Latent Semantic Analysis

SVD, LDA

CHIEN CHIN CHEN

Term Grouping and Why

Now, we know how to do document clustering. How about grouping **terms** into clusters?

- Each cluster would represent a certain concept or **TOPIC**!!
- For instance, *cell*, *android*, and *iphone* are all about mobile phone.

Term clustering makes **semantic search** possible.

- Search engines search for documents based on their meaning, rather than keyword matching.
- For instance, users search for "*iphone*" and "*12 pro*" would obtain the same result.

Identifying terms with similar meaning is a challenging text mining research topic, called **latent semantic analysis** (LSA).

- Here, we present two well-known LSA methods: **singular value decomposition** and **latent Dirichlet allocation**.

Latent Semantic Analysis Benefits A LOTS

LSA helps represent each document as a topic vector.

- Successfully reduce the dimensionality of document.

Benefits of such **dimension reduction**:

- Enhance the similarity calculation of documents that further bring out better classification/clustering performance.
  - Cosine similarity (Euclidean distance) based on sparse TFIDF/frequency vectors would be messed up.
- Reduce storage requirement and facilitate model learning and testing.

Remember Token Normalization?

Stemming and **lemmatization** also help similarity calculation by grouping terms **superficially different** together.

- *Manufacture, manufacturing, manufactured, manufactured*
- *Operator, operation, operate*

LSA take token normalization to another level by handling **synonyms**.

- *Cell, phone, mobile* → the same topic

Singular Value Decomposition (1/6)

One popular method of latent semantic analysis is **Singular Value Decomposition** (SVD).

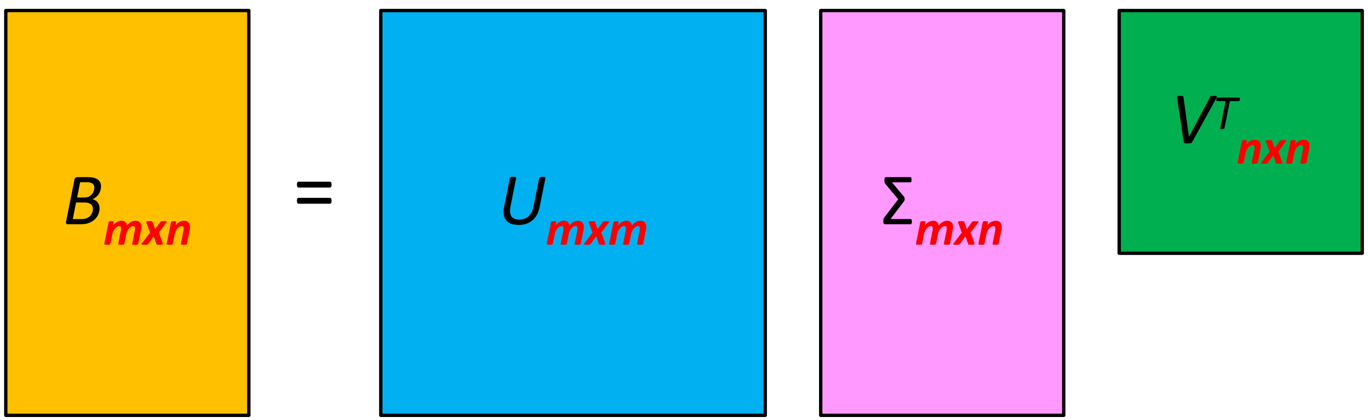
- SVD is not a new technology; you can find it in every linear algebra textbook.
- Actually, it was heavily used in the field of data mining to reduce the dimensions of data.

We first talk about the math of SVD, and then apply it to latent semantic analysis.

Singular Value Decomposition (2/6)

Mathematically, SVD decomposes a (**any**) matrix B into three matrices.

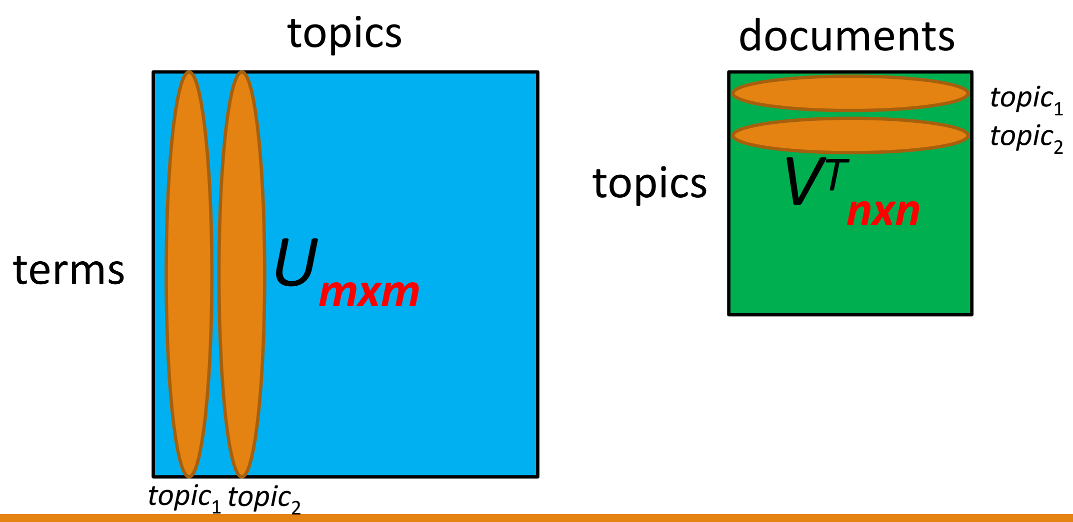
- $B = U \Sigma V^T$



Singular Value Decomposition (3/6)

The **columns** in ***U*** and ***V*** are **left** and **right** **singular vectors** respectively.

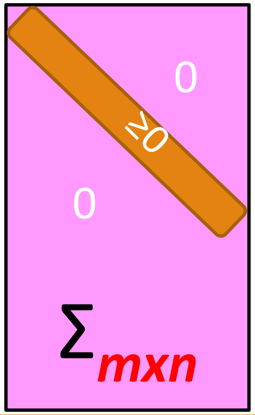
- Which are also the **eigenvectors** of BB^T and B^TB, respectively.
- In terms of text corpus, these singular vectors form the basis of retaining term-term/document-document relation.
  - Or the topics in terms of terms and documents



Singular Value Decomposition (4/6)

The matrix Σ is a **diagonal matrix**.

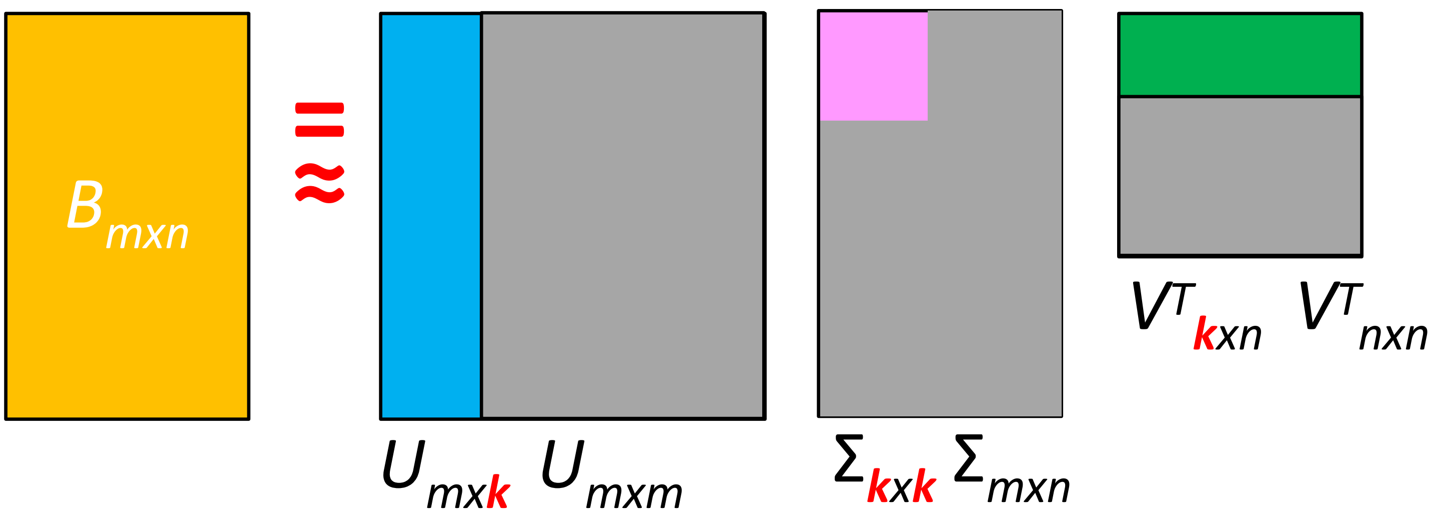
- The diagonal entries are **singular values** of *B*, and they are **non-negative**!!
  - Usually, the diagonal elements are **arranged in descending order**.
  - They tell how much each singular vector contributes **in restoring the matrix *B***.



Singular Value Decomposition (5/6)

Restoring *B* and dimension reduction:

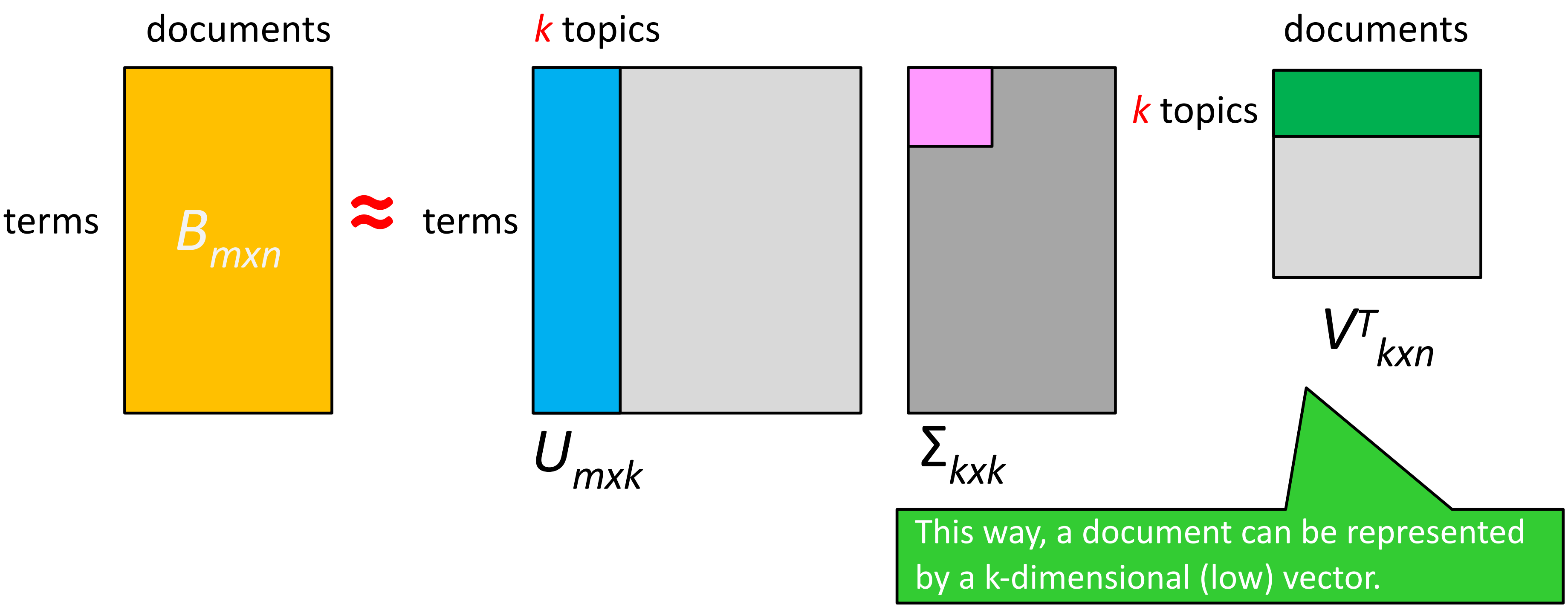
- *B* can be **approximated** by preserving ***k*** significant singular values and the corresponding singular vectors.
- k **<<** m, n



Singular Value Decomposition (6/6)

Back to the term-document matrix

- By preserving *k* most significant topics, we may re-construct the original text!! (without losing too much text information)



SVD + KNN Example (1/6)

In our previous examples, we input classification and clustering models **sparse** TFIDF vectors.

In this practice, we first convert documents into low-dimensional topic vectors using SVD.

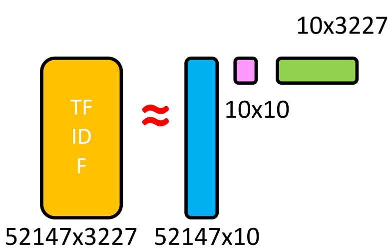
Then, using them to construct a KNN classification model.

SVD + KNN Example (2/6)

```
In [1]: import xlrd

In [2]: workbook = xlrd.open_workbook('./blog-gender-dataset.xlsx')

In [3]: booksheet = workbook.sheet_by_name('data')

In [4]: texts = []
        labels = []

In [5]: for i in range(booksheet.nrows):
            labels.append(booksheet.cell(i,1).value)
            texts.append(booksheet.cell(i,0).value)

In [ ]:

In [ ]:

In [6]: from sklearn.feature_extraction.text import TfidfVectorizer

In [7]: TFIDF_vectorizer = TfidfVectorizer(min_df=1, stop_words='english')

In [8]: TFIDF_vectors = TFIDF_vectorizer.fit_transform(texts)

In [9]: TFIDF_vectors.shape
Out[9]: (3227, 52147)
```

Nothing new here; just create the TFIDF matrix (vectors)

SVD + KNN Example (3/6)




```
In [10]: from sklearn.decomposition import TruncatedSVD

In [11]: svd_model = TruncatedSVD(n_components = 10)
         SVD_vectors = svd_model.fit_transform(TFIDF_vectors)

In [12]: SVD_vectors

Out[12]: array([[ 0.09360186, -0.00497794, -0.0092503 , ..., 0.02018717,
                 -0.04115087, 0.00619918],
                [0.11170057, 0.00100605, -0.10135507, ..., 0.00228109,
                 -0.02893276, 0.03480206],
                [0.13790035, -0.01113413, -0.01564522, ..., 0.01231188,
                  0.02816334, 0.04331045],
                ...,
                [0.09251537, -0.0023233, -0.01838235, ..., -0.02191315,
                  0.04064763, 0.02189141],
                [0.05723286, 0.0040563, -0.03914376, ..., 0.00984387,
                  0.02242617, 0.01619263],
                [0.21775351, -0.01520361, 0.0113637, ..., -0.07060371,
                 -0.03383736, -0.0051149 ]])

In [13]: SVD_vectors.shape

Out[13]: (3227, 10)

In [14]: svd_model.components_.shape

Out[14]: (10, 52147)

In [15]: svd_model.singular_values_

Out[15]: array([8.29865539, 3.86754367, 3.158246, 2.93781599, 2.71310144,
                2.65305636, 2.58014703, 2.50775483, 2.44522981, 2.36002912])
```

SVD + KNN Example (4/6)

In [16]:
```
def print_topics(model, vectorizer, n_top_words):
    words = vectorizer.get_feature_names()
    for topic_index, topic in enumerate(abs(model.components_)):
        print("\nTopic #%d:" % topic_index)
        print(" ".join([words[i] for i in topic.argsort()[:-n_top_words - 1:-1]]))
```

In [17]:
```
print_topics(svd_model, TFIDF_vectorizer, 10)
```

```
Topic #0:
just like time know people really ve don life good

Topic #1:
nbsp life know therizinosaurs therizinosaurus blog therizinosaur art shop luis

Topic #2:
life new know person love god really friend ur friends

Topic #3:
people school day life food went person lunch got health

Topic #4:
school women health people don food want care lunch eat

Topic #5:
school food lunch ur just abt students class person best

Topic #6:
blog health care school post want ve insurance women friend

Topic #7:
women men health care time year like blog fat weight

Topic #8:
love life blog ve book women game world men phone

Topic #9:
game women school blog god health weight ve games movie
```

SVD + KNN Example (5/6)

```
In [18]: x_train = SVD_vectors[0:2500]
         x_test = SVD_vectors[2500:]
         y_train = labels[0:2500]
         y_test = labels[2500:]
```

Train/testing documents now are represented as **topic vectors**!!

```
In [19]: from sklearn.neighbors import KNeighborsClassifier

         KNN_model = KNeighborsClassifier(n_neighbors = 5)
```

```
In [20]: KNN_model.fit(x_train,y_train)

Out[20]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                              metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                              weights='uniform')
```

SVD + KNN Example (6/6)

TF-IDF

In [15]:
```
from sklearn import metrics
print(metrics.classification_report(expected_results, predicted_results))
```

	precision	recall	f1-score	support
F	0.62	0.59	0.60	370
M	0.59	0.62	0.61	357
accuracy			0.60	727
macro avg	0.60	0.60	0.60	727
weighted avg	0.60	0.60	0.60	727

SVD

In [twenty three]:
```
from sklearn import metrics
print(metrics.classification_report(expected_results, predicted_results))
```

```
precision recall f1-score support

F 0.63 0.58 0.60 370
M 0.60 0.65 0.62 357

accuracy 0.61 727
macro avg 0.61 0.61 0.61 727
weighted avg 0.61 0.61 0.61 727
```

Dimension reduction: from 52147 to 10 !!

Latent Dirichlet Allocation (1/13)

A **probability-based** topic discovery model.

- Reference paper: David M. Blei, Andrew Y. Ng, and Michael I. Jordan, Latent Dirichlet Allocation, Journal of Machine Learning Research, 3 (2003), 993-1022.

dl.acm.org › doi - 翻譯這個網頁

Latent dirichlet allocation | The Journal of Machine Learning ...

Abstract. We describe latent Dirichlet allocation (LDA), a generative probabilistic model for collections of discrete data such as text corpora. LDA is a three-level hierarchical Bayesian model, in which each item of a collection is modeled as a finite mixture over an underlying set of topics.

由 DM Blei 著作 - 2003 - 被引用 33559 次 - 相關文章

Abstract · References · Index Terms

大衛·布萊

教授

譯自英文 - 大衛·布萊（David M. Blei）是哥倫比亞大學統計與計算機科學系的教授。在2014年秋季之前，他是普林斯頓大學計算機科學系的副教授。他的工作主要是機器學習。 维基百科（英文)

查看原文說明

指導教授： 麥可·喬丹

學歷： 加利福尼亞大學 (2004 年)

專業領域： 人工智慧

獲頒獎項： 總統青年成就獎; 美國計算機協會會士 (2015)

住宅： 美國

獲獎紀錄： 古根漢獎學金自然科學類，美國及加拿大

吳恩達
Andrew Ng

出生	1976年4月18日（44歲）[1] 英國[1]
居住地	美國
母校	卡內基梅隆大學, 麻省理工學院, 加州大學伯克利分校, 萊佛士書院
知名於	深度學習,MOOC
網站	www.andrewng.org
科學生涯	
研究領域	人工智慧
機構	史丹福大學
論文	Shaping and Policy Search in Reinforcement Learning（2003）
博士導師	麥可·喬丹（Michael I. Jordan）

麥可·喬丹

科學家

麥可·I·喬丹，美國計算機科學和統計學學者，主要研究機器學習和人工智慧。目前擔任加州大學伯克利分校電機工程與計算機系和統計學系教授。2016年，據《科學》報導，麥可·I·喬丹是當今世界最有影響力的計算機科學家。 他的重要貢獻包括指出了機器學習與統計學之間的聯繫，並推動機器學習界廣泛認識到貝葉斯網絡的重要性。
維基百科

出生： 1956 年 2 月 25 日（64歲），美國路易斯安那州龐沙圖拉

博士導師： David Rumelhart; 唐·諾曼

其他著名學生： 約書亞·本希奧

居住地： 美國加利福尼亞州阿拉米達郡柏克萊

模範生： 大衛·布萊， 吳恩達， 佐賓·葛拉曼尼， 邢波， 丹尼爾·沃爾伯特， Emanuel Todorov， 塔瑪拉·布羅德里克， Percy Liang， 菲力普·薩貝斯

著作： Graphical Models, Exponential Families, and Variational Inference， 更多

Latent Dirichlet Allocation (2/13)

Concept/assumption:

- A document is generally involved several topics.
  - LDA views each document as **a random mixture over latent topics**.
  - Or … each document is with a topic distribution.
- Different topics prefer different words.
  - LDA characterizes a topic by **a distribution over words**.
  - Or … each topic is with a word distribution.

When composing a document, LDA assumes:

- A topic is selected based on the document's topic mixture.
- Then, a word is emitted in accordance with the topic's word distribution.

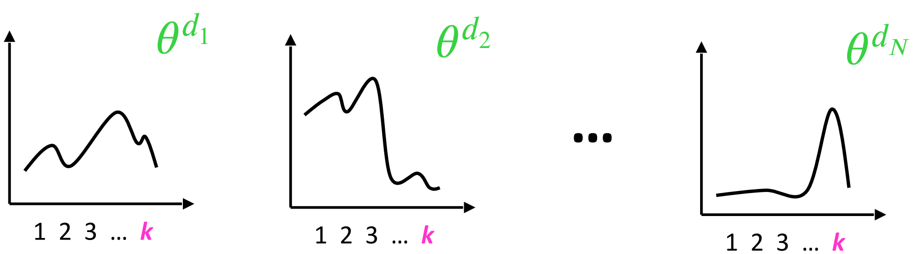
The two sets of distributions are unobserved!!
We would like to infer them from a text corpus.

Latent Dirichlet Allocation (3/13)

Formal definition:

Pre-defined!!

- Assuming there are ***k*** topics.
- Each document has a topic distribution θ^{d_i}, which obviously is a **multinomial distribution**.



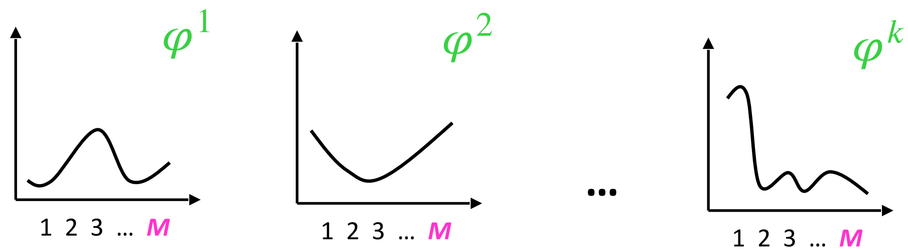
- The probability that a document d_i belongs to topic z ($1 \le z \le$ ***k***) can be expressed as

$$P(z \mid d_i) = \theta_z^{d_i}$$

This distribution, usually represented as a *k*-dimensional vector, is what the task of dimension reduction wants.

Latent Dirichlet Allocation (4/13)

- A topic can also be represented as a distribution over words (unique terms), $\boldsymbol{\varphi}^{z}$.



  - *M* is the size of the vocabulary.
  - Different topics prefer using different words.
  - φ^{z} is also a **multinomial distribution**.

- The probability that a term *w* is used to compose a content regarding to topic *z* is

$$P(w \mid z) = \varphi_{w}^{z}$$

Latent Dirichlet Allocation (5/13)

Based on the above distributions, the probability of seeing a word *w* in document d_i is expressed as:

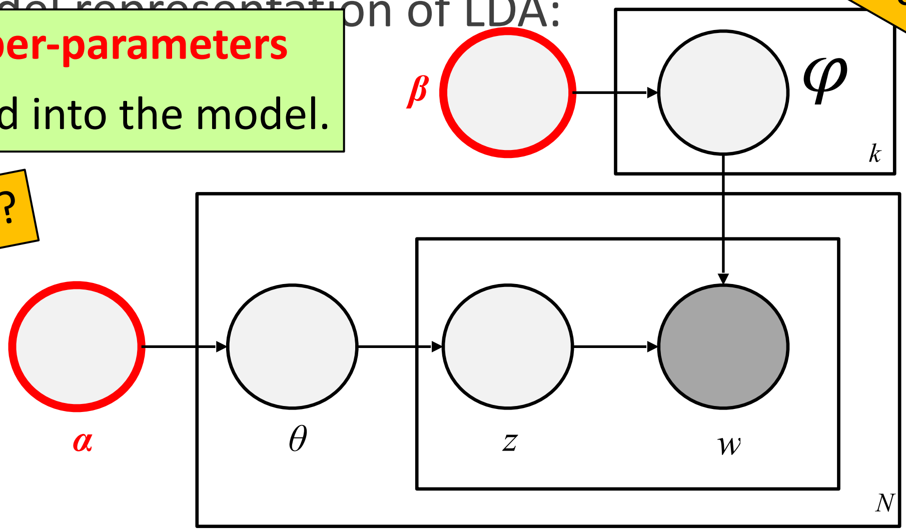
$$P(w \mid d_i) = \sum_{z=1}^{k} P\left(z \middle| d_i\right) P(w \mid z) = \sum_{z=1}^{k} \theta_z^{d_i} \varphi_w^z$$

A graphical model representation of LDA:

Two **Dirichlet hyper-parameters** ***α*** and ***β*** are added into the model.

What are those?

Good … but … where is **Dirichlet**?



Latent Dirichlet Allocation (6/13)

To understand the role of the two Dirichlet hyper-parameters, we have to talk about **maximum a posteriori (MAP) estimate**.

- Track back to the model-based clustering.

$$\theta_{MAP} = \underset{\theta \in \text{model space}}{\operatorname{argmax}} P(\theta \mid D)$$

$$= \underset{\theta \in \text{model space}}{\operatorname{argmax}} \frac{P(D \mid \theta) P(\theta)}{P(D)}$$

$$= \underset{\theta \in \text{model space}}{\operatorname{argmax}} P(D \mid \theta) P(\theta)$$

What if the prior is not uniform??

Can we incorporate our prior beliefs into the parameter estimate process?

To **simplify** the model search problem, **we generally assume that the prior probability of every model is equal** (e.g., $P(\vartheta_i) = P(\vartheta_j)$ for all i and j in the model space).

- Then…

$$\theta_{MAP} = \theta_{ML} = \underset{\theta \in \text{model space}}{\operatorname{argmax}} P(D \mid \theta)$$

Latent Dirichlet Allocation (7/13)

In LDA, the model parameters we are going to learn (estimate) from text corpus are θ^{d_i} and φ^{z}.

- They are all **multinomial distributions**.
- LDA adopts **Dirichlet distribution** to incorporate **our prior knowledge** into the model estimate process.
  - **Dirichlet distributions** is a **conjugate prior** for multinomial.
    - **Conjugate**: the posterior has the same form as the prior

Latent Dirichlet Allocation (8/13)

Dirichlet distribution:

- Suppose there are *K* outcomes of a certain experiment.
- Let ***q*** = [q_1, q_2, …, q_k] be a set of outcome probabilities.
  - $q_i \geq 0$ and $\sum q_i = 1$
- Let ***α*** = [α_1, α_2, …,α_k] be a set of positive numbers that $\alpha 0 = \sum \alpha i$
  - ***α*** is called the hyper-parameter of Dirichlet distribution.
  - Specified by domain experts.
- $Dirichlet(\boldsymbol{q}|\boldsymbol{\alpha}) = \frac{\Gamma(\alpha_0)}{\Gamma(\alpha_1)\ldots\Gamma(\alpha_k)} \prod_{i=1}^{k} q_i^{\alpha_i - 1}$

Very similar to multinomial distribution?

$\Gamma(x)$ Is the Gamma function, and is (*x*-1)! If *x* is a positive integer

Latent Dirichlet Allocation (9/13)

Back to the MAP of multinomial distribution ***q***:

- Let ***D*** be a set of observed **samples** and [N_1, N_2, …, N_k] lists the number of times each outcome occurs.
  - *Ni* is the number of times *i* outcome occurs.
  - $N_0 = \sum \mathrm{Ni}$ be the number of multinomial experiments.
- Given ***D*** and the expert-defined hyper-parameters ***α***, we calculate the following posterior probability to select a good model parameter ***q***.

$$P(\boldsymbol{q} \mid D) \propto P(D \mid \boldsymbol{q}) P(\boldsymbol{q})$$

$$\propto \prod_{i=1}^{k} q_i^{N_i} \prod_{i=1}^{k} q_i^{\alpha_i - 1}$$

$$\propto \prod_{i=1}^{k} q_i^{N_i + \alpha_i - 1}$$

Latent Dirichlet Allocation (10/13)

The equation helps interpret the role of the hyper-parameters in the model estimate process.

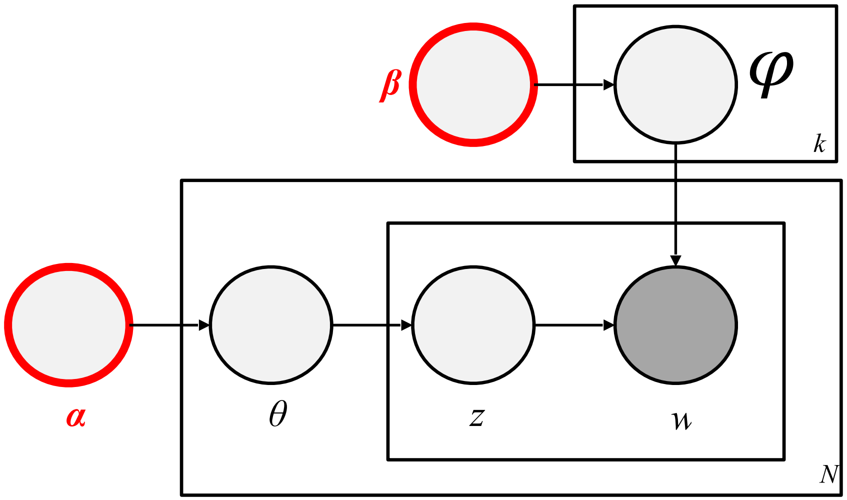
$$P(\boldsymbol{q} \mid D) \propto P(D \mid \boldsymbol{q}) P(\boldsymbol{q})$$
$$\propto \prod_{i=1}^{k} q_i^{N_i + \alpha_i - 1}$$

- The posterior of *q* is calculated by combining two datasets.
  - One is the **observed sample *D*** (i.e., [N_1, N_2, …, N_k]).
  - The other one is an **imaginary dataset** originated from our prior beliefs (i.e., $\boldsymbol{\alpha}$ = [α_1, α_2, …,α_k]).
  - Larger $\boldsymbol{\alpha}$ implies we have a higher confidence in our prior beliefs.

Latent Dirichlet Allocation (11/13)

The model parameters of LDA - θ^{d_i} and φ^{z} are all multinomial distributions.

- LDA utilizes **two** Dirichlet hyper-parameters ***α*** and ***β*** to incorporate priors into the model estimate process.
- Note that ***α*** and ***β*** are vectors of length *k* and *m*, respectively; but in many text mining packages, we only need to specify each a single value.



Latent Dirichlet Allocation (12/13)

LDA's parameter inference is complicated. One frequently used inference approach is based on **Gibbs sampling**:

- First randomly assign each word in the text corpus an integer in [1, k].
  - That is, to randomly assign a topic to each word.
- Then, sequentially examine the words and compute the following probability.

$$P(z_x = j \mid z_{-x}, W) \propto \frac{n_{-x,j}^{w_x} + \beta}{n_{-x,j}^{*} + m\beta} \times \frac{n_{-x,j}^{d_x} + \alpha}{n_{-x}^{d_x} + k\alpha}$$

  - Use the above topic distribution <u>to sample a topic for the current word</u>.

Latent Dirichlet Allocation (13/13)

With a sufficient number of sampling, model parameters are estimated as follows:

$$\theta_z^{d_i} = \frac{n_z^{d_i} + \alpha}{n_*^{d_i} + k\alpha} \quad \text{and} \quad \varphi_w^z = \frac{n_z^w + \beta}{n_z^* + m\beta}$$

This way, you represent a document as a k-dimensional vector.

The distribution would reveal what a topic is.

No worry too much, packages help you acquire the distribution with a few lines of code.

So …. Let’s practice

LDA + KNN Example (1/4)

LDA is based on multinomial distribution, and the input of `LatentDirichletAllocation` needs to be frequency vectors.

```
In [6]: from sklearn.feature_extraction.text import CountVectorizer

In [7]: TF_vectorizer = CountVectorizer(min_df=1, stop_words='english')

In [8]: TF_vectors = TF_vectorizer.fit_transform(texts)

In [9]: TF_vectors.shape

Out[9]: (3227, 52147)
```

LDA + KNN Example (2/4)

In [10]:
```
from sklearn.decomposition import LatentDirichletAllocation as LDA
```

In [11]:
```
lda_model = LDA(n_components = 10)
LDA_vectors = lda_model.fit_transform(TF_vectors)
```

In [12]:
```
LDA_vectors
```

Out[12]:
```
array([[1.26608181e-03, 1.26597368e-03, 1.26601590e-03, ...,
        1.26621815e-03, 1.26596600e-03, 1.26599974e-03],
       [6.21240699e-04, 4.10959484e-02, 1.23134743e-01, ...,
        2.34742462e-02, 6.21204360e-04, 6.21252836e-04],
       [2.02059803e-04, 2.02978137e-02, 2.02048953e-04, ...,
        2.02042545e-04, 1.71479939e-02, 2.02042095e-04],
       ...,
       [1.28231729e-03, 1.28245939e-03, 3.58765871e-01, ...,
        1.28230953e-03, 1.28218939e-03, 1.28231107e-03],
       [1.56289647e-03, 1.56261424e-03, 1.56274218e-03, ...,
        1.56283843e-03, 1.56316012e-03, 6.34210974e-01],
       [8.00232028e-04, 8.00100084e-04, 8.00038940e-04, ...,
        8.00058829e-04, 8.00143395e-04, 8.00119193e-04]])
```

In [13]:
```
LDA_vectors.shape
```

Out[13]:
```
(3227, 10)
```

In [14]:
```
lda_model.components_.shape
```

Out[14]:
```
(10, 52147)
```

You can specify α and β using parameters `doc_topic_prior` and `word_topic_prior`.

The default value of them is `1/n_components`

LDA + KNN Example (3/4)

```
In [28]: print_topics(lda_model, TF_vectorizer, 10)

Topic #0:
like just people know time life don love want think

Topic #1:
ironpython avr windows avrdude python using code microsoft make studio

Topic #2:
nbsp like time just people day new good blog year

Topic #3:
game cards just like new time use ve set don

Topic #4:
just like time ve really think good know going new

Topic #5:
la les et le poems vous en une space eigner

Topic #6:
sudan darfur abt said government peace trust people tat al

Topic #7:
new like com work design people time way site use

Topic #8:
like time just new said ve life blog know think

Topic #9:
time like just good little day food really make did
```

LDA + KNN Example (4/4)

In [17]:
```
x_train = LDA_vectors[0:2500]
x_test = LDA_vectors[2500:]
y_train = labels[0:2500]
y_test = labels[2500:]
```

In [twenty two]:
```
from sklearn import metrics
print(metrics.classification_report(expected_results, predicted_results))
```

```
            precision recall f1-score support

         F 0.59 0.57 0.58 370
         M 0.57 0.59 0.58 357

  accuracy 0.58 727
 macro avg 0.58 0.58 0.58 727
weighted avg 0.58 0.58 0.58 727
```

Summary

Now, we are able to extract topics from documents and represent each document/term by a topic vector!!

Is that good enough?? How about **POLYsemy**??

- *I walked into a bank aside a river bank.*
- SVD and LDA use a single topic vector for these two '*bank*', even though their meanings are totally different!!
- Polysemy may overestimate the similarity between documents.

Can we determine the meaning of a word according to its **context**??

- Here comes **BERT**!!!!!