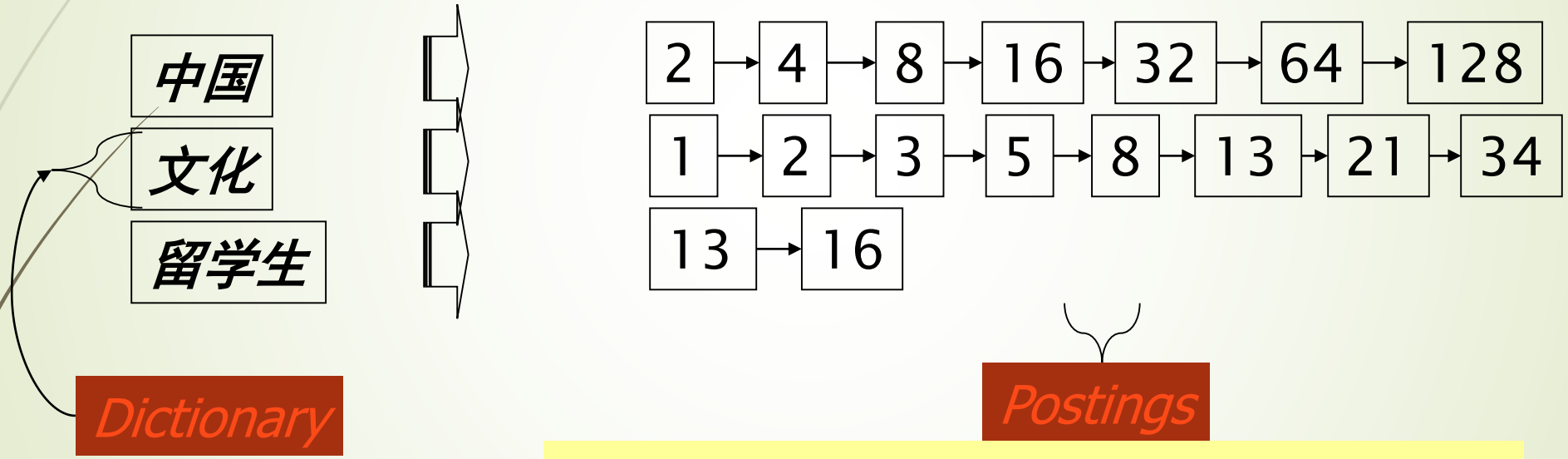
- term-document 关联矩阵
- 每个term对应一个0/1或词频 向量, incidence vector

	D_1	D_2	D_3	D_4	D_5	D_6	…
中国	1	0	1	1	1	0	
文化	1	1	0	0	1	0	
日本	0	1	1	0	1	1	
留学生	0	1	1	1	0	0	
教育	1	0	0	1	0	0	
北京	1	0	0	0	1	1	
…							

1 or TF if page contains word, otherwise 0

倒排索引 Inverted index

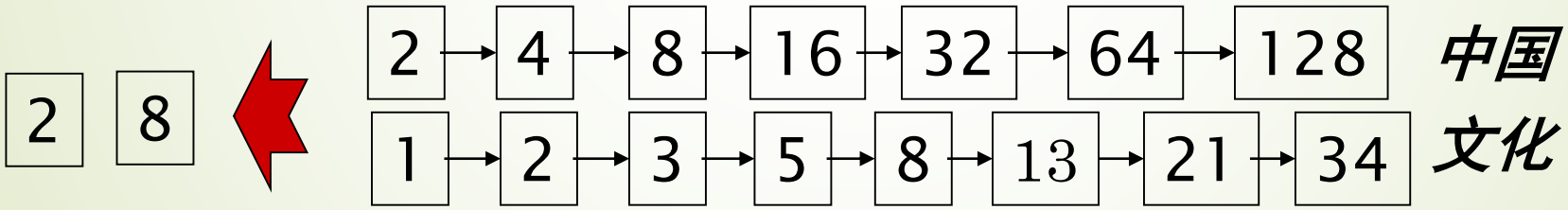
- 对每个 term *T*: 保存包含T的文档(编号)列表



Sorted by docID (more later on why).

Boolean Query processing

- 查询:*中国* AND *文化*
  - 查找Dictionary，定位*中国*;
    - 读取对应的postings.
  - 查找Dictionary，定位*文化*;
    - 读取对应的postings.
  - “Merge” 合并(AND)两个postings:



ranking: density-based

- 按query，给文档打分scoring，根据score排序
- Idea
  - 如果一个文档 talks about a topic *more*, then it is a *better* match
    - ➔如果包含很多次query term的出现，文档是relevant(相关的)
  - ➔ **term weighting?**

电影数据

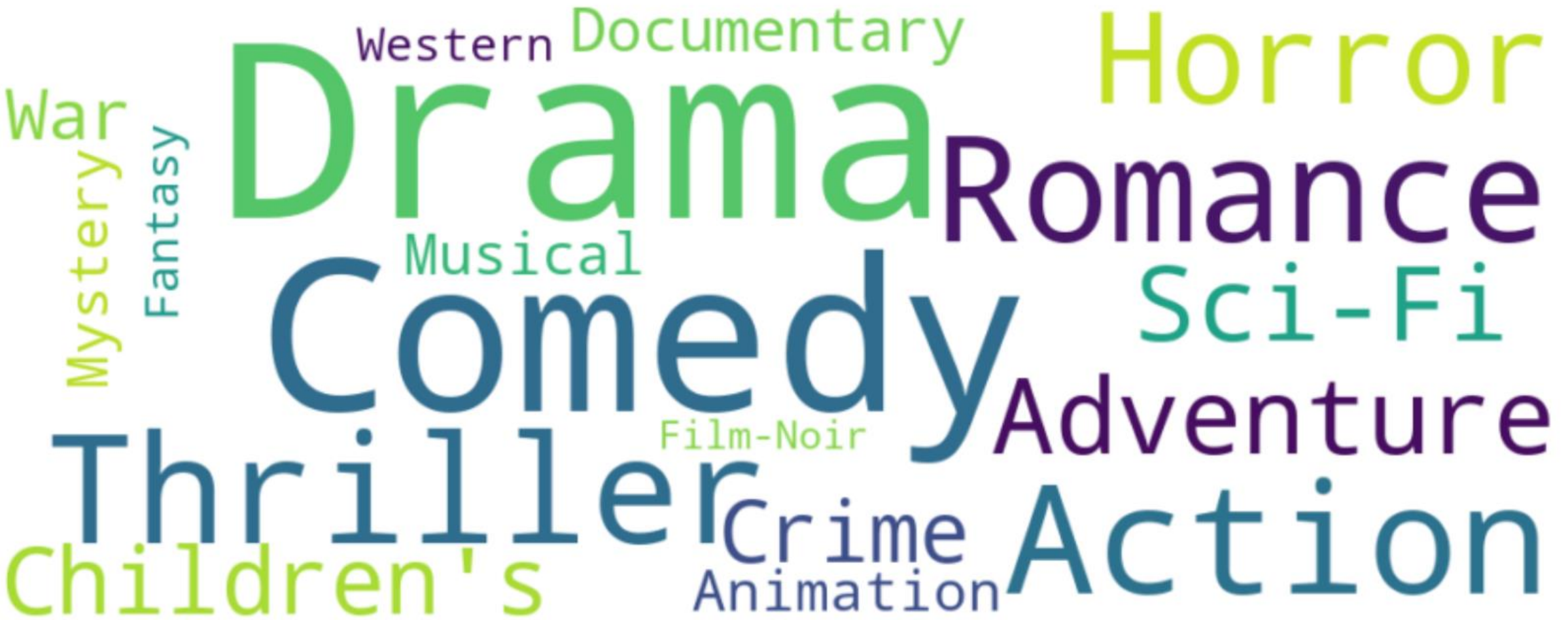
```
# Join all 3 files into one dataframe
dataset = pd.merge(pd.merge(movies, ratings),users)
# Display 20 movies with highest ratings
dataset[['title','genres','rating']].sort_values('rating', ascending=False).head(20)
```

	title	genres	rating
0	Toy Story (1995)	Animation|Children's|Comedy	5
489283	American Beauty (1999)	Comedy|Drama	5
489259	Election (1999)	Comedy	5
489257	Matrix, The (1999)	Action|Sci-Fi|Thriller	5
489256	Dead Ringers (1988)	Drama|Thriller	5
489237	Rushmore (1998)	Comedy	5
489236	Simple Plan, A (1998)	Crime|Thriller	5
489226	Hands on a Hard Body (1996)	Documentary	5
489224	Pleasantville (1998)	Comedy	5
489212	Say Anything... (1989)	Comedy|Drama|Romance	5

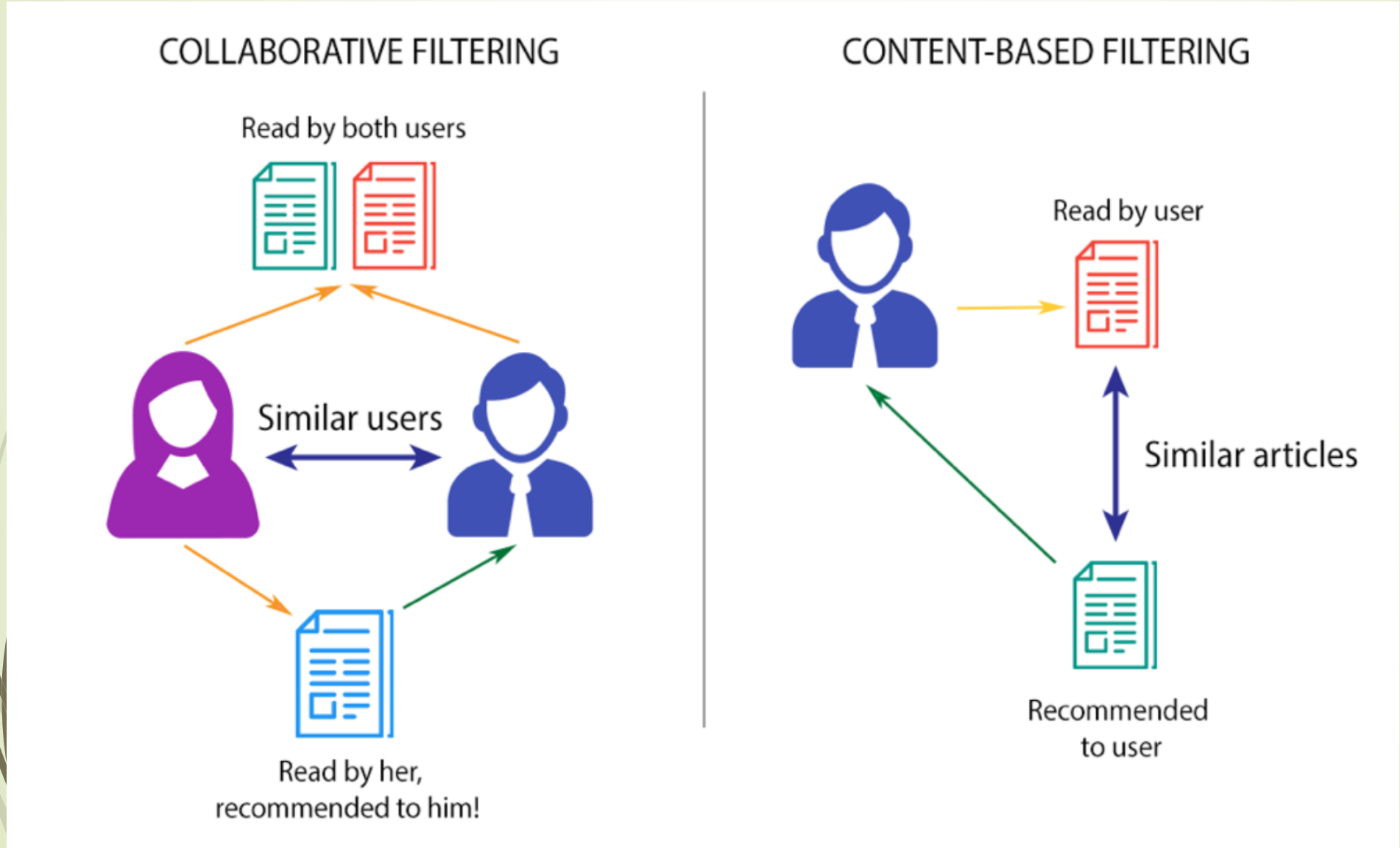
关键词统计

```python
# Make a census of the genre keywords
genre_labels = set()
for s in movies['genres'].str.split('|').values:
    genre_labels = genre_labels.union(set(s))

# Function that counts the number of times each of the genre keywords appear
def count_word(dataset, ref_col, census):
    keyword_count = dict()
    for s in census:
        keyword_count[s] = 0
    for census_keywords in dataset[ref_col].str.split('|'):
        if type(census_keywords) == float and pd.isnull(census_keywords):
            continue
        for s in [s for s in census_keywords if s in census]:
            if pd.notnull(s):
                keyword_count[s] += 1
    #____________________________________________________________
    # convert the dictionary in a list to sort the keywords by frequency
    keyword_occurences = []
    for k,v in keyword_count.items():
        keyword_occurences.append([k,v])
    keyword_occurences.sort(key = lambda x:x[1], reverse = True)
    return keyword_occurences, keyword_count

# Calling this function gives access to a list of genre keywords which are sorted by decreasing frequency
keyword_occurences, dum = count_word(movies, 'genres', genre_labels)
keyword_occurences[:5]
```

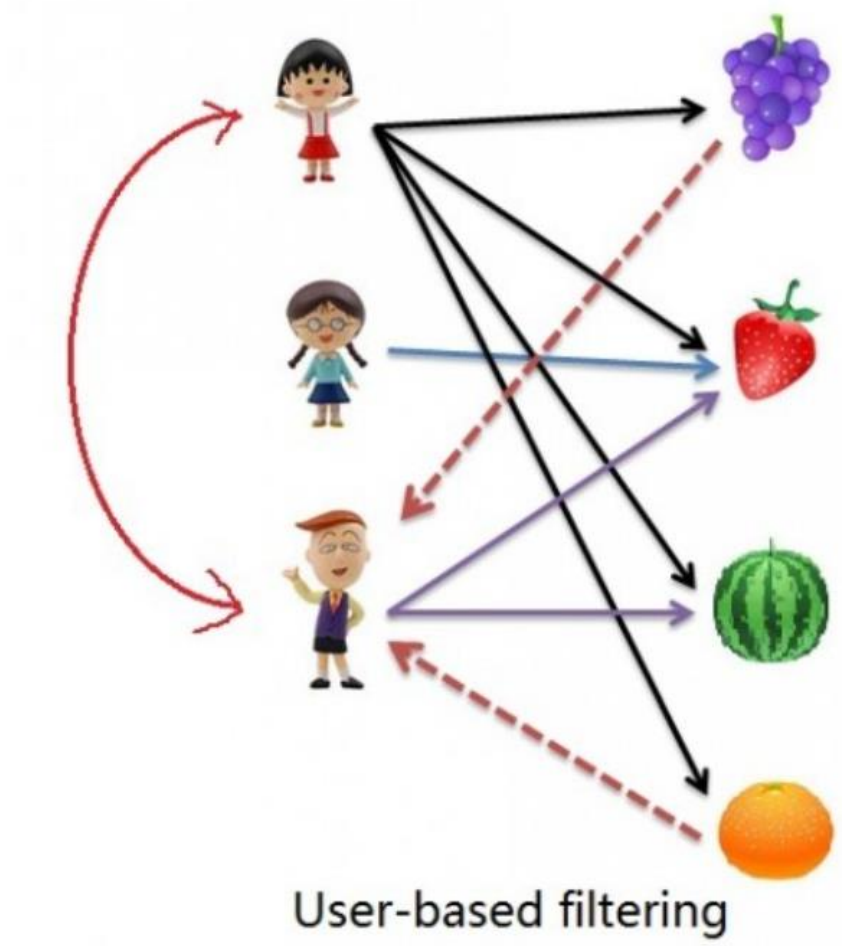
```
10
11 # Plot the wordcloud
12 f, ax = plt.subplots(figsize=(16, 8))
13 plt.imshow(genre_wordcloud, interpolation="bilinear")
14 plt.axis('off')
15 plt.show()
```

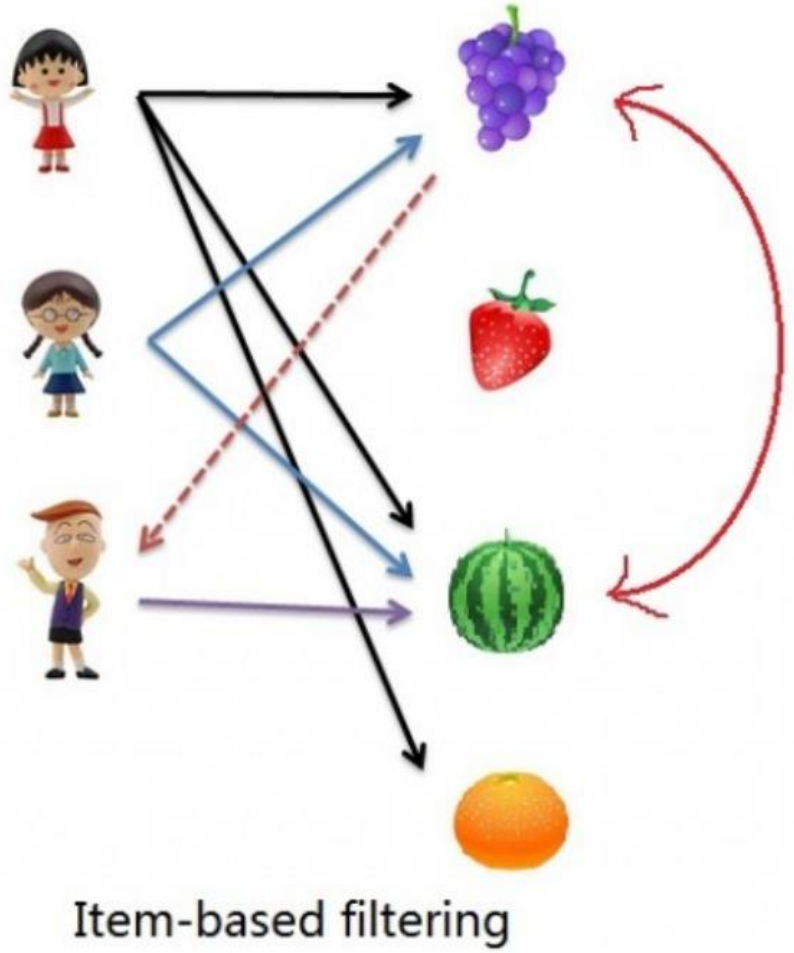


协同过滤推荐：


COLLABORATIVE FILTERING
Read by both users
Similar users
Read by her,
recommended to him!
CONTENT-BASED FILTERING
Read by user
Similar articles
Recommended
to user

1. **User-User Collaborative Filtering**: Here we find look alike users based on similarity and recommend movies which first user's look-alike has chosen in past. This algorithm is very effective but takes a lot of time and resources. It requires to compute every user pair information which takes time. Therefore, for big base platforms, this algorithm is hard to implement without a very strong parallelizable system.
2. **Item-Item Collaborative Filtering**: It is quite similar to previous algorithm, but instead of finding user's look-alike, we try finding movie's look-alike. Once we have movie's look-alike matrix, we can easily recommend alike movies to user who have rated any movie from the dataset. This algorithm is far less resource consuming than user-user collaborative filtering. Hence, for a new user, the algorithm takes far lesser time than user-user collaborate as we don't need all similarity scores between users. And with fixed number of movies, movie-movie look alike matrix is fixed over time.





tf x idf term weights

- tf x idf 权值计算公式:
  - term frequency (*tf*)
    - or *wf*, some measure of term density in a doc
  - inverse document frequency (*idf*)
    - 表达term的重要度(稀有度)
    - 原始值$idf_t = 1/df_t$
      - 通常会作平滑

$$idf_t = \log\left(\frac{N}{df_t}\right)$$

- 实际文档词向量中计算其tf*idf权重来表达与query的相关性:

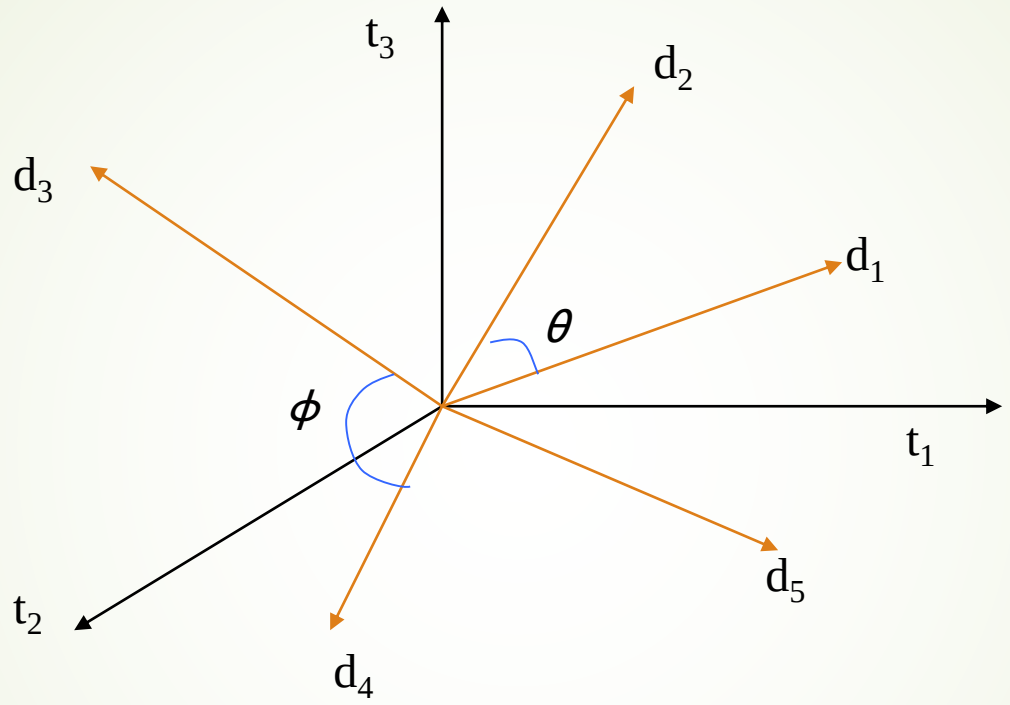
$$w_{t,d} = tf_{t,d} \times \log(N / df_t)$$

Documents to vectors

	D_1	D_2	D_3	D_4	D_5	D_6	…
中国	4.1	0.0	3.7	5.9	3.1	0.0	
文化	4.5	4.5	0	0	11.6	0	
日本	0	3.5	2.9	0	2.1	3.9	
留学生	0	3.1	5.1	12.8	0	0	
教育	2.9	0	0	2.2	0	0	
北京	7.1	0	0	0	4.4	3.8	
…							

- 每一个文档 *j* 能够被看作一个向量，每个term 是一个维度，取值为tf.idf
- So we have a vector space
  - terms are axes
  - docs live in this space
  - 高维空间：即使作stemming, may have 20,000+ dimensions

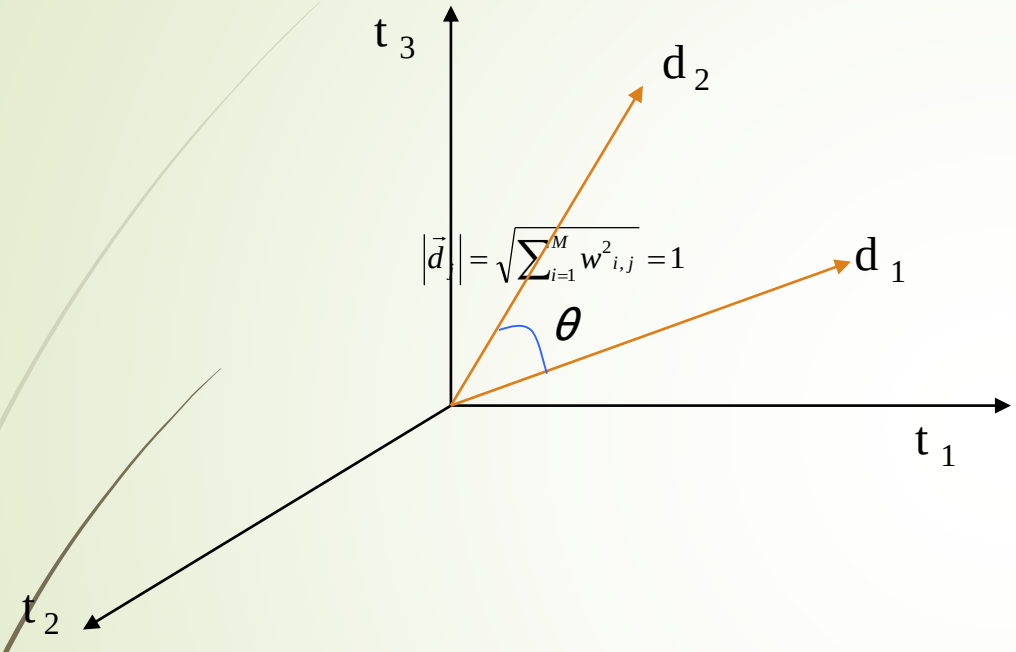
文档词向量空间模型（简称向量空间模型）



一个符合直觉的假定：在vector space中“close together”的文档会talk about the same things.

用例：Query-by-example，Free Text query as vector

Cosine similarity



- 向量d_1和d_2的“closeness”可以用它们之间的夹角大小来度量
- 具体的，可用cosine of the angle x*来计算向量相似度*.
- 向量按长度归一化Normalization

$$sim(d_j, d_k) = \frac{\vec{d}_j \cdot \vec{d}_k}{\left|\vec{d}_j\right|\left|\vec{d}_k\right|} = \frac{\sum_{i=1}^{M} w_{i,j} w_{i,k}}{\sqrt{\sum_{i=1}^{M} w_{i,j}^2}\sqrt{\sum_{i=1}^{M} w_{i,k}^2}}$$

计算文档的TF*IDF向量

- 统计文档总数
- 统计每篇文档的TF向量
- 统计每个词出现的文档数
- 计算每个词的IDF
- 在TF向量中乘上对应词的IDF（可以按词表广播？）

自定义最大向前匹配函数

```
table['words'] = table['contance'].apply(lambda x: ' '.join(list(cut(x, word_list, 3))))
```

```
table
```

	ID	Poem_id	line_number	contance	words
0	1	4371	-100	##錢唐永昌(一作錢唐郎中洛陽令)	錢 唐 永昌 一作 錢 唐 郎中 洛陽 令
1	2	4371	-1	SS沈佺期	沈 期
2	3	4371	1	洛陽舊有(一作出)神明宰	洛陽 舊有 一作 出 神明 宰
3	4	4371	2	草韌由來天地中	草韌 由來 天地 中
4	5	4371	3	餘邑政成何足貴	餘 邑 政成 何足 貴
...	...	...	...	...	...
46272	46273	39205	-1	SS李舜弦	李 舜弦

1.2 统计每个词的TF-IDF值

```
# 按照空格分开，stack一下
split_words = table['words'].str.split(' ', expand=True).stack().rename('word').reset_index()
new_data = pd.merge(table['Poem_id'], split_words, left_index=True, right_on='level_0')
new_data
```

	Poem_id	level_0	level_1	word
0	4371	0	0	饑
1	4371	0	1	唐
2	4371	0	2	永昌
3	4371	0	3	一作
4	4371	0	4	饑
...	...	...	...	...
198498	39205	46275	4	屏

https://www.oreilly.com/library/view/python-data-science/9781491912126/

1 生成“词-上下文词”的二维索引序列表

- grouby + merge(, how = ‘cross’):

	doc_id	position	word
0	1	0	i
1	1	1	know
2	1	2	that
3	1	3	the
4	1	4	day
5	1	5	will
6	1	6	come
7	1	7	when
8	1	8	my
9	1	9	sight

doc_id		position_x	word_x	position_y	word_y
1	0	0	i	0	i
	1	0	i	1	know
	2	0	i	2	that
	3	0	i	3	the
	4	0	i	4	day
...	...	...	...	...	...
9	164	12	overlooked	8	i
	165	12	overlooked	9	ever
	166	12	overlooked	10	spurned
	167	12	overlooked	11	and
	168	12	overlooked	12	overlooked

merge

	doc_id	level_1	position_x	word_x	position_y	word_y
1	1	1	0	i	1	know
2	1	2	0	i	2	that
3	1	3	0	i	3	the
4	1	4	0	i	4	day
5	1	5	0	i	5	will
...	...	...	...	...	...	...
1624	9	163	12	overlooked	7	that
1625	9	164	12	overlooked	8	i
1626	9	165	12	overlooked	9	ever
1627	9	166	12	overlooked	10	spurned
1628	9	167	12	overlooked	11	and

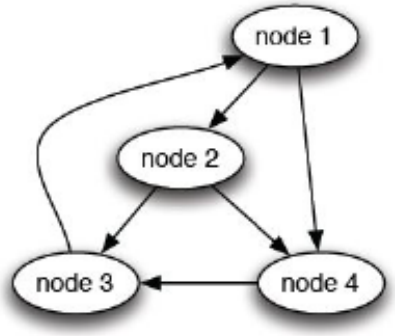
reset_index()之后

HITS算法：基于超文本链接的主题搜索排名

- 关键词检索生成base-set
- 通过base-set网页间的超链接关系生成扩充candidate –set
- Items之间的关联构成相互推荐的一个网络（矩阵）
- HITS算法求解最佳排名

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:

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

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

$h^k=(h_1, h_2, \ldots h_n)^T, a^k=(a_1, a_2, \ldots a_n)^T$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)

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

$$h^k = Ma^k \tag{6}$$

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

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

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

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 MM^T 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

$$h^k = = \frac{(MM^T)^k h^0}{\|(MM^T)^k h^0\|} \tag{14}$$

$$= \frac{q_1 c_1^k z_1 + q_2 c_2^k z_2 + \ldots q_n c_n^k z_n}{\|q_1 c_1^k z_1 + q_2 c_2^k z_2 + \ldots q_n c_n^k z_n\|} \tag{15}$$

$$= \frac{q_1 z_1 + q_2 (\frac{c_2}{c_1})^k z_2 + \ldots q_n (\frac{c_2}{c_1})^k z_n}{\|q_1 z_1 + q_2 (\frac{c_2}{c_1})^k z_2 + \ldots q_n (\frac{c_2}{c_1})^k z_n\|} \tag{16}$$

$$= \frac{q_1 z_1}{\|q_1 z_1\|} \tag{17}$$

故h_k收敛，同理可得a_k收敛

用Hits算法优化诗歌检索：

- 1、计算诗歌间的相似度：
  - 基于表或稀疏矩阵计算：poem-id：word向量的cosin距离 （归一化，A*A.T）
    - 高于的阈值的解读为有关联关系
- 2、用关键词命中一组诗歌（集）
- 3、将与这些诗歌有关联关系的诗歌加入诗歌集（诗歌间关联关系构成一对称矩阵）
- 4、迭代求该矩阵的主特征向量（全1向量乘X的n次，到收敛）
- 5、按与该向量的cosin结果排降序得到检索结果排序

Hits算法与pagrank

- pagrank加入了随机游走方案

$$PR(u) = \frac{1-d}{N} + d \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

矩阵降维、loss与平滑

- 矩阵分解与最小二乘（MSE）loss
- 奇异值分解与特征空间变换-降维
- SVD到LSI（隐含语义索引）

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

LSI (LSA)

- Singular Value Decomposition (SVD) used for the word-document matrix
  - A least-squares method for dimension reduction

	Term 1	Term 2	Term 3	Term 4
Query	user	interface		
Document 1	user	interface	HCI	interaction
Document 2			HCI	interaction

Classic LSI Example (Deerwester)

Titles

c1: *Human* machine *interface* for Lab ABC *computer* applications
c2: A *survey of user* opinion of *computer system response time*
c3: The *EPS user interface* management *system*
c4: *System* and *human system* engineering testing of *EPS*
c5: Relation of *user*-perceived *response time* to error measurement
m1: The generation of random, binary, unordered *trees*
m2: The intersection *graph* of paths in *trees*
m3: *Graph minors* IV: Widths of *trees* and well-quasi-ordering
m4: *Graph minors: A survey*

Terms	Documents								
	c1	c2	c3	c4	c5	m1	m2	m3	m4
human	1	0	0	1	0	0	0	0	0
interface	1	0	1	0	0	0	0	0	0
computer	1	1	0	0	0	0	0	0	0
user	0	1	1	0	1	0	0	0	0
system	0	1	1	2	0	0	0	0	0
response	0	1	0	0	1	0	0	0	0
time	0	1	0	0	1	0	0	0	0
EPS	0	0	1	1	0	0	0	0	0
survey	0	1	0	0	0	0	0	0	1
trees	0	0	0	0	0	1	1	1	0
graph	0	0	0	0	0	0	1	1	1
minors	0	0	0	0	0	0	0	1	1

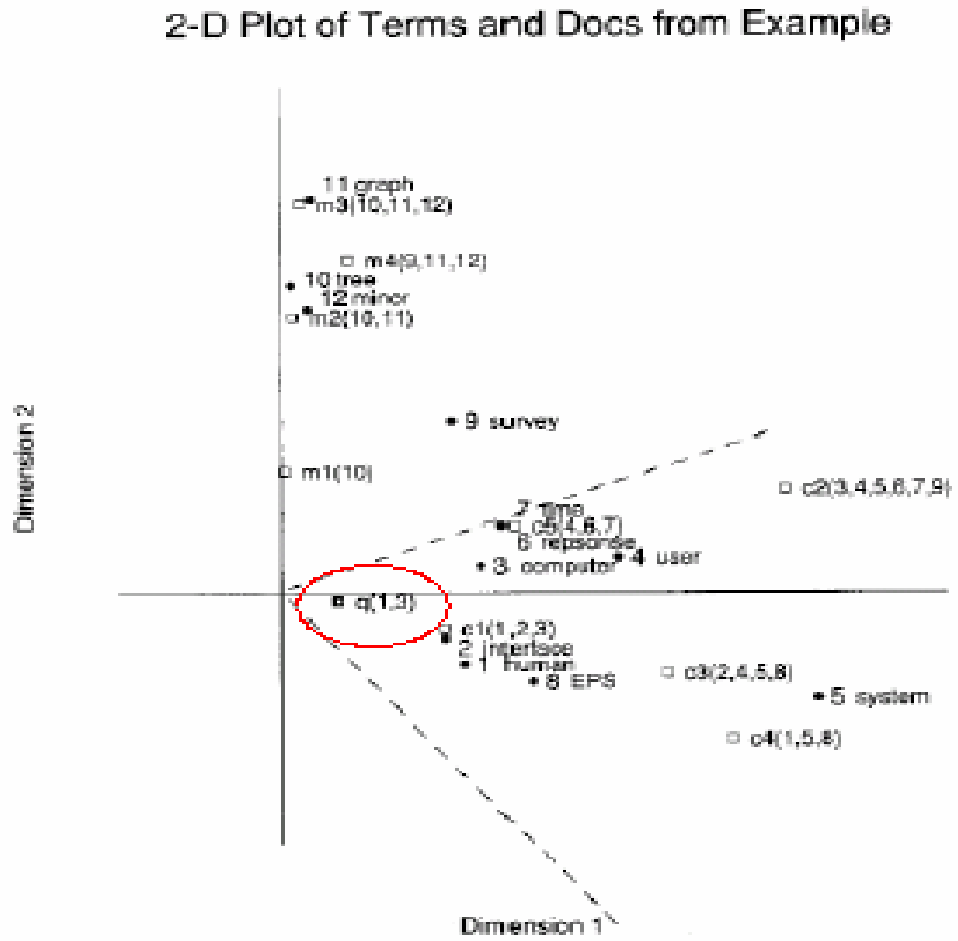


FIG. 1. A two-dimensional plot of 12 Terms and 9 Documents from the sample TM set. Terms are represented by filled circles. Documents are as open squares, and component terms are indicated parenthetically. The query ("human computer interaction") is represented as a pseudo-docu point q. Axes are scaled for Document-Document or Term-Term comparisons. The dotted cone represents the region whose points are within a c .9 from the query q. All documents about human-computer (c1–c5) are "near" the query (i.e., within this cone), but none of the graph theory do (m1–m4) are nearby. In this reduced space, even documents c3 and c5 which share no terms with the query are near it.

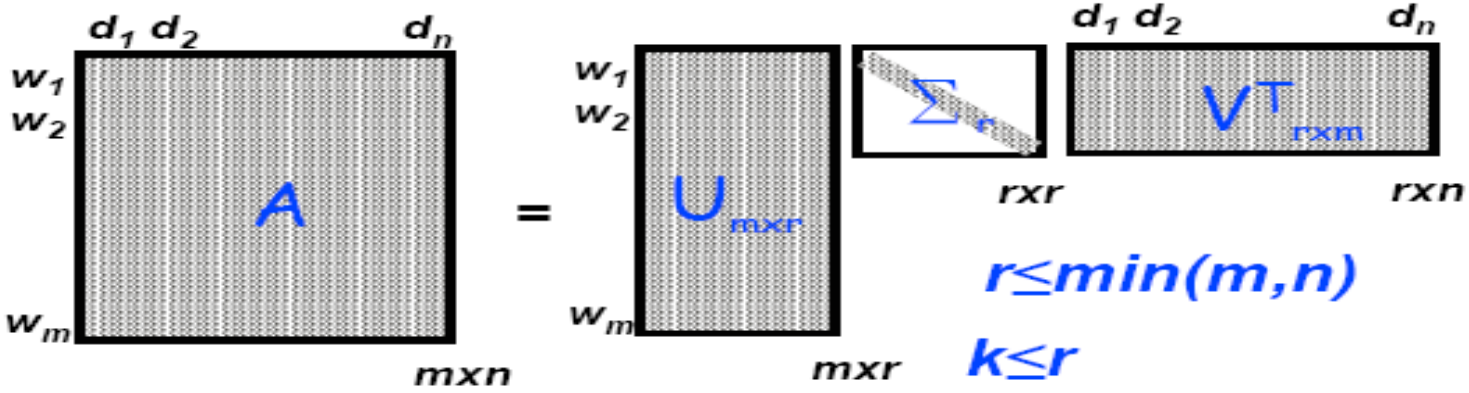
LSI, SVD, & Eigenvectors

- SVD decomposes:
  - Term x Document matrix X as
    - $X=U\Sigma V^T$
      - Where U,V left and right singular vector matrices, and
      - Σ is a diagonal matrix of singular values
    - Corresponds to eigenvector-eigenvalue decompostion: $Y=VLV^T$
      - Where V is orthonormal and L is diagonal
    - U: matrix of eigenvectors of $Y=XX^T$
    - V: matrix of eigenvectors of $Y=X^TX$
    - Σ : diagonal matrix L of eigenvalues

$$XX^T = (U\Sigma V^T)(U\Sigma V^T)^T = (U\Sigma V^T)(V^{T^T}\Sigma^T U^T) = U\Sigma V^T V\Sigma^T U^T = U\Sigma\Sigma^T U^T$$
$$X^TX = (U\Sigma V^T)^T(U\Sigma V^T) = (V^{T^T}\Sigma^T U^T)(U\Sigma V^T) = V\Sigma U^T U\Sigma V^T = V\Sigma^T\Sigma V^T$$

SVD: Dimensionality Reduction

- Singular Value Decomposition (SVD)



Both U and V has orthonormal column vectors

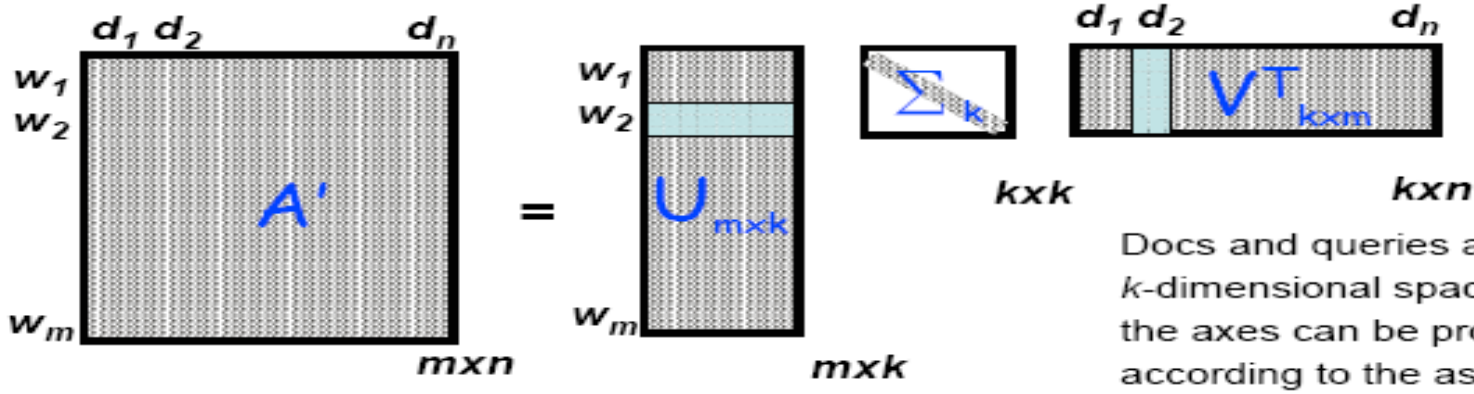
$U^TU=I_{kXk}$

$V^TV=I_{kXk}$

$r \leq min(m,n)$

$k \leq r$

$||A||_F^2 \geq ||A'||_F^2$



Docs and queries are represented in a *k*-dimensional space. The quantities of the axes can be properly weighted according to the associated diagonal values of Σ_k

降维的意义

- 减少特征数量，实现正交化，降低数据冗余度
  - 某种意义上减少了随机噪声（如果有的话）
- 得到一个低秩的特征空间表达
  - 在低秩空间中挖掘了同类样本的相关性（协同过滤）
    - 由向量压缩模型恢复到原空间后获得特征平滑

作业稍晚布置