第五章 文本聚类和主题检测

文本聚类

1.1 聚类问题

1.2 文本相似度和聚类方法

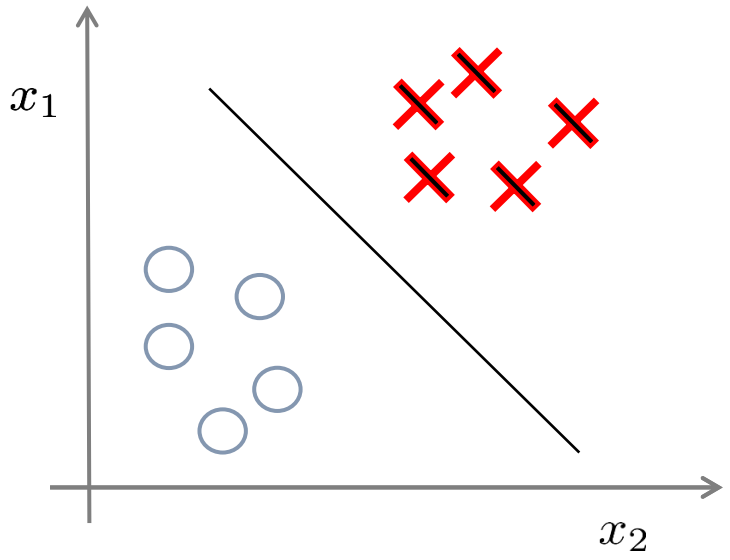
1.3 案例实现

监督学习 vs 无监督学习

分类：分类模型训练需要标记好的数据，是监督学习。

$\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots (\boldsymbol{x}_n, y_n)\}$

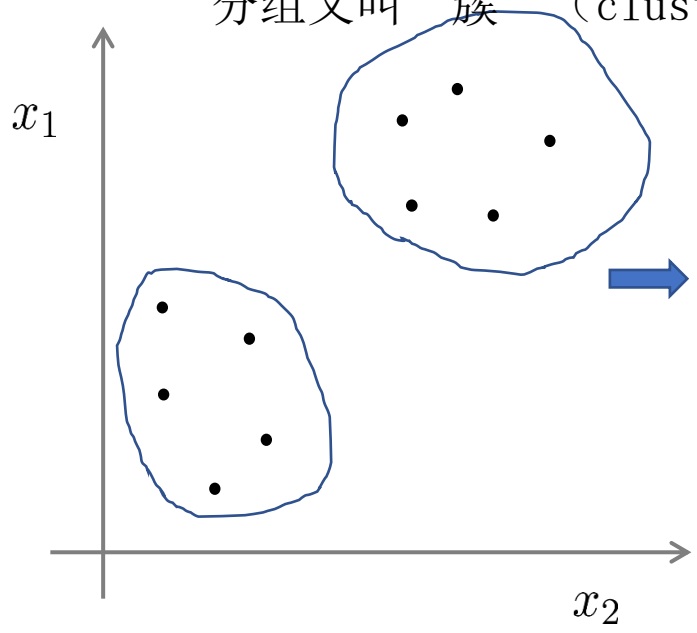
标签哪里来？人工给出，成本高。



$\{\boldsymbol{x}_1, \boldsymbol{x}_2 \cdots \boldsymbol{x}_n\}$

没有标签我们能做什么分析？

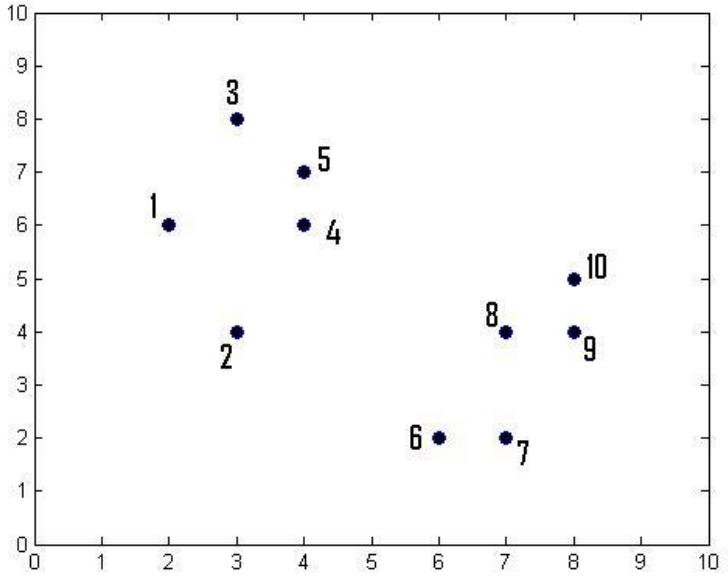
分组又叫“簇”（cluster）



聚类分析：最常用的无监督学习方法。

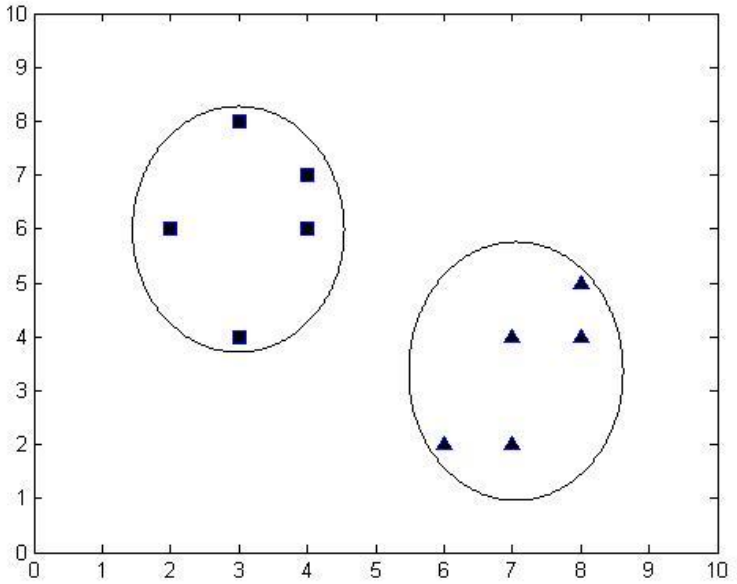
划分聚类

划分聚类（partitioning clustering）一般得到若干个互斥的簇。



待聚类的数据集χ

划分聚类



上图：

$\chi = C_1 \cup C_2$，且 $C_1 \cap C_2 = \emptyset$；

每个样本属于且只属于一个簇；

K-均值算法

对每个样本所属簇和每个簇中心的迭代更新。

随机初始化K个簇中心点 $\mu_1, \mu_2, \cdots \mu_K \in \mathbb{R}^s$

重复以下两步 {

//更新每个样本所属的簇

for i = 1 to n

$C_i \leftarrow$ K个中心点中与x_i最近的那个簇

//以均值作为簇中心进行更新

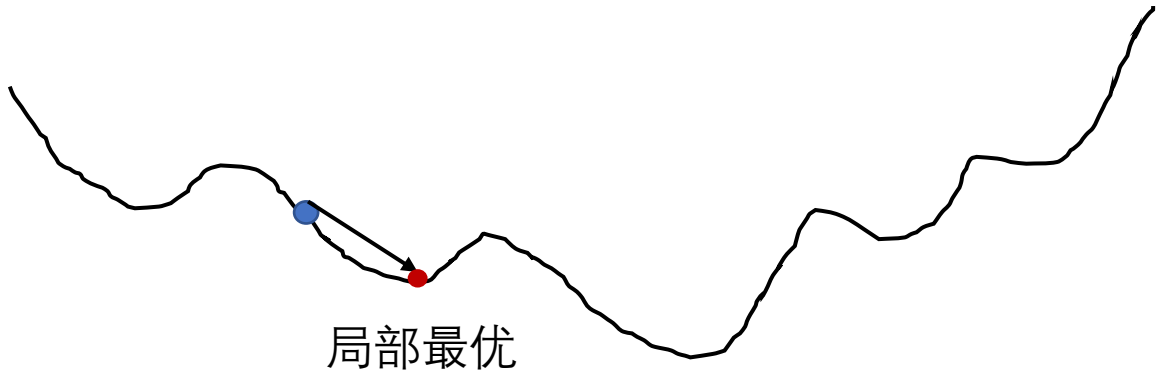
for k = 1 to K

$\mu_k \leftarrow$ 所有被标记到第k个簇的样本的均值

}直到分组不再变化或达到最大迭代次数

参数n, s, K分别表示样本集的大小，维度，以及簇的个数。

初始化对结果的影响



原因：通过前面迭代算法只能得到目标函数的局部最优。

针对随机初始化的改进措施：

- 随机选择k个训练样本作为k个簇的初始中心点；
- 多次基于随机初始化运行，选择目标函数值最小的结果。

样例文本层次聚类

```
corpus = ['The sky is blue and beautiful.',
          'Love this blue and beautiful sky!',
          'The quick brown fox jumps over the lazy dog.',
          "A king's breakfast has sausages, ham, bacon, eggs, toast and beans",
          'I love green eggs, ham, sausages and bacon!',
          'The brown fox is quick and the blue dog is lazy!',
          'The sky is very blue and the sky is very beautiful today',
          'The dog is lazy but the brown fox is quick!' ]
labels = ['weather', 'weather', 'animals', 'food', 'food', 'animals', 'weather', 'ani
```

特征表示（这里没有进行任何预处理）

```
from sklearn.feature_extraction.text import TfidfVectorizer
tv = TfidfVectorizer(min_df=0., max_df=1., norm='l2', use_idf=True, smooth_idf=True
tv_matrix = tv.fit_transform(corpus)
tv_matrix = tv_matrix.toarray()
```

KMeans()

➢属性

- cluster_centers_：k*d 矩阵，表示k个簇中心的坐标
- labels_：每个样本对应的簇编号
- inertia_：$\sum_{c=1}^{k}\sum_{x_i\in\delta_c}\|x_i-\delta_c\|^2$

➢方法

- fit(X[, y, sample_weight]) ： 进行kmeans聚类
- predict(X[, sample_weight]): 得到离样本最近的簇

归一化互信息NMI计算：

NMI_score= metrics.normalized_mutual_info_score(labels_true, labels)
其中labels_true为真实分组，labels为聚类得到的簇.

k均值聚类结果

```
#k均值聚类
from sklearn.cluster import KMeans
km = KMeans(n_clusters=3, max_iter=100, n_init=50, random_state=42).fit(tv_matrix)
```

n_clusters: 簇的数目， n_init：随机初始化后跑的次数，最终取最好的一次

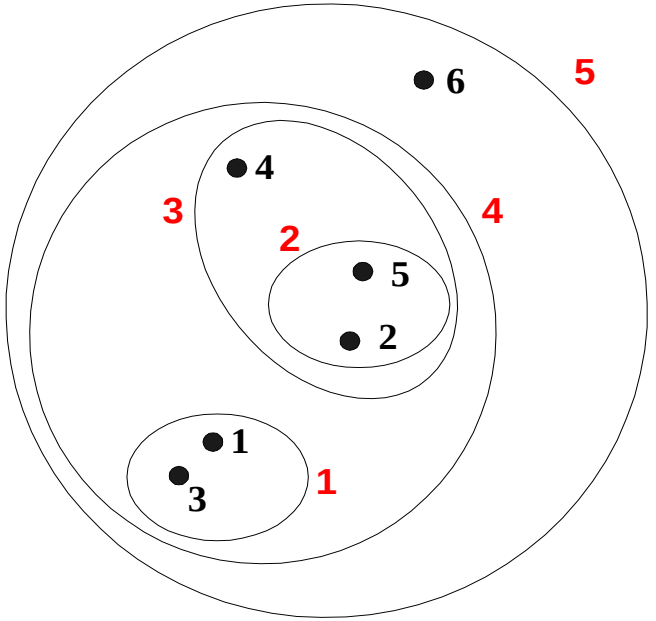
```
#k均值聚类结果
from collections import Counter
import numpy as np
corpus = np.array(corpus)
corpus_df = pd.DataFrame({'Document': corpus, 'Category': labels})
corpus_df['ClusterLabel']=km.labels
Counter(km.labels_)
corpus_df
```

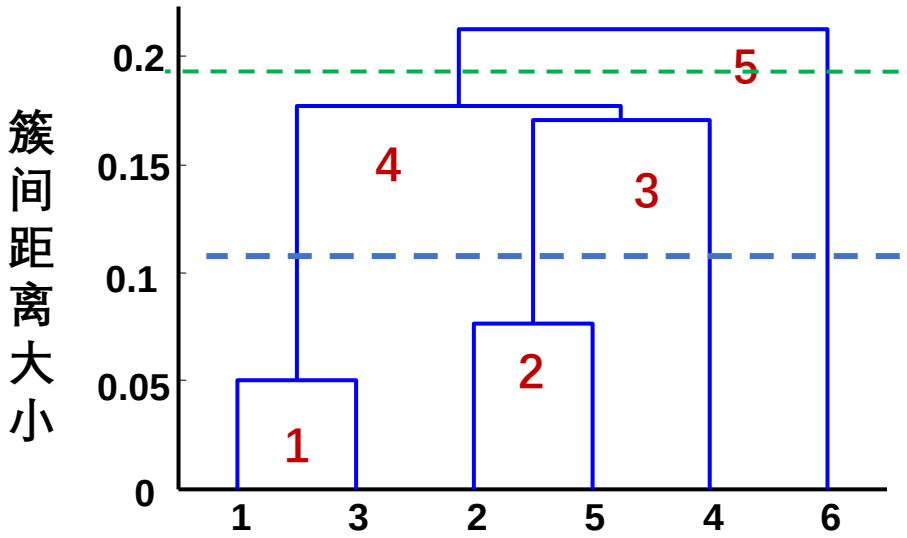
	Document	Category	ClusterLabel
0	The sky is blue and beautiful.	weather	0
1	Love this blue and beautiful sky!	weather	0
2	The quick brown fox jumps over the lazy dog.	animals	1
3	A king's breakfast has sausages, ham, bacon, e...	food	2
4	I love green eggs, ham, sausages and bacon!	food	2
5	The brown fox is quick and the blue dog is lazy!	animals	1
6	The sky is very blue and the sky is very beaut...	weather	0
7	The dog is lazy but the brown fox is quick!	animals	1

凝聚层次聚类
AGNES(Agglomerative Nesting)

AGNES算法基本步骤

1. 每个样本各自为一个簇，把簇距离矩阵设置为样本之间的距离矩阵
2. **Repeat**
3. 把两个最近的簇合并
4. 更新簇距离矩阵
5. **Until** 只剩下一个簇





关键问题：如何计算两个簇之间的紧密度

不能的度量方法得到不同的层次聚类算法

设定不同簇距离阈值，得到多种划分结果。

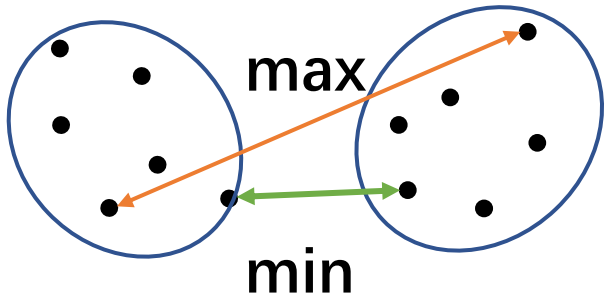
基于互连性(Linkage-based)的方法

- 最短距离/单连接(single-linkage):

  属于两个不同簇的样本之间距离的最小值：$l(C_r, C_s) = min_{x_{ri} \in C_r, x_{sj} \in C_s} d(x_{ri}, x_{sj})$

- 最长距离/全连接(Complete-linkage):

  属于两个不同簇的样本之间距离的最大值：
  $l(C_r, C_s) = max_{x_{ri} \in C_r, x_{sj} \in C_s} d(x_{ri}, x_{sj})$



- 平均距离/平均连接(Group-average):

  属于两个不同簇的样本之间距离的平均值：
  $l(C_r, C_s) = \frac{1}{|C_r||C_s|} \sum_{x_{ri} \in C_r, x_{sj} \in C_s} d(x_{ri}, x_{sj})$

Ward linkage

- Ward's linkage:最小化簇内距离，比如簇内每两个样本之间的距离。一般默认为欧式距离的平方。
- 在现有簇集合中，找到两个合并后带来方差增量最小的两个簇进行合并。

$$V(C_r) = \sum_{x_{ri}, x_{sj} \in C_r} d(x_{ri}, x_{sj})$$

$$l(C_r, C_s) = V(C_r \cup C_s) - V(C_r) - V(C_s)$$

基于ward-linkage层次聚类

```
from scipy.cluster.hierarchy import ward, dendrogram , linkage
from sklearn.metrics.pairwise import cosine_similarity
similarity_matrix = cosine_similarity(tv_matrix)
Z = linkage(similarity_matrix,'ward')
```

```
import pandas as pd
pd.DataFrame(Z, colu
```

sklearn.cluster.AgglomerativeClustering

如果linkage采用“ward”, 只接受 “euclidean” 距离度量。

]:

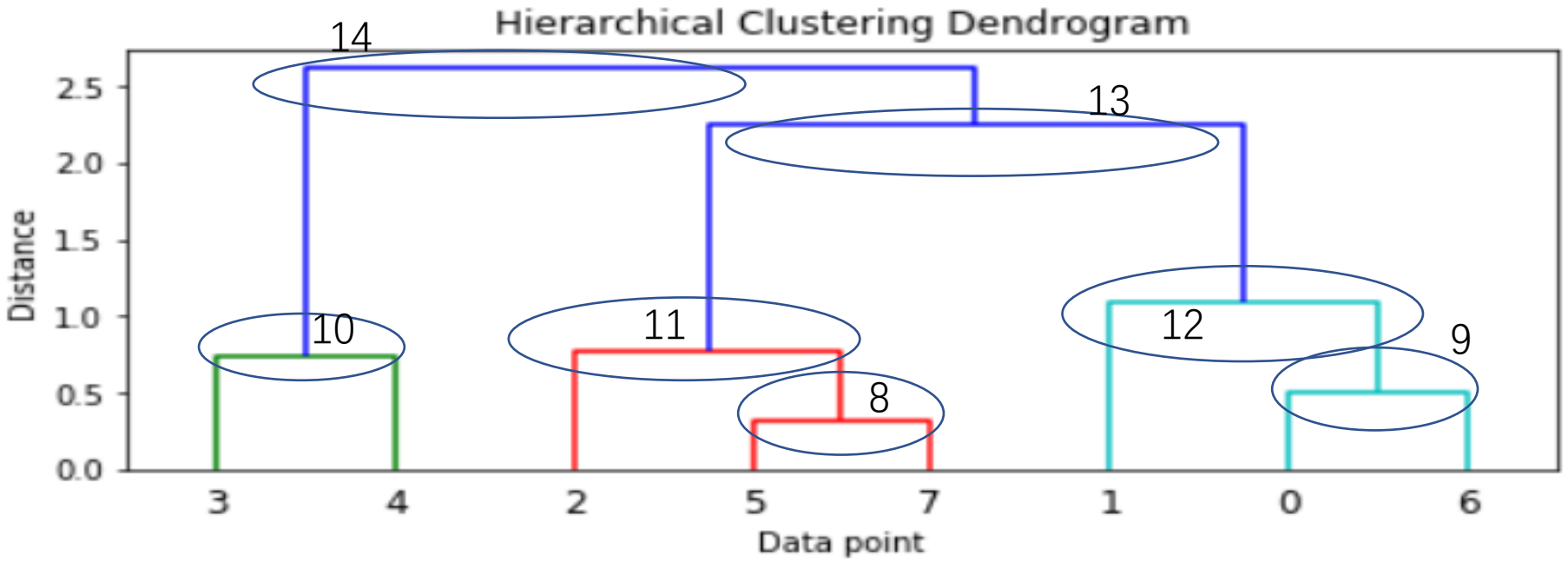
	Document\Cluster 1	Document\Cluster 2	Distance	Cluster Size
0	5	7	0.313327	2
1	0	6	0.494521	2
2	3	4	0.740642	2
3	2	8	0.772872	3
4	1	9	1.08406	3
5	11	12	2.25089	6
6	10	13	2.61438	8

每次合并产生的簇的id

8
9
10
11
12
13
14

打印树状结构

```
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 3))
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Data point')
plt.ylabel('Distance')
dendrogram(Z)
plt.show()
```



从层次结构得到划分

```
from scipy.cluster.hierarchy import fcluster
max_dist = 1.0
cluster_labels = fcluster(Z, max_dist, criterion='distance')
```

```
import numpy as np
corpus = np.array(corpus)
corpus_df = pd.DataFrame({'Document': corpus, 'Category': labels})
corpus_df['ClusterLabel']=cluster_labels
corpus_df
```

5]:

	Document	Category	ClusterLabel
0	The sky is blue and beautiful.	weather	3
1	Love this blue and beautiful sky!	weather	4
2	The quick brown fox jumps over the lazy dog.	animals	2
3	A king's breakfast has sausages, ham, bacon, e...	food	1
4	I love green eggs, ham, sausages and bacon!	food	1
5	The brown fox is quick and the blue dog is lazy!	animals	2
6	The sky is very blue and the sky is very beaut...	weather	3
7	The dog is lazy but the brown fox is quick!	animals	2

对20newsgroups数据集进行聚类

```
import pandas as pd
import numpy as np
data=pd.read_csv('D:\Anaconda3\Lib\site-packages\sklearn\datasets\data\clean_newsgro
pro_corpus=np.array(data['Clean Article'])
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import cross_val_score
# 对训练集建立词袋模型
tv = TfidfVectorizer(min_df=0.0, max_df=1.0)
tv_matrix = tv.fit_transform(pro_corpus)
```

```
from sklearn.cluster import KMeans
km = KMeans(n_clusters=20, max_iter=1000, n_init=50, random_state=42).fit(tv_matrix)
```

聚类结果分析

- 除了簇标签还有什么可以用来描述聚类结果?
- 每个簇最重要的特征/关键词?
- 每个簇最具有代表性的样本?

聚类和相似度

由于没有监督信息，聚类在很大程度上依赖于样本之间的相似度。

一个对数据集有效描述的相似度往往比聚类算法本身更重要。

如何得到有效的样本之间的相似度?

- 对当前特征空间采用更合适相似度度量-对词袋表示用余弦相似度来计算文档之间相似度。
- 基于语义、知识库等外部信息直接计算相似度。
- 通过特征空间变换得到更好的表示，即相似的样本具有相近的表示—得到更好的词嵌入空间，用常用距离度量就能有效描述。

主题建模

1.1 背景及应用

1.2 基于矩阵分解的模型LSI、NMF

1.3* 基于概率的主题模型LDA

1.4 对主题模型的解释和应用

主题检测

考虑以下任务：

老师给了你1000篇论文，让你总结一下，这些论文在研究哪些课题？ -Information overload

- 文档归类：通过监督（分类）或无监督（聚类）方法把文档分组，使得相同类别的文档在同一组，偏向于文档**处理**任务，方便后续的处理和分析。
- 主题检测：快速了解一个大文档数据集的主要内容，包括如主题检测、文本摘要，偏向文本**分析**任务。
- 虽然处理和分析任务有不同侧重点，但方法之间有很多交叉，比如通过聚类也可以进行主题检测。

主题检测

- 从包含大量文档的数据中提取出不同的主题或概念，即隐藏着的语义结构。注意：每个文档可能与多个主题相关。

相关问题：

检测出文档数据集中主要的话题；

对这些结果进行解释和分析；

基于已有模型对新的文档进行主题预测。

主要方法

问题：给定term-doc矩阵$\boldsymbol{X}$，一般是基于词袋模型，每列对应一个文档，每行对应一个词，得到term-topic和doc-topic关系。

常用方法:

- 基于矩阵分解的方法（LSI、NMF）
- 基于概率模型的方法（LDA）

截断SVD：低秩矩阵分解

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$$

$$[\boldsymbol{u}_1, \ldots \boldsymbol{u}_r] \begin{bmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_r \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_1^T \\ \vdots \\ \boldsymbol{v}_r^T \end{bmatrix}$$

取最大的k个奇异值以及对应的$\boldsymbol{U}$和$\boldsymbol{V}$

低秩近似矩阵 $\boldsymbol{X'} = \boldsymbol{U_k}\boldsymbol{\Sigma_k}\boldsymbol{V_k^T}$

隐语义检索（Latent Semantic Indexing ）

- 基于对term-doc矩阵的SVD分解。
- 基本思想：相似的词趋向于出现在相同的上下文。

给定term-doc矩阵$\boldsymbol{X}_{t\times d}$，以及给定隐空间维度k，通过SVD得到$\boldsymbol{U_k}$、$\boldsymbol{V_k}$、和$\boldsymbol{\Sigma_k}$,其中隐空间中的每个维度对应一个概念，$\boldsymbol{V}_{d\times k} = [\boldsymbol{v}_1, \ldots \boldsymbol{v}_k]$对应每个doc在k维隐空间中的坐标，也可以理解为文档-概念的关联矩阵。
把$\boldsymbol{U\Sigma}^{-1}$作为新的坐标系，那么$\boldsymbol{V}_{d\times k}$就是$\boldsymbol{X}$在这个坐标系上的投影，即$\boldsymbol{V}=\boldsymbol{XU\Sigma}^{-1}$。

对于一个新的文档或者查询$\boldsymbol{q}$，它在这个隐空间的投影为 $\boldsymbol{qU\Sigma}^{-1}$。

思考：怎么计算两个文档在k维隐语义空间的欧式距离？

SVD的应用

- 文本分析：对term-doc 的co-occurrence矩阵进行分解，
  即LSI(Latent factor indexing), 又叫LSA(Latent semantic analysis)
- 推荐系统：对item-user打分矩阵进行分解

回顾：PCA降维中是如果利用SVD？

gensim实现

```
#文本样例
corpus = ['The sky is blue and beautiful.',
          'Love this blue and beautiful sky!',
          'The quick brown fox jumps over the lazy dog.',
          "A king's breakfast has sausages, ham, bacon, eggs, toast and beans",
          'I love green eggs, ham, sausages and bacon!',
          'The brown fox is quick and the blue dog is lazy!',
          'The sky is very blue and the sky is very beautiful today',
          'The dog is lazy but the brown fox is quick!' ]
labels = ['weather', 'weather', 'animals', 'food', 'food', 'animals', 'weather', 'animals']
```

```
norm_corpus=normalize_corpus(corpus,doc_tokenize=True)
dictionary = gensim.corpora.Dictionary(norm_corpus)
bow_corpus = [dictionary.doc2bow(text) for text in norm_corpus]
bow_corpus[0:3]
```

```
[[(0, 1), (1, 1), (2, 1), (3, 1)],
 [(1, 1), (2, 1), (3, 1), (4, 1), (5, 1)],
 [(0, 1), (6, 1), (7, 1), (8, 1), (9, 1), (10, 1), (11, 1)]]
```

bow_corpus为gensim输入的格式，每行对应一个文档，每个二元组为该文档中出现的词的id和次数。

```
import gensim
import pandas as pd
import numpy as np
TOTAL_TOPICS=3
#建模
lsi_bow = gensim.models.LsiModel(bow_corpus, id2word=dictionary,
 num_topics=TOTAL_TOPICS, onepass=True, chunksize=1740, power_iters=1000)
```

```
#查看结果
for topic_id, topic in lsi_bow.print_topics(num_topics=3,
    num_words=5):
    print('Topic #'+str(topic_id+1)+':')
    print(topic)
    print()
```

num_words:
对应每个主题打印关键词的个数

```
Topic #1:
0.789*"," + 0.241*"sausages" + 0.241*"eggs" + 0.241*"bacon" + 0.241*"ham"

Topic #2:
-0.376*"brown" + -0.376*"fox" + -0.376*"quick" + -0.376*"lazy" + -0.376*"dog"

Topic #3:
0.638*"sky" + 0.450*"beautiful" + 0.415*"blue" + 0.189*"today" + -0.176*"dog"
```

权重可能出现负数

得到SVD的三个矩阵

```
term_topic = lsi_bow.projection.u
singular_values = lsi_bow.projection.s
topic_document = (gensim.matutils.corpus2dense(lsi_bow[bow_corpus],
                  len(singular_values)).T / singular_values).T
```

corpus2dense:转成普通矩阵，第二个参数为列的数目。

```
document_topics = pd.DataFrame(np.round(topic_document.T, 3),
                               columns=['T'+str(i) for i in range(1, TOTAL_TOPICS+1)])
```

	T1	T2	T3
0	0.005	-0.182	0.443
1	0.023	-0.233	0.476
2	0.013	-0.507	-0.256
3	0.865	0.083	-0.008
4	0.500	-0.047	0.009
5	0.032	-0.583	-0.122
6	0.006	-0.200	0.663
7	0.030	-0.517	-0.240

doc-topic 矩阵

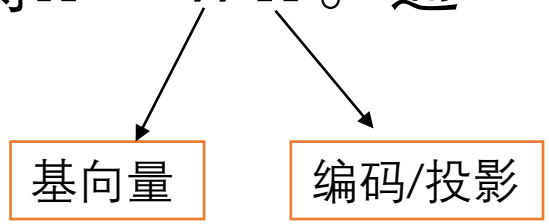
与直接调用SVD实现得到的结果一致（正负号、以及每个主题的编号不固定）

```
from scipy.sparse.linalg import svds
td_matrix = gensim.matutils.corpus2dense(corpus=bow_corpus,  num_terms=len(dictionary))
u, s, vt = svds(td_matrix, k=TOTAL_TOPICS, maxiter=10000)
```

```
document_topics2 = pd.DataFrame(np.round(vt.T, 3),
                                columns=['T'+str(i) for i in range(1, TOTAL_TOPICS+1)])
```

非负矩阵分解

- 把一个非负矩阵X分解成两个非负矩阵W和H，使得$X \approx WH$。通过最小化以下分解误差来求解



$$\text{argmin}_{W,H} \frac{1}{2} \|X - WH\|_F^2$$

s.t.$W \geq 0, H \geq 0$

与SVD的差别：不要求正交，但是要求非负。

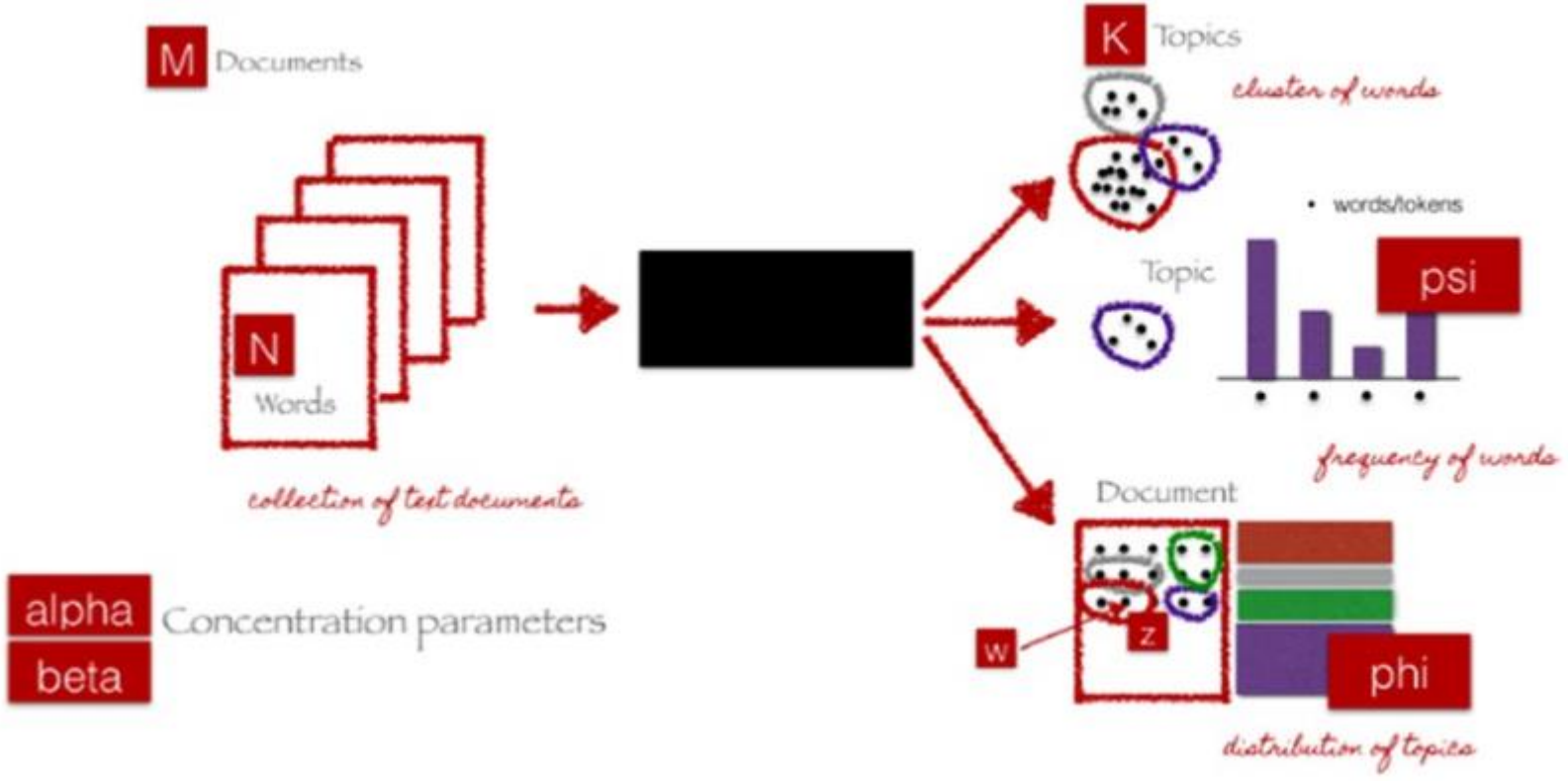
好处：非正交性允许存在重叠，非负权重有更好的可解释性。

除了用均方误差，也可以用KL散度作为分解误差度量。

参考文献： Daniel D lee and H.Sebastian Seung.” Learning the parts of objects by non-negative matrix factorization”. Nature， 1999, 401(6755): 788-791

基于概率模型的方法

LDA（Latent Dirichlet Allocation ）生成式模型，假设主题服从Dirichlet 分布。



多项式分布

多项式分布（Multinomial Distribution），二项式分布的推广。

某随机实验如果有k个可能结局$A_1, A_2 \cdots A_k$，分别将他们的出现次数记为随机变量X_1、$X_2 \cdots X_k$，它们的概率分布分别是p_1，$p_2 \cdots p_k$，那么在n次采样的总结果中，A_1出现n_1次、A_2出现n_2次…A_k出现n_k次的这种事件的出现概率P符合多项式分布：

$$Mul(\boldsymbol{X}|\boldsymbol{p}) = P(X_1 = n_1, X_2 = n_2, \ldots X_k = n_k) = \frac{n!}{n_1! \ldots n_k!} {p_1}^{n_1} \cdots {p_k}^{n_k} = n! \prod_{i=1}^{k} {p_i}^{n_i} / n_i!,$$

其中 $\sum_{i=1}^{k} n_i = n$，$p_i \geq 0, \sum_{i=1}^{k} p_i = 1$

特殊情况，各个单独事件出现的概率相等：$p_1 = p_2 \cdots = p_k = \frac{1}{k}$，比如扔骰子，6个面对应有k=6个可能的不同点数，它们的概率分布分别是$p_1 = p_2 \cdots = p_k = \frac{1}{6}$。但如果这个骰子不是一个规则的形状，那么每个概率就不一定相等。

$$\boldsymbol{X} \sim Mul(\boldsymbol{X}|\boldsymbol{p})$$

狄利克雷分布作为先验

- 狄利克雷分布（Dirichlet Distribution）或多元Beta分布，是Beta分布在高维情形的推广，其概率密度函数为：

$$\operatorname{Dir}(\boldsymbol{X}|\boldsymbol{\alpha}) = \frac{\Gamma(\alpha_0)}{\prod_{i=1}^{d}\Gamma(\alpha_i)}\prod_{i=1}^{d-1}X_i^{\alpha_i-1}(1-X_1-\cdots-X_{d-1})^{\alpha_d-1}$$

从Dirichlet 分布随机采样一个k维向量$(x_1, x_2 \cdots x_k)$,由于则各元素相加之和等于1，所以在贝叶斯推断（Bayesian inference）中作为多项分布的共轭先验，即多项式分布中的$p_1, p_2 \cdots p_k$：

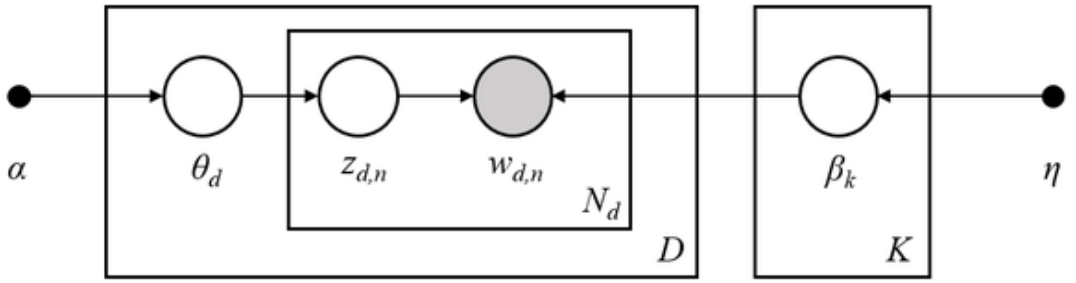
$$\boldsymbol{p} \sim Dir(\boldsymbol{p}|\boldsymbol{\alpha})$$

$$\boldsymbol{x} \sim Mul(\boldsymbol{x}|\boldsymbol{p})$$

基于概率模型的方法

K: 主题数目
N: 词的数目
D: 文档数目
$\boldsymbol{\alpha}$和$\boldsymbol{\eta}$为Dirichlet分布参数，有用户给定。
其中：
$\boldsymbol{\alpha}$: 文档到主题的Dirichlet分布参数，为K维向量
$\boldsymbol{\eta}$: 主题到词的Dirichlet分布参数，为N维向量

Doc-topic 概率(D×K)　　Topic-word 概率(K×N)



中间变量

- $\boldsymbol{\beta}_k$:N个词在主题k里面出现的概率,服从Dir($\boldsymbol{\eta}$)分布;
- $\boldsymbol{\theta}_d$:K个主题在文档d中的分布,服从Dir($\boldsymbol{\alpha}$)分布;
- z_{dn} :文档d中第n个词的主题，给定$\boldsymbol{\theta}=\{\boldsymbol{\theta}_d\}_{d=1\ldots D}$，主题$\boldsymbol{z}$服从Multinomial分布 $\mathrm{p}(\mathrm{z}|\boldsymbol{\theta}) = \prod_{d=1}^{D}\prod_{k=1}^{K}\theta_{dk}{}^{m_{dk}}$，其中 m_{dk} 为在文档d中属于主题k的词的数目;
- w_{dn} （观测到的变量）：文档d中的第n个词。给定$\boldsymbol{\beta}=\{\boldsymbol{\beta}_k\}_{k=1\ldots K}$，$w$服从Multinomial分布$\mathrm{p}(w|\mathrm{z},\boldsymbol{\beta}) = \prod_{k=1}^{K}\prod_{n=1}^{N}\boldsymbol{\beta}_{kn}{}^{m_{kn}}$，其中$m_{kn}$表示语料库中单词n被赋予主题k的次数。

生成过程

1.对每个主题$k \in K$，从Dir($\boldsymbol{\beta}|\boldsymbol{\eta}$)中抽样一个(长度为N， 元素和为1的)向量$\boldsymbol{\beta}_k$;
2.对每个文档$d \in D$，从Dir($\boldsymbol{\theta}|\boldsymbol{\alpha}$)中抽样一个(长度为K，元素和为1的)向量$\boldsymbol{\theta}_d$;
3. 对文档d中的每个词n:
3.1 从Mul($z|\boldsymbol{\theta}_d$)中采样出z_{dn};
3.2 从Mul($w\ |\beta_{z_{dn}}$)中采样得到观察到的词w_{dn}.

吉布斯采样（Collapsed Gibbs sampling）

1. Initialize the necessary parameters.
2. For each document, randomly initialize each word to one of the **K** topics.
3. Start an iterative process as follows and repeat it several times. For each document **D**, for each word **W** in document, and for each topic **T**:
   - Compute $P(T|D)$, which is proportion of words in **D** assigned to topic **T**.
   - Compute $P(W|T)$, which is proportion of assignments to topic **T** over all documents having the word **W**.
   - Reassign word **W** with topic **T** with probability $P(T|D) \times P(W|T)$, considering all other words and their topic assignments.

Gensim实现

```
lda_model = gensim.models.LdaModel(corpus=bow_corpus, id2word=dictionary,
                                   chunksize=1740, alpha='auto', eta='auto',
                                   random_state=42, iterations=500,
                                   num_topics=TOTAL_TOPICS, passes=20, eval_every=None)

for topic_id, topic in lda_model.print_topics(num_topics=10, num_words=5):
    print('Topic #'+str(topic_id+1)+':')
    print(topic)
```

```
Topic #1:
0.123*"blue" + 0.123*"sky" + 0.094*"!" + 0.094*"beautiful" + 0.066*"dog"
Topic #2:
0.087*"jumps" + 0.087*"." + 0.087*"lazy" + 0.087*"fox" + 0.087*"brown"
Topic #3:
0.209*"," + 0.077*"bacon" + 0.077*"ham" + 0.077*"eggs" + 0.077*"sausages"
```

LSI和LDA的结果（term-topic）

LSI	Topic1	Topic2	Topic3
Term1	,	lazy	sky
Term2	sausages	brown	beautiful
Term3	eggs	quick	blue
Term4	bacon	fox	today
Term5	ham	dog	dog
Term6	king	!	brown
Term7	'	blue	lazy
Term8	toast	sky	quick
Term9	beans	.	fox
Term10	breakfast	beautiful	Love

LDA	Topic1	Topic2	Topic3
Term1	blue	jumps	,
Term2	sky	.	bacon
Term3	!	lazy	ham
Term4	beautiful	fox	eggs
Term5	dog	brown	sausages
Term6	quick	quick	king
Term7	brown	dog	toast
Term8	fox	sky	beans
Term9	lazy	blue	'
Term10	.	beautiful	green

定量评估

- Coherence

量化的衡量检测到的主题的质量或有效性

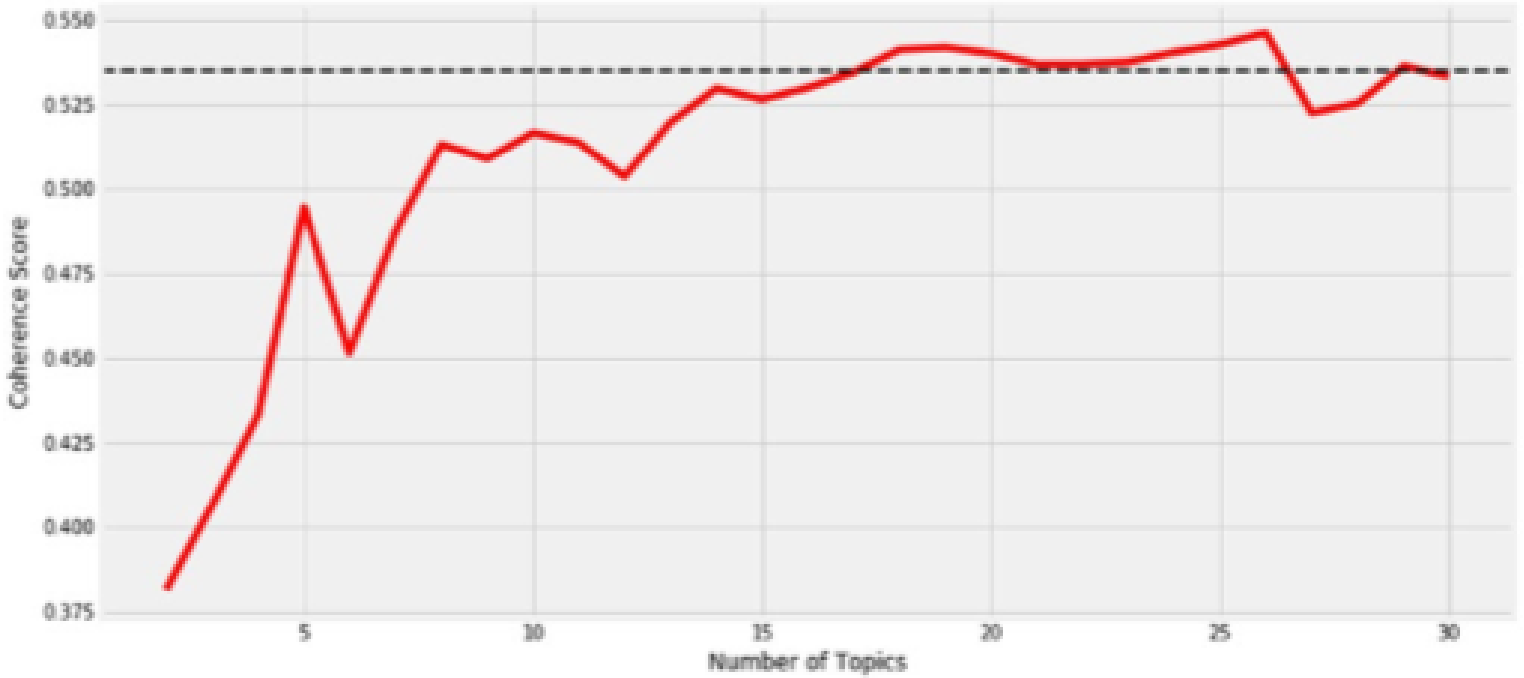
```
cv_coherence_model_lda = gensim.models.CoherenceModel( model=lda_model,
                                                       corpus=bow_corpus,
                                                       texts=norm_corpus,
                                                       dictionary=dictionary,
                                                       coherence='c_v')
avg_coherence_cv = cv_coherence_model_lda.get_coherence()
avg_coherence_cv
```

```
0.3881761586628363
```

model=lsi_bow时，得到的结果为：0.3779

主题数目

- 与聚类一样，需要给出主题数目是一个超参数。拐点法。



对主题模型的解释和分析

- 最相关主题分布

把每个文档只关联到其最相关的主题；

```
#对已经建立的主题模型如lda_model，输入词袋表示的目标文档数据如bow_corpus得到主题分布
tm_results = lda_model[bow_corpus]
#每个文档对应的权重最大的主题id以及权重
corpus_topics = [sorted(topics, key=lambda record: -record[1])[0] for topics in tm_results]
corpus_topics
```

```
[(0, 0.97704136),
 (0, 0.98143846),
 (1, 0.9746297),
 (2, 0.98769987),
 (2, 0.98235667),
 (0, 0.98657924),
 (0, 0.9814385),
 (0, 0.984422)]
```

对主题模型的解释和分析

- 查看所有主题在文档中的分布，即有多少文档是关联到某个主题

	Dominant Topic	Doc Count	% Total Docs	Topic Desc
0	1	5	62.5	blue, sky, !, beautiful, dog, quick, brown, fox, lazy, .
1	2	1	12.5	jumps, ., lazy, fox, brown, quick, dog, sky, blue, beautiful
2	3	2	25.0	,, bacon, ham, eggs, sausages, king, toast, beans, ', green

- 每个主题中最具有代表性的文档

Dominant Topic	Document	Dominant Topic	Contribution %	Topic Desc	Paper
1	5	1	98.66	blue, sky, !, beautiful, dog, quick, brown, fox, lazy, .	The brown fox is quick and the blue dog is lazy!
2	2	2	97.46	jumps, ., lazy, fox, brown, quick, dog, sky, blue, beautiful	The quick brown fox jumps over the lazy dog.
3	3	3	98.77	,, bacon, ham, eggs, sausages, king, toast, beans, ', green	A king's breakfast has sausages, ham, bacon, eggs, toast and beans

预测新文档相关的主题

给定已经建立的主题模型，预测新的文档到这些主题的关系，即新的文档与哪些主题最相关。

此过程与分类中的预测阶段类似，不同的是主题建模即训练过程是无监督的。

首先对需要预测主题的文档进行预处理和特征表示，然后调用已经训练好的主题模型，具体在genism 和 sklearn里面的操作为：

- 基于gensim：topic_prediction=topic_model[corpus]

  其中corpus为对词袋表示的数据，topic_model为训练好的某个主题模型比如lsi或ida等。
- 基于sklearn用model.transform实现，如: nmf_model.transform(cv_new_features)

  其中cv_new_features为doc-term 矩阵。

预测新文档相关的主题

以下函数基于gensim对词袋表示的数据corpus，以及训练好的主题模型topic_model，返回最相关的topn个主题（默认为3个）。

```
def get_topic_predictions(topic_model, corpus, topn=3):
    topic_predictions = topic_model[corpus]
    best_topics = [[(topic, round(wt, 3))  for topic, wt in sorted(topic_predictions[i],
        key=lambda row: -row[1]) [:topn]] for i in range(len(topic_predictions))]
    return best_topics
```

以下对上面函数的调用得到对新的文档（norm_bow_features）对于前面lda模型的最相关的2个主题。

```
topic_preds = get_topic_predictions(topic_model=lda_model,
                                    corpus=norm_bow_features, topn=2)
```

主题作为特征

- 经过主题检测，得到doc-topic矩阵。
- 把以上矩阵中的每列，即一个主题看成一个特征，每行即由k个主题为特征表示的文本向量。
- 如何用于其他任务，如分类？

  和其他特征如词袋特征拼接起来