Python与数据科学导论-11

—— 数据分析示例、有监督学习、贝叶斯分类器

胡俊峰 北京大学

2023/04/10

用户观影数据分析示例（评分表）：

```
ratings[int(1e6):int(1e6+10)]
```

	user_id	movie_id	rating	timestamp
1000000	6040	3552	2	956715942
1000001	6040	1952	5	957717017
1000002	6040	1954	3	960972782
1000003	6040	25	3	957717322
1000004	6040	348	2	956704972
1000005	6040	29	4	960972720
1000006	6040	1960	4	956715597
1000007	6040	1961	4	956703977
1000008	6040	1962	3	956715569
1000009	6040	1963	4	960972887

```
ratings_user_stats2 = ratings.groupby('user_id').agg({'rating':'mean','movie_id':'co
ratings_user_stats2
```

user_id		
1	4.188679	53
2	3.713178	129
3	3.901961	51
4	4.190476	21
5	3.146465	198
...	...	...
6036	3.302928	888
6037	3.717822	202
6038	3.800000	20
6039	3.878049	123
6040	3.577713	341

6040 rows × 2 columns

用户 评分均值-观影数

评分-观影数综合分布

```
ratings_user_stats2.rename(columns={'rating':'rating_average','movie_id':'movie_count'},
ratings_user_stats2.describe()
```

	rating_average	movie_count
count	6040.000000	6040.000000
mean	3.702705	165.597517
std	0.429622	192.747029
min	1.015385	20.000000
25%	3.444444	44.000000
50%	3.735294	96.000000
75%	4.000000	208.000000
max	4.962963	2314.000000

```python
ratings_user_stats2[ratings_user_stats2['movie_count'] > 1300] # 标准化？
```

user_id	rating_average	movie_count
889	2.840580	1518
1150	2.590630	1302
1181	2.815911	1521
1680	3.555676	1850
1941	3.054545	1595
2063	2.945578	1323
3618	3.008185	1344
4169	3.551858	2314
4277	4.134825	1743

超级用户观察

```
In [100]:
ratings_movie_statis = ratings.groupby('movie_id').agg({'rating':'mean','user_id':'count'})
ratings_movie_statis.rename(columns={'rating':'rating_average','user_id':'user_count'}, inp
ratings_movie_statis    # reset_index()?
```

Out[100]:

	rating_average	user_count
movie_id		
1	4.146846	2077
2	3.201141	701
3	3.016736	478
4	2.729412	170
5	3.006757	296
...	...	...
3948	3.635731	862
3949	4.115132	304
3950	3.666667	54
3951	3.900000	40

电影角度的观察

打分与观影分布特征：

```
ratings_movie_statis.describe() # 分数直接用预测量? + kalman
```

	rating_average	user_count
count	3706.000000	3706.000000
mean	3.238892	269.889099
std	0.672925	384.047838
min	1.000000	1.000000
25%	2.822705	33.000000
50%	3.331546	123.500000
75%	3.740741	350.000000
max	5.000000	3428.000000

Junk-pop movies：

```
pop_junk_moives = ratings_movie_statis[(ratings_movie_statis['rating_average'] < 3) &
pop_junk_moives
```

movie_id	rating_average	user_count
788	2.995781	948
1377	2.976722	1031
1391	2.900372	1074
2054	2.933014	1045
2369	2.954762	840
2701	2.158537	902
3354	2.595208	793

Golden-pop movies：

```
pop_golden_moives = ratings_movie_statis[(ratings_movie_statis['rating_average'] > 4.45) & (rating
pop_golden_moives
```

movie_id	rating_average	user_count
50	4.517106	1783
260	4.453694	2991
318	4.554558	2227
527	4.510417	2304
858	4.524966	2223
904	4.476190	1050

看一眼是那些电影：Join操作不work

```
junk_gener_distri = pop_junk_moives.join(movies,on = 'movie_id',how = 'left' ) # index 
junk_gener_distri
```

	rating_average	user_count	movie_id	title	genres
movie_id					
788	2.995781	948	798	Daylight (1996)	Action|Adventure|Thriller
1377	2.976722	1031	1398	In Love and War (1996)	Romance|War
1391	2.900372	1074	1414	Mother (1996)	Comedy
2054	2.933014	1045	2123	All Dogs Go to Heaven (1989)	Animation|Children's
2369	2.954762	840	2438	Outside Ozona (1998)	Drama|Thriller
2701	2.158537	902	2770	Bowfinger (1999)	Comedy
3354	2.595208	793	3423	School Daze (1988)	Drama

```
junk_gener_distri =  pop_junk_moives.merge(movies, left_on = 'movie_id', right_on ='movie_id')
junk_gener_distri
```

	movie_id	rating_average	user_count	title	genres
0	788	2.995781	948	Nutty Professor, The (1996)	Comedy|Fantasy|Romance|Sci-Fi
1	1377	2.976722	1031	Batman Returns (1992)	Action|Adventure|Comedy|Crime
2	1391	2.900372	1074	Mars Attacks! (1996)	Action|Comedy|Sci-Fi|War
3	2054	2.933014	1045	Honey, I Shrunk the Kids (1989)	Adventure|Children's|Comedy|Fantasy|Sci-Fi
4	2369	2.954762	840	Desperately Seeking Susan (1985)	Comedy|Romance
5	2701	2.158537	902	Wild Wild West (1999)	Action|Sci-Fi|Western
6	3354	2.595208	793	Mission to Mars (2000)	Sci-Fi

```
golden_gener_distri =  pop_golden_moives.merge(movies, left_on = 'movie_id', right_on ='movie_id')
golden_gener_distri
```

	movie_id	rating_average	user_count	title	genres
0	50	4.517106	1783	Usual Suspects, The (1995)	Crime|Thriller
1	260	4.453694	2991	Star Wars: Episode IV - A New Hope (1977)	Action|Adventure|Fantasy|Sci-Fi
2	318	4.554558	2227	Shawshank Redemption, The (1994)	Drama

```
users
```

用户信息表

	user_id	gender	zipcode	age_desc	occ_desc
0	1	F	48067	Under 18	K-12 student
1	2	M	70072	56+	self-employed
2	3	M	55117	25-34	scientist
3	4	M	02460	45-49	executive/managerial
4	5	M	55455	25-34	writer
...	...	...	...	...	...
6035	6036	F	32603	25-34	scientist
6036	6037	F	76006	45-49	academic/educator
6037	6038	F	14706	56+	academic/educator
6038	6039	F	01060	45-49	other or not specified
6039	6040	M	11106	25-34	doctor/health care

6040 rows × 5 columns

In [118]:

```
1 pop_junk_moives2
```

Out[118]:

	rating_average	user_count
movie_id		
95	2.876176	638
153	2.642214	777
160	2.238938	565
173	2.308511	564
185	2.869947	569
196	2.823636	550
208	2.631336	651
435	2.606004	533
442	2.992188	640
466	2.795812	573

```
(ratings_movie_statis['user_count']>500)]
```

```python
junk_moive_users = ratings.merge(pop_junk_moives2, left_on ='movie_id', right_on = 'movie_id',
junk_moive_users
```

	user_id	movie_id	rating	timestamp	rating_average	user_count
0	2	3257	3	978300073	2.859425	626
1	8	3257	3	978247143	2.859425	626
2	18	3257	2	978153771	2.859425	626
3	22	3257	1	978136958	2.859425	626
4	26	3257	4	978139867	2.859425	626
...	...	...	...	...	...	...
25007	5636	3697	3	959053960	2.867807	643
25008	5682	3697	4	959043421	2.867807	643
25009	5714	3697	3	959223406	2.867807	643
25010	5767	3697	3	959620056	2.867807	643
25011	5831	3697	2	1024075950	2.867807	643

获得movie_ID

链接用户信息

```
junk_moive_users_distri = junk_moive_users.merge(users, left_on ='user_id', right_on = 'user_id', how = 'left')
junk_moive_users_distri
```

	user_id	movie_id	rating	timestamp	rating_average	user_count	gender	zipcode	age_desc	occ_desc
0	2	3257	3	978300073	2.859425	626	M	70072	56+	self-employed
1	8	3257	3	978247143	2.859425	626	M	11413	25-34	programmer
2	18	3257	2	978153771	2.859425	626	F	95825	18-24	clerical/admin
3	22	3257	1	978136958	2.859425	626	M	53706	18-24	scientist
4	26	3257	4	978139867	2.859425	626	M	23112	25-34	executive/managerial
...	...	...	...	...	...	...	...	...	...	...
25007	5636	3697	3	959053960	2.867807	643	M	98102	25-34	executive/managerial
25008	5682	3697	4	959043421	2.867807	643	M	23455-4959	18-24	other or not specified
25009	5714	3697	3	959223406	2.867807	643	M	96753	35-44	artist
25010	5767	3697	3	959620056	2.867807	643	M	75287	25-34	artist
25011	5831	3697	2	1024075950	2.867807	643	M	92120	25-34	academic/educator

```
show_distri_freq = show_distri.merge(user_occ_distri, left_on ='occ_desc', right_on = 'occ_desc', how = 'inner')
show_distri_freq['freq'] = show_distri_freq['movie_id'] / show_distri_freq['user_id']
show_distri_freq.sort_values('freq',inplace = True)
show_distri_freq
```

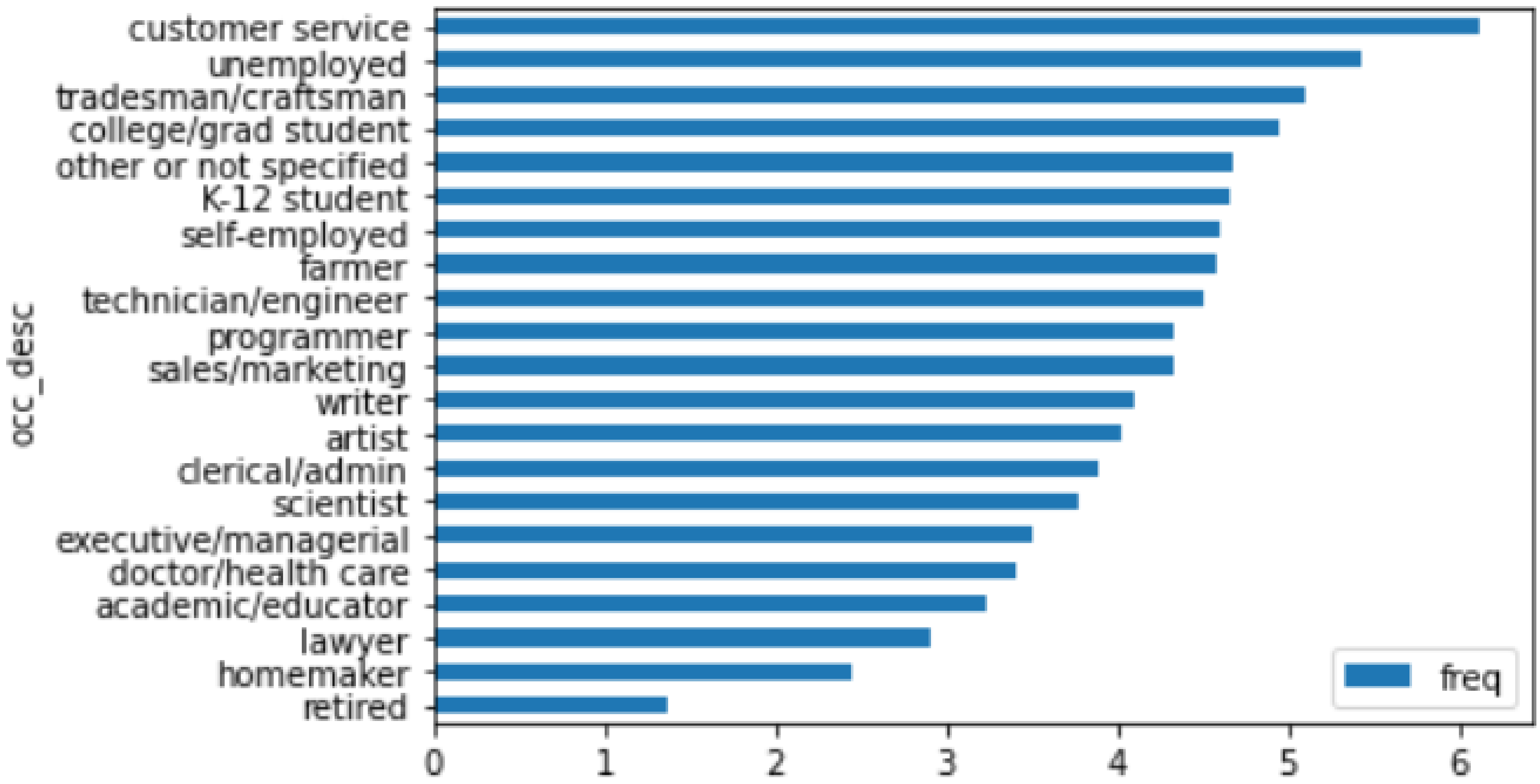
	movie_id	user_id	freq
occ_desc			
retired	194	142	1.366197
homemaker	225	92	2.445652
lawyer	375	129	2.906977
academic/educator	1707	528	3.232955
doctor/health care	804	236	3.406780
executive/managerial	2381	679	3.506627
scientist	543	144	3.770833
clerical/admin	675	173	3.901734
artist	1076	267	4.029963
writer	1151	281	4.096085
sales/marketing	1307	302	4.327815
programmer	1684	388	4.340206
technician/engineer	2267	502	4.515936

Where all the junks gone?

[167]:

```
1 show_distri_freq.plot.barh( y =' freq' )
```

t[167]: <AxesSubplot:ylabel=' occ_desc' >



回顾一下上述数据分析流程：

- 观察数据
- 挖掘有效信息：给出概念化表述、更好的数据认知
- 信息关联与分析
- 进一步给出关联信息分析结果的概念化表述与设想
  - 根据问题优化模型
  - 根据应用场景应用结果

电影推荐系统（问题分析）：

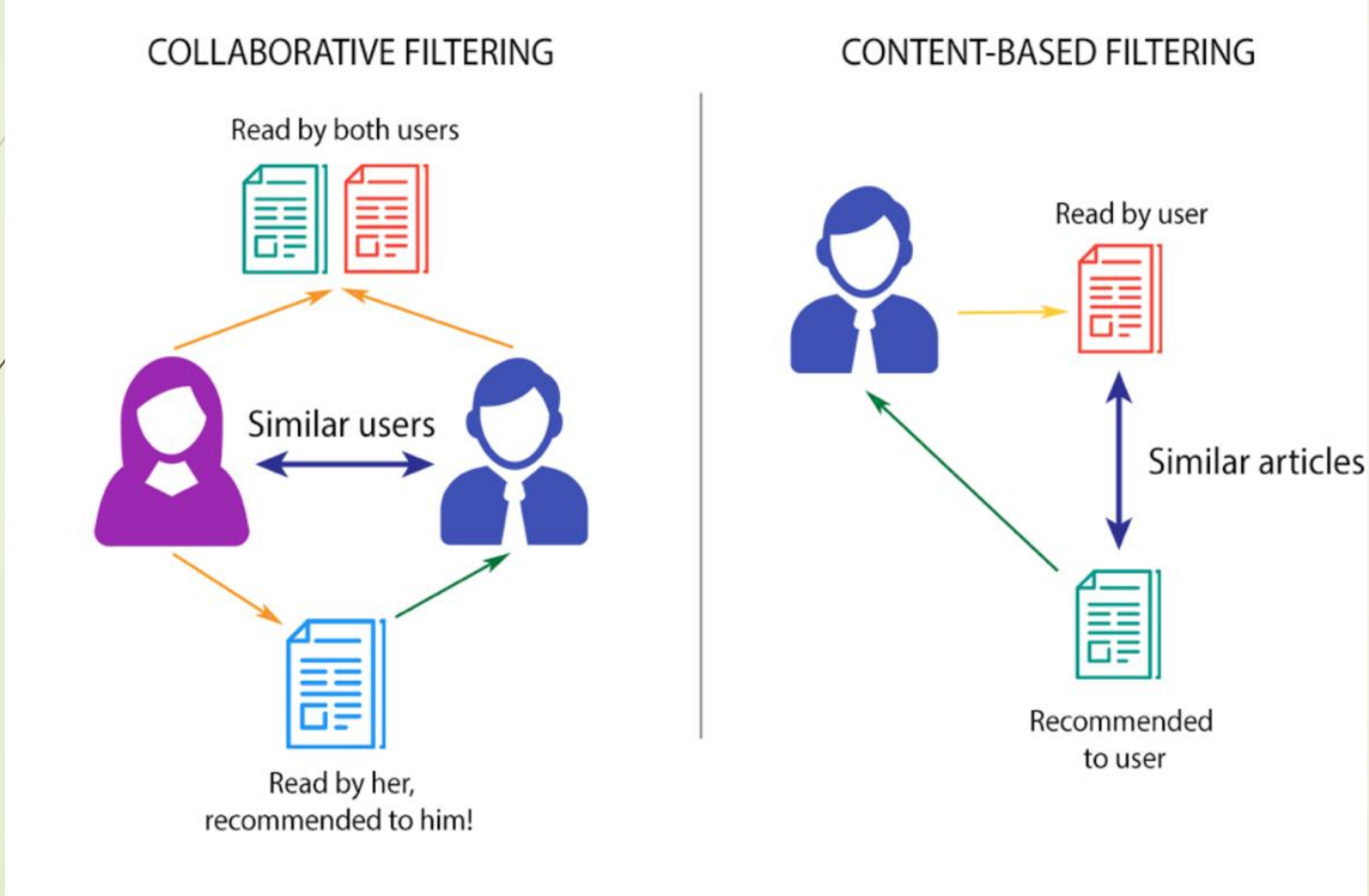
目标：

- 推荐给已有用户更多的候选电影
  - 推荐给新用户可选的电影

条件：

- 已知所有用户历史观影及评分
- 已知用户部年龄、职业、性别信息
- 已知电影名称、类属信息

关联挖掘与推荐模型：



Content-Based Recommendation Model

机器学习——各种距离度量方法

参考：https://blog.csdn.net/qs17809259715/article/details/96110056

一、欧氏距离(EuclideanDistance)
二、曼哈顿距离(ManhattanDistance)
三、切比雪夫距离 (Chebyshev Distance)
四、 闵可夫斯基距离(Minkowski Distance)
五、标准化欧式距离(Standardized Euclidean Distance)
六、马氏距离(Mahalanobis distance)
七、余弦距离(Cosine Distance)
八、汉明距离(Hamming Distance)
九、杰卡德距离(Jaccard Distance)
十、相关距离(Correlation distance)
十一、信息熵(Information Entropy)

九、杰卡德距离(Jaccard Distance)

1.定义

- 杰卡德相似系数(Jaccard similarity coefficient)：两个集合A和B的交集元素在A，B的并集中所占的比例，称为两个集合的杰卡德相似系数，用符号J(A,B)表示：

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

杰卡德相似系数

- 杰卡德距离(Jaccard Distance)：与杰卡德相似系数相反，用两个集合中不同元素占所有元素的比例来衡量两个集合的区分度：

$$J_\delta(A,B) = 1 - J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

杰卡德距离

信息相关度：互信息-点互信息

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) log \frac{p(x,y)}{p(x)p(y)}$$

$$\operatorname{pmi}(x;y) \equiv \log \frac{p(x,y)}{p(x)p(y)} = \log \frac{p(x|y)}{p(x)} = \log \frac{p(y|x)}{p(y)}.$$

作业环境中给出的一个实现：

生成User-Movies二维rating矩阵

- 2. 生成User-Movies二维rating矩阵

```
# 填写数据矩阵
train_data_matrix = np.array(train_data.pivot_table('rating', index='user_id', columns='movie_id', aggfunc='mean').fillna(0))

# 1. todu 填写测试集矩阵
test_data_matrix = np.array(test_data.pivot_table('rating', index='user_id', columns='movie_id', aggfunc='mean').fillna(0))

train_data_matrix[0:3,:] #每行为一个用户的观影评分向量
```

```
array([[5., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

相似度计算：

计算 user-user 相似度矩阵 和 item-item 相似度矩阵

```
from sklearn.metrics.pairwise import pairwise_distances
user_similarity = pairwise_distances(train_data_matrix, metric=’cosine’)    # 用户

# 2, todo: 生成电影相似度矩阵

user_similarity[0]
```

向量空间模型的前提？

```
array([0.        , 0.96598031, 0.94814587, ..., 1.        , 0.80161937,
       0.87312904])
```

通过相似度矩阵进行预测

```python
def predict(train_ratings, similarity, type='user'):
    if type == 'user':
        mean_user_rating = train_ratings.mean(axis=1) # 平均分
        ratings_diff = (train_ratings - mean_user_rating[:, np.newaxis]) # 评分向量中心化，
        # 通过相似度矩阵生成推荐，然后再把中心偏置加回来
        pred = mean_user_rating[:, np.newaxis] + \
            similarity.dot(ratings_diff) / np.array([np.abs(similarity).sum(axis=1)]).T #  user_sim

    elif type == 'item':
        pred = train_ratings.dot(similarity) / np.array([np.abs(similarity).sum(axis=1)]) # item_sim
        # 3、todo:这里尝试对电影的属性做一下中心化，观察结果数据有无改善。能否尝试分析原因

    return pred

item_prediction = predict(train_data_matrix, item_similarity, type='item')
user_prediction = predict(train_data_matrix, user_similarity, type='user')

user_prediction.shape
```

(6040, 3660)

$$RMSE = \sqrt{\frac{1}{N}\sum(x_i - \hat{x}_i)^2}$$

I'll use the scikit-learn's **mean squared error** function as my validation metric. Comparing user- and item-based collaborative filtering, it looks
collaborative filtering gives a better result.

评估预测效果

```python
from sklearn.metrics import mean_squared_error
from math import sqrt

def rmse(prediction, ground_truth):
    prediction = prediction[ground_truth.nonzero()].flatten()  # 推荐结果 与 ground truth比
    ground_truth = ground_truth[ground_truth.nonzero()].flatten()
    errorR = sqrt(mean_squared_error(prediction, ground_truth))

    return errorR

print('User-based CF RMSE: ' + str(rmse(user_prediction, test_data_matrix)))
print('Item-based CF RMSE: ' + str(rmse(item_prediction, test_data_matrix)))

# 4, todu:目前这种评测方案有没有问题？能否设计一种更好的评价方案（简单说明思路和道理+实现）。选做题。
# 能完成的同学可以在作业压缩包学号后面添加一个#号以提醒助教优先评阅
```

```
User-based CF RMSE: 3.6224523289261183
Item-based CF RMSE: 3.6385367743842383
```

- RMSE对极端值敏感，更换MAE（平均绝对误差）
- F1-score评价推荐情况

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

1900012973 王乐安

2000013157 刘知一

二、电影推荐

- 看前k个预测准确几个
- 可以把预测和label都排个序然后看对应位置有几个hit
- 或者我们可以更复杂一点，可以加入一些匹配的元素，如果离某个位置偏离不超过m我们就算预测正确，同样可以排序然后遍历实现

2000013064 张泽楷（没有代码）

问题1-n：

- 观察值为NULL <> 观察值为0
- 评分高低 与 推荐-不推荐
- 推荐电影结果的loss的计算方案
  - 评测数据的设计

第一步尝试：将推荐问题设定为独立问题，用贝叶斯模型进行topk 推荐。计算准确率、召回率

数据科学的基本原则

- 通过数据认识世界
- 借助模型定义方案

机器学习

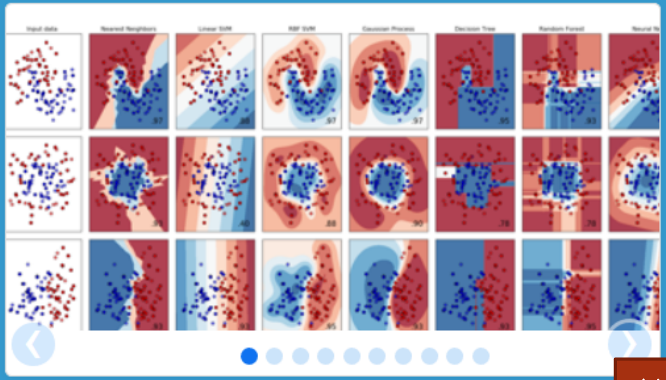
- 假定存在一个预测函数 y = F（x）
  - 输入为一组特征
  - 输出为一个预测空间的概率分布
- 函数F称为 **模型（model）**
- 通过训练集学习得到模型的参数称为 **回归 （regression，fit）**
- 根据特征计算输出概率分布或返回最大概率解称为 **预测（prediction）**

分类 与 回归（机器学习）

- 对于一个样本集，如果能找到一个合理的分类函数，使得：
  F(X) ≈ 1 (当Y = 1)； F(X) ≈ 0 (当Y = 0)

- 则可以称 我们找到了一个原样本集的一个‘似然’ 函数。
- 如果F是以最大概率符合样本数据，则F称为最大似然函数。

常用的机器学习软件包



scikit-learn

Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

模型参数回归

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.
Algorithms: SVM, nearest neighbors, random forest, …

— Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, ridge regression, Lasso, …

— Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, …

— Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: PCA, feature selection, non-negative matrix factorization.

— Examples

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning
Modules: grid search, cross validation, metrics.

— Examples

Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.
Modules: preprocessing, feature extraction.

— Examples

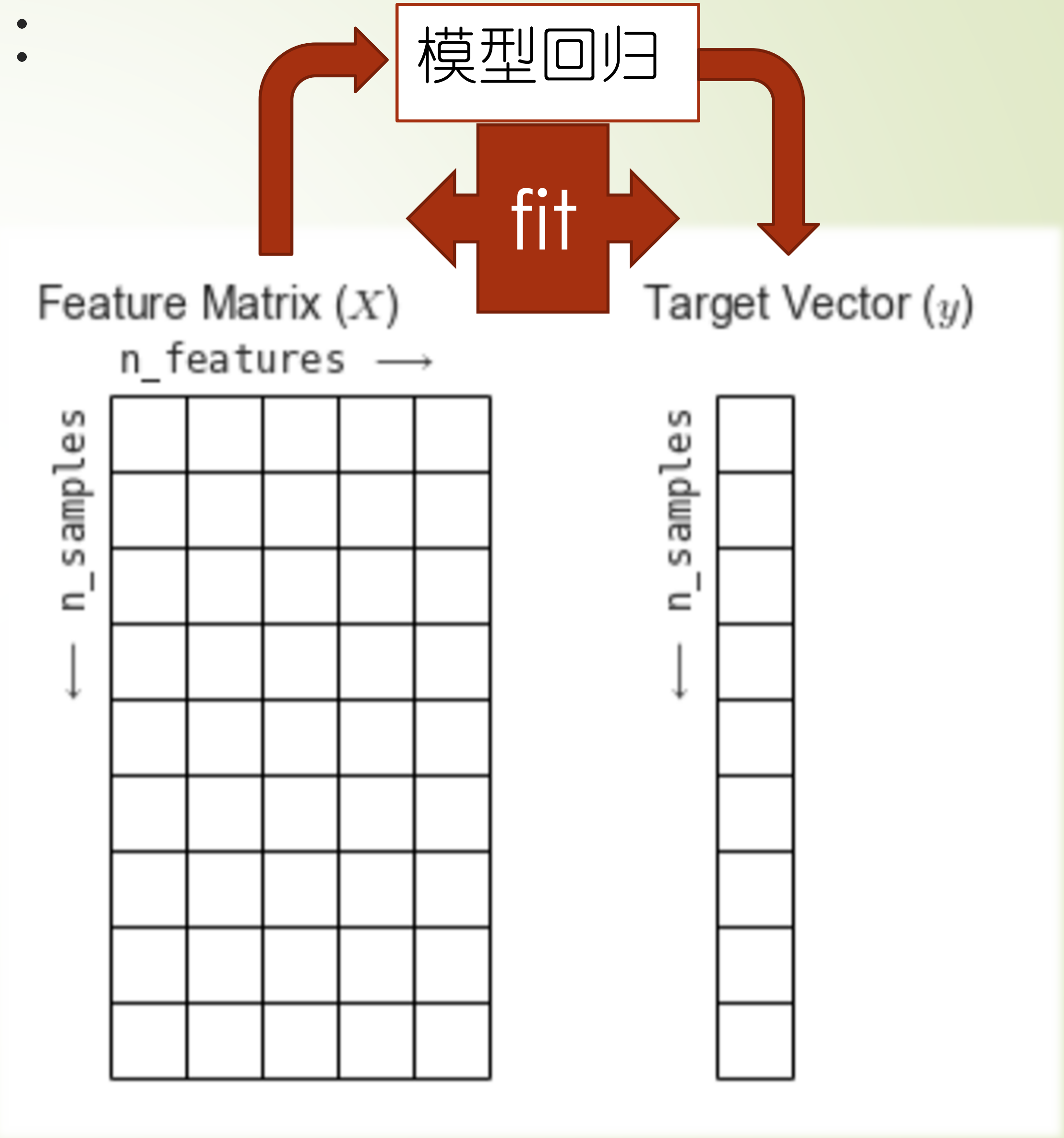
训练数据一般组织形式：

数据集一般包括：

训练集：用于训练模型

验证集：用于调整模型参数

测试集：用于评测模型效果



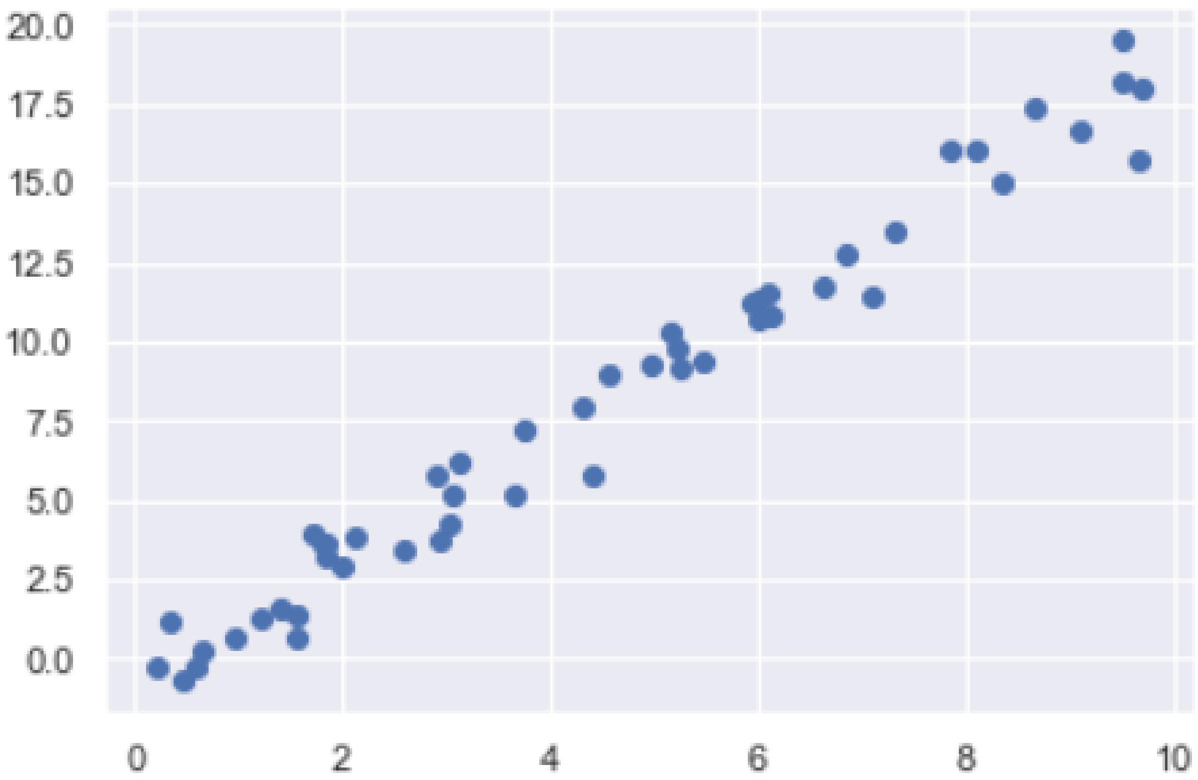
看一个线性回归的例子 模型： y = ax + b

```python
import matplotlib.pyplot as plt
import numpy as np

rng = np.random.RandomState(42) # 设置伪随机数种子
x = 10 * rng.rand(50)
y = 2 * x - 1 + rng.randn(50) # y = 2 * x - 1 + 0-1随机
plt.scatter(x, y);
```

Numpy 生成随机数据

Pyplot画图



线性回归（参数回归与线性拟合）

```
from sklearn.linear_model import LinearRegression
model = LinearRegression(fit_intercept=True)
print(model)
```

选择模型

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

```
X = x[:, np.newaxis]
X.shape
```

```
(50, 1)
```

```
model.fit(X, y)
```

拟合模型参数

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

```
model.coef_ ,model.intercept_
```

观察模型参数-斜率，截距

```
(array([1.9776566]), -0.903310725531164)
```

模型预测

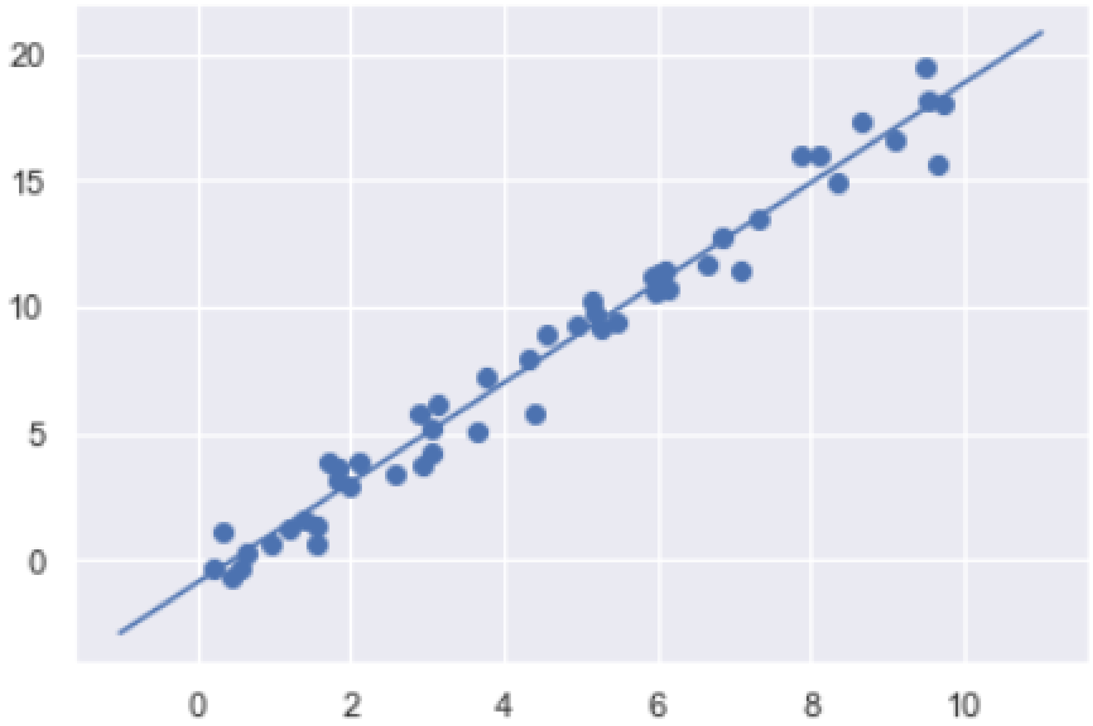
```
Xfit = xfit[:, np.newaxis]
yfit = model.predict(Xfit)
xfit,yfit
```

使用模型进行预测

```
(array([-1.        ,  0.33333333,  1.66666667,  3.        ,  4.33333333,
         5.66666667,  7.        ,  8.33333333,  9.66666667, 11.        ]),
 array([-2.88096733, -0.24409186,  2.39278361,  5.02965908,  7.66653454,
        10.30341001, 12.94028548, 15.57716094, 18.21403641, 20.85091188]))
```

```
plt.scatter(x, y)
plt.plot(xfit, yfit);
```

输出显示模型生成结果



分类与回归

- 对于一个样本集，如果能找到一个合理的分类函数，使得：
  F(X) ≈ 1 (当Y = 1)； F(X) ≈ 0 (当Y = 0)
- 则可以称 我们找到了一个原样本集的一个‘似然’函数。
- 如果F是以最大概率符合样本数据，则F称为最大似然函数。

机器学习的一般步骤

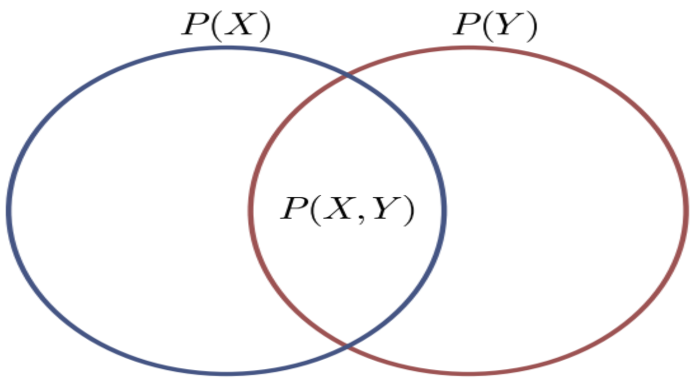
- 数据预处理、特征工程
  - 数据清洗
  - Na平滑，拉普拉斯平滑/降噪
  - 特征降维与均衡（embedding），隐含语义平滑
- 模型选择、超参数设计
- 模型学习与评测（可视化）

贝叶斯分类与有监督学习

- 概率模型
- 朴素贝叶斯分类
- 有监督学习

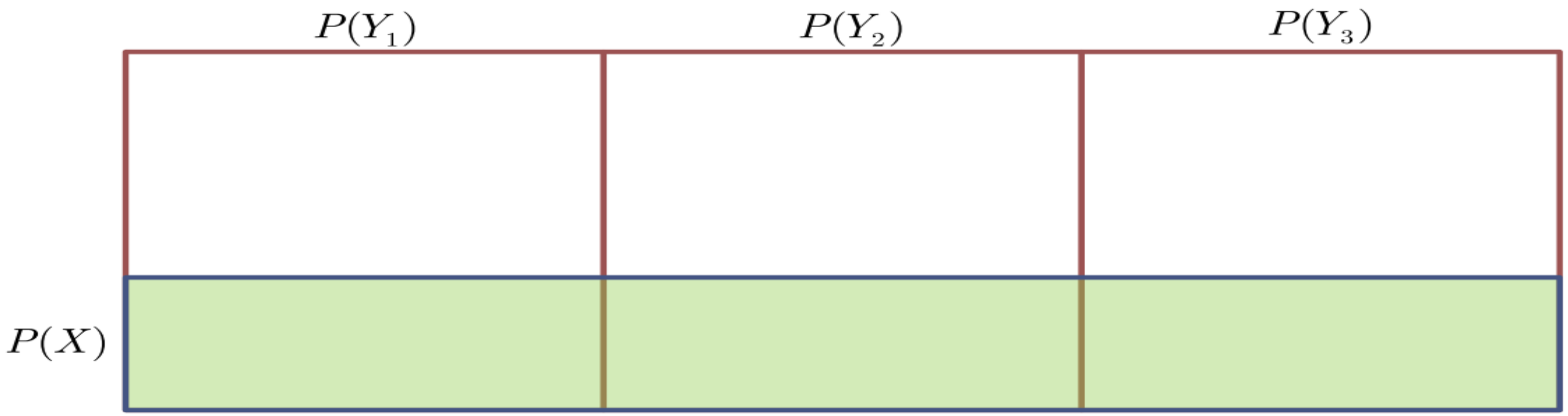
联合概率与条件概率

$$P(X \mid Y) = \frac{P(X,Y)}{P(Y)}$$



全概率公式

$$P(X) = \sum_Y P(X \mid Y)P(Y)$$



贝叶斯公式 与 贝叶斯推断

由：P（y|x）* P（x）= P（x|y）* P（y）

可以导出：

$$P(y|x) = \frac{P(x|y) * P(y)}{P(x)}$$

$$P(Y \mid X) = \frac{P(X \mid Y)P(Y)}{P(X)} = \frac{P(X \mid Y)P(Y)}{\sum_y P(X \mid y)P(y)}$$

贝叶斯概念：

想预测Y在未来特定要素下的表现，

只需了解过往Y情况下要各素的分布

贝叶斯概念：

先验概率乘经过条件似然进行修饰，

得到带约束条件的后验概率

举个例子：

- 如果小A精神好，80%可能会起来跑步。
- 小A如果精神不好，40%可能会起来跑步
- 总体观察小A精神好的概率为60%

$$P(Y \mid X) = \frac{P(X \mid Y)P(Y)}{P(X)} = \frac{P(X \mid Y)P(Y)}{\sum_y P(X \mid y)P(y)}$$

目前看到小A正在跑步（看书，听音乐，打游戏 ...）

问：小A同学此时精神好的概率？

$$P(y1 \mid x) = \frac{P(x \mid y1) * P(y1)}{P(x \mid y1) + P(x \mid y2)} = \frac{0.8 * 0.6}{0.8 * 0.6 + 0.4 * 0.4} = \frac{48}{64}$$

机器学习能做什么？

- 根据历史信息统计分析出 **事件-现象** 之间的概率关系

  —— 学习

- 根据目前观测到的 **现象** 对未发生事件的概率给出判断

  —— 预测

常见的分类问题描述

- 输入x是一个d维特征组成的向量R^d
- 模型F（x）的输出为一个k分类的唯一分类（one hot）的向量

$$\mathbb{R}^d \longrightarrow \{1, \ldots, k\}$$

朴素贝叶斯分类器（Naive Bayes Classifier）

- 基于贝叶斯推断方案： 先验概率 * 似然 → 后验概率
- **假定**特征之间相互独立：

$$P(y \mid x_1, \cdots, x_n) = \frac{P(y)P(x_1, \dots x_n \mid y)}{P(x_1, \cdots, x_n)} \Rightarrow P(y \mid x_1, \cdots, x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \cdots, x_n)}$$

$$P(y \mid x_1, \cdots, x_n) \propto P(y)\prod_{i=1}^{n} P(x_i \mid y)$$

- 模型实际返回值：最大后验概率（MAP）

模型参数估计（以词袋子特征为例）

- 模型所需的参数有 $P(y)$, $P(x_i \mid y)$.
- 最大似然估计:

$$\hat{P}(y) = \frac{|N(y)|}{Total}$$

$$\hat{P}(x_i \mid y) = \frac{n_{x_i,y}}{n_y}$$

- 问题?
  - 概率为 0 的情况. 若类 1 中出现词 x, 类 2 中没有.
  - 则 $P(\mathrm{x}|2) = 0$. 一个含有 x 的词永远无法被分入类 2.
  - 这是我们不希望看到的.

平滑 — 置信度 — 先验分布（伪计数）

平滑 (smoothing)

- 拉普拉斯（+1）平滑:

$$\hat{P}(y) = \frac{|N(y)|}{Total}$$

$$\hat{P}(x_i \mid y) = \frac{n_{x_i,y} + 1}{n_y + |V|}$$

- 带系数:

$$\hat{P}(x_i \mid y) = \frac{n_{x_i,y} + \alpha}{n_y + \alpha|V|}$$

隐含语义平滑？

- 通过矩阵分解与恢复得到平滑后的训练集
- 用平滑后的训练集进行训练

- 理论上属于高斯平滑

分类模型评价指标

- 混淆矩阵

		Actual	
		Positive	Negative
Predicted	Positive	TP	FP
	Negative	FN	TN

recall = TP/(TP + FN)

precision = TP/(TP+FP)

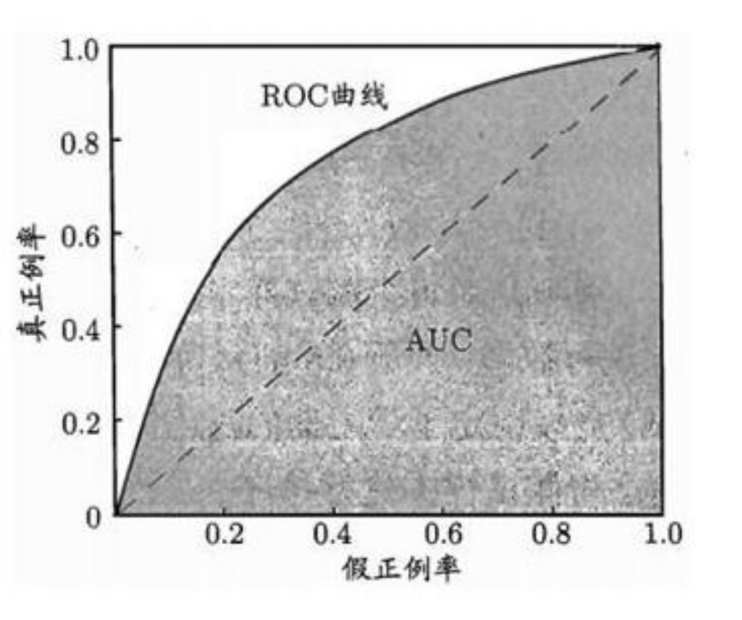
F1=2*recall*precision/(precision+recall)

分类模型评价指标

真正例率TPR=TP/(TP+FN)

假正例率FPR=FP/(TN+FP)

		Actual	
		Positive	Negative
Predicted	Positive	TP	FP
	Negative	FN	TN



文本分类的例子：

```
documents = [(list(movie_reviews.words(fileid)), category)
 for category in movie_reviews.categories()
 for fileid in movie_reviews.fileids(category)]
random.shuffle(documents)
train_set, test_set = featuresets[500:], featuresets[:500]  # 分离训练集、测试集
```

```
def document_features(document):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['contains({})'.format(word)] = (word in
document_words)
    return features  # 词袋子
featuresets = [(document_features(d), c) for (d,c) in documents]  # 词袋子特征，文本类标 集合
train_set, test_set = featuresets[500:], featuresets[:500]       # 分离训练集、测试集
```

```
classifier = nltk.NaiveBayesClassifier.train(train_set)
print(nltk.classify.accuracy(classifier, test_set))
```

查看最有用的特征：

```
>>> classifier.show_most_informative_features(5)
Most Informative Features
        contains(segal) = True              neg : pos    =     11.3 : 1.0
   contains(outstanding) = True              pos : neg    =      8.6 : 1.0
         contains(wasted) = True              neg : pos    =      7.3 : 1.0
          contains(mulan) = True              pos : neg    =      7.2 : 1.0
   contains(wonderfully) = True              pos : neg    =      6.3 : 1.0
```

针对连续特征的 GAUSSIAN NAIVE BAYES

- 特征是实数量
- 服从高斯分布
- 假设特征之间独立

看一个鸢尾花数据集：

```python
import seaborn as sns
iris = sns.load_dataset('iris')
print(iris.head(n = 3))
ir = iris.groupby('species')
ir.head(n = 2)
```

4个特征：花萼长宽，花瓣长宽

种属

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
50	7.0	3.2	4.7	1.4	versicolor
51	6.4	3.2	4.5	1.5	versicolor
100	6.3	3.3	6.0	2.5	virginica
101	5.8	2.7	5.1	1.9	virginica

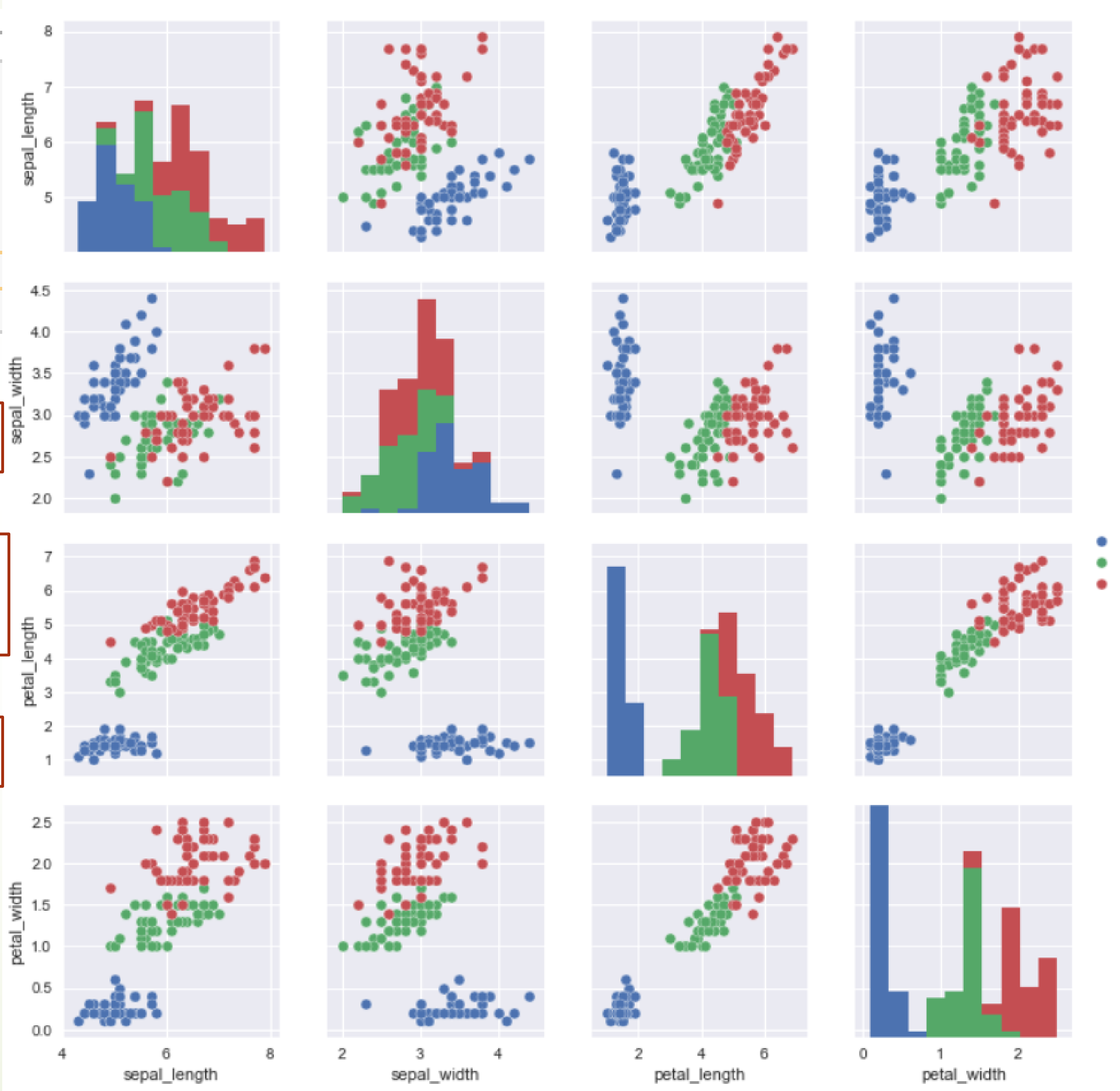
鸢尾花特征高维数据可视化（数据维度两两组合）

```
%matplotlib inline
import seaborn as sns # ; sns.set
sns.pairplot(iris, hue='species',
```

对角线元素显示其他三个维度的取值在当前维度下的分布

非角线元素显示当前维度与另一个维度展开的二维平面上样本数据的分布情况

特征之间并不独立，每个特征在数据集上的分布也不均匀



Classification with Gaussian Naïve Bayes

- Possibility one: Disregard correlation —> Naïve
  - For each feature:
    - Calculate sample mean μ and sample standard deviation σ
    - Use these as estimators of the population mean and deviation
  - For a given feature value *x*, calculate the probability density assuming that x is in a category c
    - $P(x \mid c) \sim \mathcal{N}(\mu_c, \sigma_c)$

https://tschwarz.mscs.mu.edu/Classes/PDSF20/Modules/Bayes/presentation.pdf

Classification with Gaussian Naïve Bayes

- Estimate the probability for observation $(x_1, x_2, \ldots, x_n)$ as the product of the densities

$$P((x_1, \ldots, x_n) \,|\, c_j) \sim \mathcal{N}(x_1, \sigma_{1,c_j}, \mu_{1,c_j}) \cdot \ldots \cdot \mathcal{N}(x_1, \sigma_{n,c_j}, \mu_{1,c_j})$$

x_n

- Then use Bayes formula to invert the conditional probabilities
  - This means estimating the prevalence of the categories
  - $$P(c_j \,|\, (x_1, \ldots, x_n)) = \frac{P((x_1, \ldots, x_n) \,|\, c_j) P(c_j)}{P((x_1, \ldots, x_n))}$$

手动实现一个G-NB：

```python
class Gaussian(object):

    def __init__(self):
        # 这里可以设置平滑或先验参数
        pass

    def fit(self,X_train,Y_train):
        self._data_with_label = X_train.copy()
        self._Y_train = Y_train.copy()
        self._data_with_label['label'] = Y_train[0] # 有监督数据

        self._mean_mat = self._data_with_label.groupby("label").mean() # 每个类别的特征分布 均值
        self._var_mat = self._data_with_label.groupby("label").var()    # 方差
        self.prior_rate = self.__Priori() # 统计类别先验
        return self

    #Priori probability
    def __Priori(self):
        labels = self._Y_train[0].value_counts().sort_index()      # label计数
        prior_rate = np.array([ i /sum(labels) for i in labels])  # label比例
        return prior_rate
```

```python
#Priori probability
def __Priori(self):
    labels = self._Y_train[0].value_counts().sort_index()     # label计数
    prior_rate = np.array([ i /sum(labels) for i in labels])  # label比例
    return prior_rate

def predict(self,X_test): # 模型预测
    pred = [self.__Condition_formula(self.mean_mat,self.var_mat,row ) * self.prior_rate for row in X_test.values
    class_result = np.argmax(pred, axis=1)  # 返回 argmax
    return class_result

#Gaussian Bayes condition formula
def __Condition_formula(self,mu,sigma2,row):
    P_mat = 1/np.sqrt(2*math.pi*sigma2) * np.exp(-(row-mu)**2/(2*sigma2)) # 高斯函数计算先验
    P_mat = pd.DataFrame(P_mat).prod(axis=1) # 返回一列的乘积
    return P_mat
```

```
# 导入数据集测试一下:
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

iris = load_iris()
iris.target = pd.DataFrame(iris.target)
iris.data = pd.DataFrame(iris.data)

#  train_size, test_size None, it will be 0.25 by default
X_train, X_test, Y_train, Y_test = train_test_split(iris.data, iris.target, test_size=0.4, random_state=1)

np.set_printoptions(suppress=True) # 不用科学计数法的形式输出
```

```
X_train
```

	0	1	2	3
11	4.8	3.4	1.6	0.2
113	5.7	2.5	5.0	2.0
123	6.3	2.7	4.9	1.8
12	4.8	3.0	1.4	0.1
2	4.7	3.2	1.3	0.2

对比直接调包的效果：

```
NB = Gaussian() # 使用自定义类
NB.fit(X_train, y_train)
y_train_NB = NB.predict(X_train)
y_test_NB = NB.predict(X_test)
print("Use custom Gaussian Naive Bayes algorithm\naccuracy on train
    accuracy_score(y_test,y_test_NB))

print("--- %s seconds ---" % (time.time() - start_time))
```

```
Use custom Gaussian Naive Bayes algorithm
accuracy on train set:  0.9555555555555556
accuracy on test set:  0.95
--- 0.2263889312744140б seconds ---
```

```
from sklearn.naive_bayes import GaussianNB
start_time = time.time()
NB2 = GaussianNB()
NB2.fit(X_train, y_train.values.ravel())
y_train_NB2 = NB2.predict(X_train)
y_test_NB2 = NB2.predict(X_test)
print("Use sklearn Gaussian Naive Bayes algorithm\naccuracy on train set:
    accuracy_score(y_test,y_test_NB2))

print("--- %s seconds ---" % (time.time() - start_time))
```

```
Use sklearn Gaussian Naive Bayes algorithm
accuracy on train set:  0.9555555555555556
accuracy on test set:  0.95
--- 0.004966259002685547 seconds ---
```

显示混淆矩阵

```
import matplotlib.pyplot as plt
import seaborn as sns
con_matrix = pd.crosstab(pd.Series(y_test.values.flatten(), name='Actual' ),pd.Series(y_test_NB, name='Predicted'))
plt.title("Test set Confusion Matrix on Gaussian Naive Bayes")
sns.heatmap(con_matrix, cmap="Blues", annot=True, fmt='g')
plt.show()
```