臺北醫學大學
TAIPEI MEDICAL UNIVERSITY

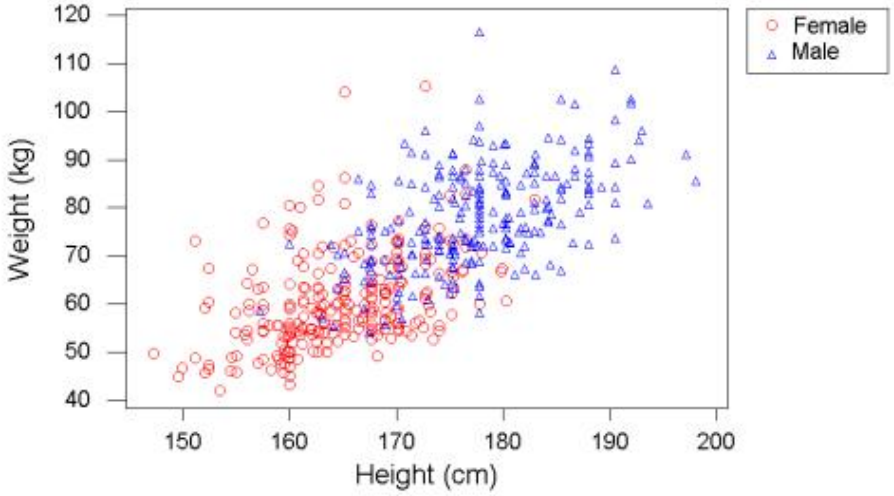
Data Representation
Term Weighting and Vector Space Model

Yung-Chun Chang, Ph.D.

Graduate Institute of Data Science, Taipei Medical University, Taiwan.
Email: changyc@tmu.edu.tw

How to represent each people? (1/2)

How to represent each people? (2/2)



How to represent users for online shopping recommendation?

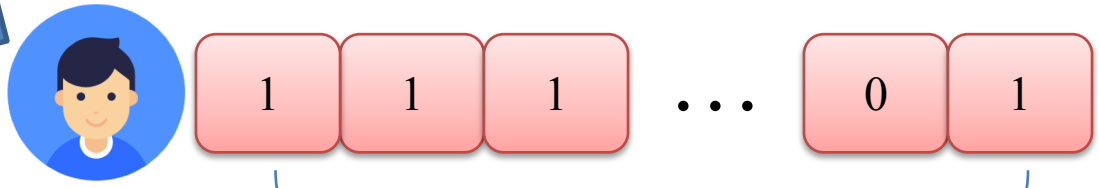
Online
Shopping

Vector Representation

- We can represent data as a Boolean vector.

User ID

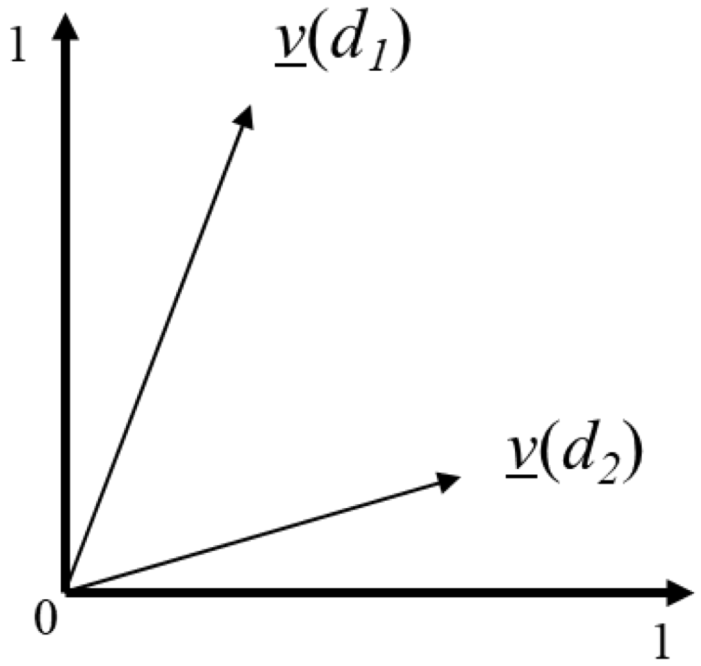
Item	User-1	User-2	User-3	User-4	User-5	User-6
Item1	1	1	0	0	0	1
Item2	1	1	0	1	0	0
Item3	1	1	0	1	1	1
Item4	0	1	0	0	0	0
Item5	1	0	0	0	0	0
Item6	1	0	1	1	1	1
Item7	1	0	1	1	1	0



items

Vector Space Model (1/4)

- We may view each data as a vector:



Vector Space Model (2/4)

– ***Cosine similarity***:

$$sim(d_1, d_2) = \frac{\underline{V}(d_1) \cdot \underline{V}(d_2)}{|\underline{V}(d_1)| \times |\underline{V}(d_2)|}$$

inner product of vectors

$$\sum \underline{V}(d_1)_t * \underline{V}(d_2)_t$$

vector length

$$\sqrt{\sum \underline{V}(d_1)_t * \underline{V}(d_1)_t}$$

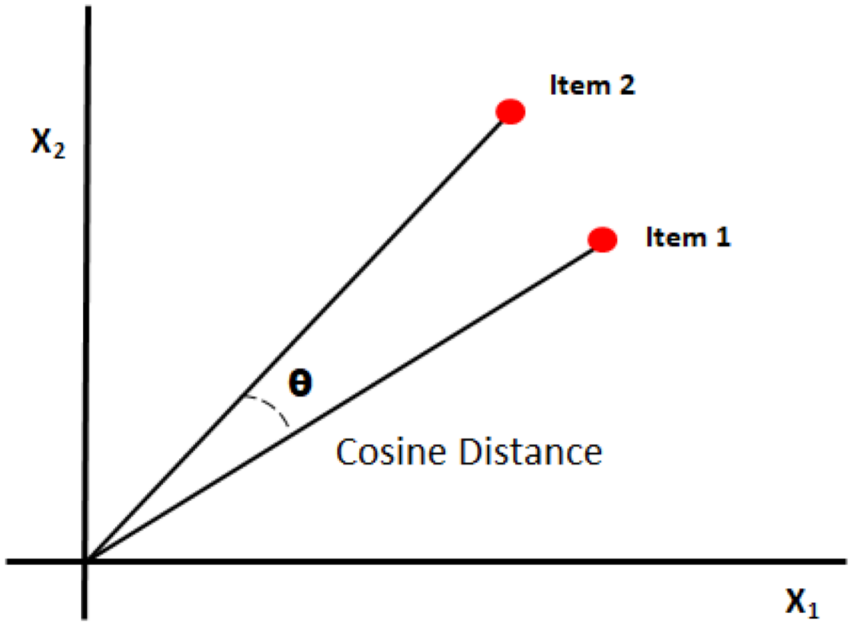
- The effect of the denominator is to normalized vectors to **unit vector**.
- The range of cosine similarity is on [0,1].
  - 1 → identical.
  - 0 → orthogonal.
- This measure is the cosine of the angle θ between the two vectors.

Vector Space Model (3/4)

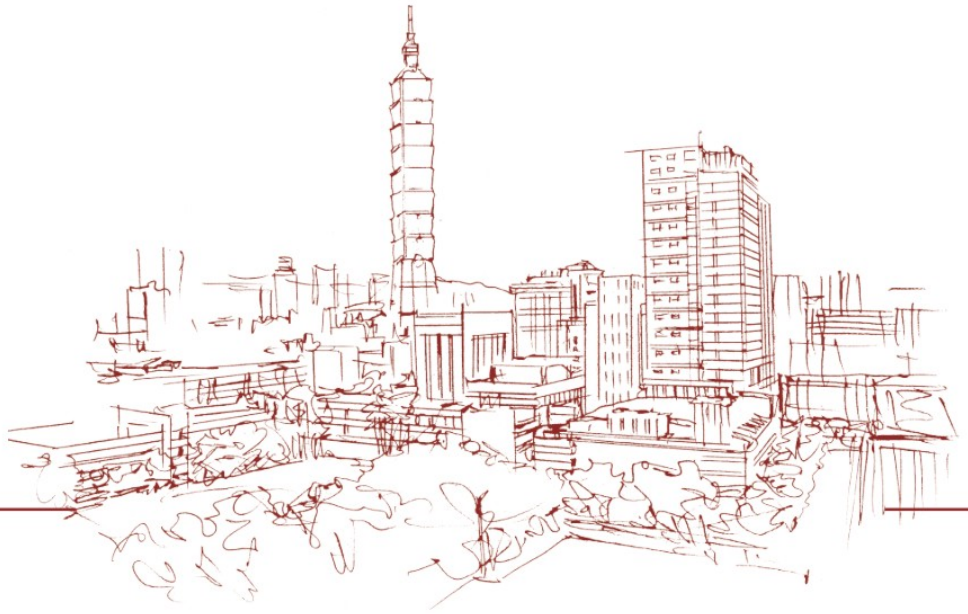
$$sim(d_1, d_2) = \frac{\underline{V}(d_1) \cdot \underline{V}(d_2)}{|\underline{V}(d_1)||\underline{V}(d_2)|}$$

$$= \frac{(0.81*0.81) + (0*3.51) + (2.02*0) + (0.56*0.56) + (0*0.36)}{\sqrt{(0.81^2 + 0 + 2.02^2 + 0.56^2 + 0)} * \sqrt{(0.81^2 + 3.51^2 + 0 + 0.56^2 + 0.36^2)}}$$

Vector Space Model (4/4)



https://kknews.cc/zh-tw/entertainment/kbym6p.html



Preprocessing and Representation of Unstructured Data

TAIPEI MEDICAL UNIVERSITY
臺北醫學大學
TAIPEI MEDICAL UNIVERSITY

Text Representation (1/4)

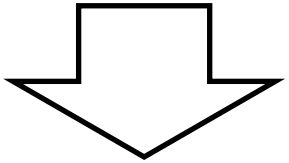
今日はいい日です　วันนี้เป็นวันที่ดี

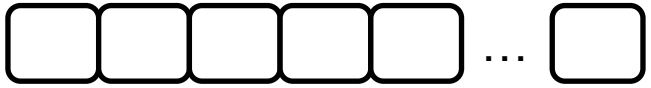
today is a good day　今天是美好的一天

aujourd'hui est un bon jour

今天是个好日子　오늘은 좋은 날이다

Human Readable: Language

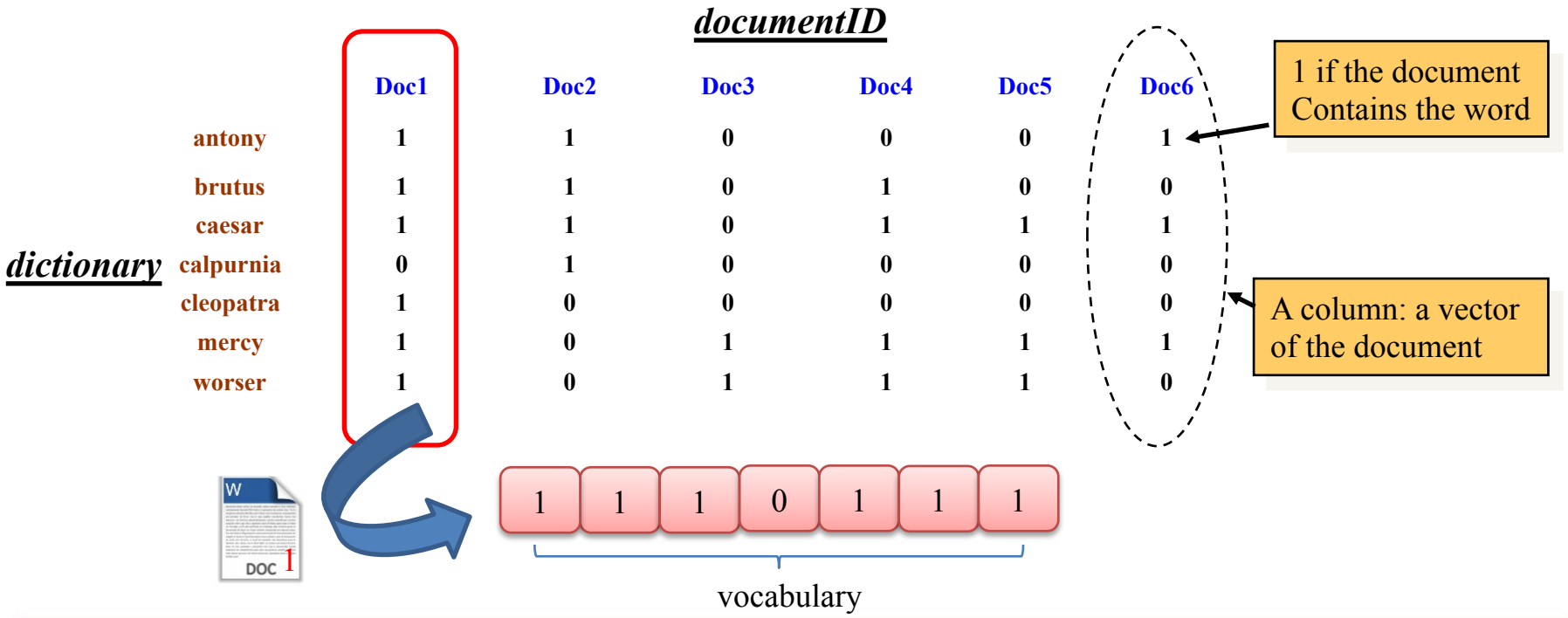




Machine Readable: Vector

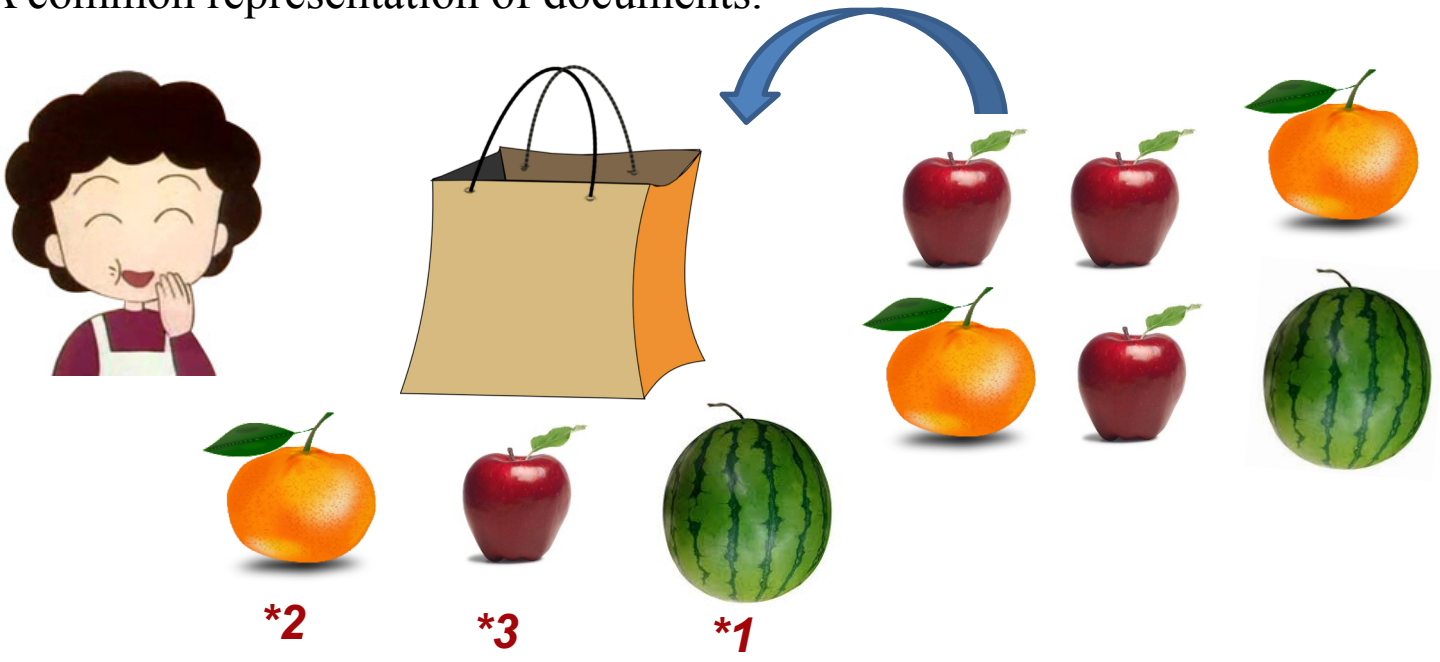
Text Representation (2/4)

- Major Steps in Index Construction
  - Collect the documents to be indexed.
  - ***Tokenize*** the text.
  - Do ***linguistic preprocessing*** of tokens.
  - ***Index*** the documents that each term occurs in.



Text Representation (3/4)

- ***Term Frequency*** **(*TF*)**:
  - The weight of a term depends on the number of occurrences of the term in the document.
  - Notation: $tf_{t,d}$ — the number of occurrences of term *t* in document *d*.
- The ***bag of words*** model:
  - A common representation of documents.



Text Representation (4/4)

- The ***bag of words*** model:
  - A common representation of documents.
  - The representation of a document *d* is the **set** of weights of its terms.
    - Example: *"term i and term j are synonyms"* → { <term,2>, <and, 1>, …}
    - Set: **the ordering of the terms is ignored**!!

can be term frequency or determined by other weighting schemes

Mary is quicker than John

John is quicker than Mary

"Mary is quicker than John" == "John is quicker than Mary"

Text Representation (5/5)

- Each fruit is considered equally important. But….


*2


*3


*1

Inverse Document Frequency (1/4)

- A critical problem of term frequency weighting scheme:
  - Each term occurrence is considered **equally important**.
  - *"term i and term j are synonyms"*

weight contribution: 1 1 1 1 1 1 1

- In fact, certain terms have little or **no discriminating power**.
  - For instance, a collection of documents on the auto industry is likely to have the term *'auto'* in almost every document.
  - **We need a mechanism for reducing the ef t occur too often in the collection**.

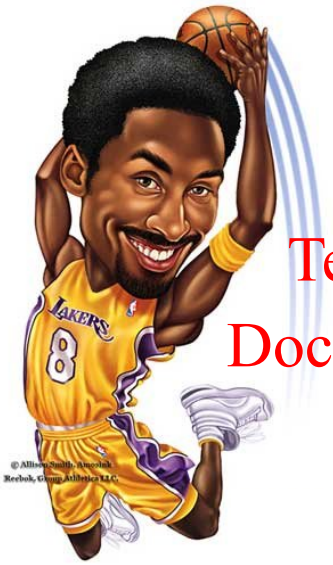
Inverse Document Frequency (2/4)

- ***Document frequency***:
  - Notation: df_t
  - The **number of documents** in the collection that contain a term *t*.

科比布萊恩今日率領湖人隊....，
這也讓科比布萊恩的生涯得分…

昨日攻下81分的科比布萊恩

科比布萊恩出席了NIKE舉辦的
夏令營活動，活動中科比布萊恩
與孩童有許多的互動。



Term Frequency?
Document Frequency?

Inverse Document Frequency (3/4)

- ***Inverse document frequency*** **(*IDF*)**:
  - Notation: $idf_t = \log \frac{N}{df_t}$ ← the number of documents in a collection
  - The *idf* of a rare term is high, and is likely to be low for a frequent term.

$idf_t \uparrow$ $df_t \downarrow$

$idf_t \downarrow$ $df_t \uparrow$

Inverse Document Frequency (4/4)

- Example of IDF values of terms in the Reuters dataset of 806,791 documents.

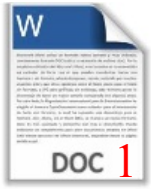
term	DF	IDF
'car'	18,165	1.65
'auto'	6,723	2.08
'insurance'	19,241	1.62
'best'	25,235	1.5

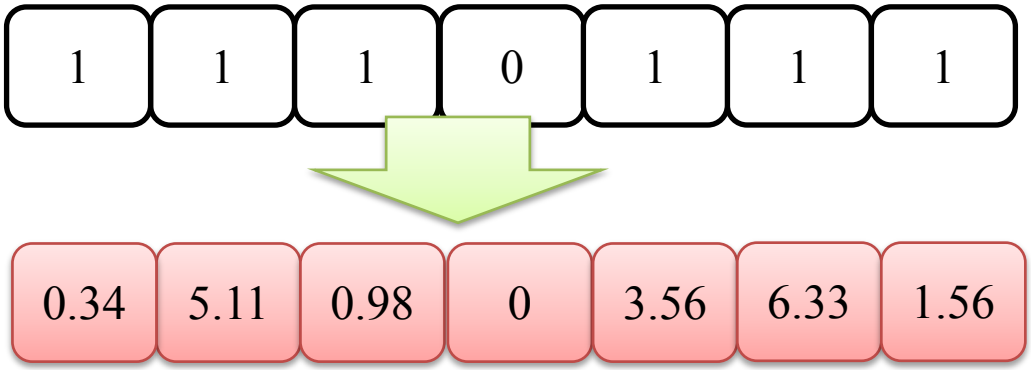
TF-IDF weighting (1/2)

- *TF-IDF* combines the concept of term frequency and inverse document frequency to assign the weight of term *t* in document *d* as follows:
  - $tf\text{-}idf_{t,d} = tf_{t,d} \times idf_t$.
  - The weight of term *t* in document *d* is:
    - **High**, when *t* occurs many times in *d* and appears within a small number of documents.
    - **Low**, when *t* is a rare term in *d* and occurs in virtually all documents in the collection.

TF-IDF weighting (2/2)

- A better way to represent a document is using weighted vector.
  - Bag-of-Words:
    - from sklearn.feature_extraction.text import CountVectorizer
    - vectorizer = **CountVectorizer**()
    - vectors_training = vectorizer.fit_transform(trainingData.data)

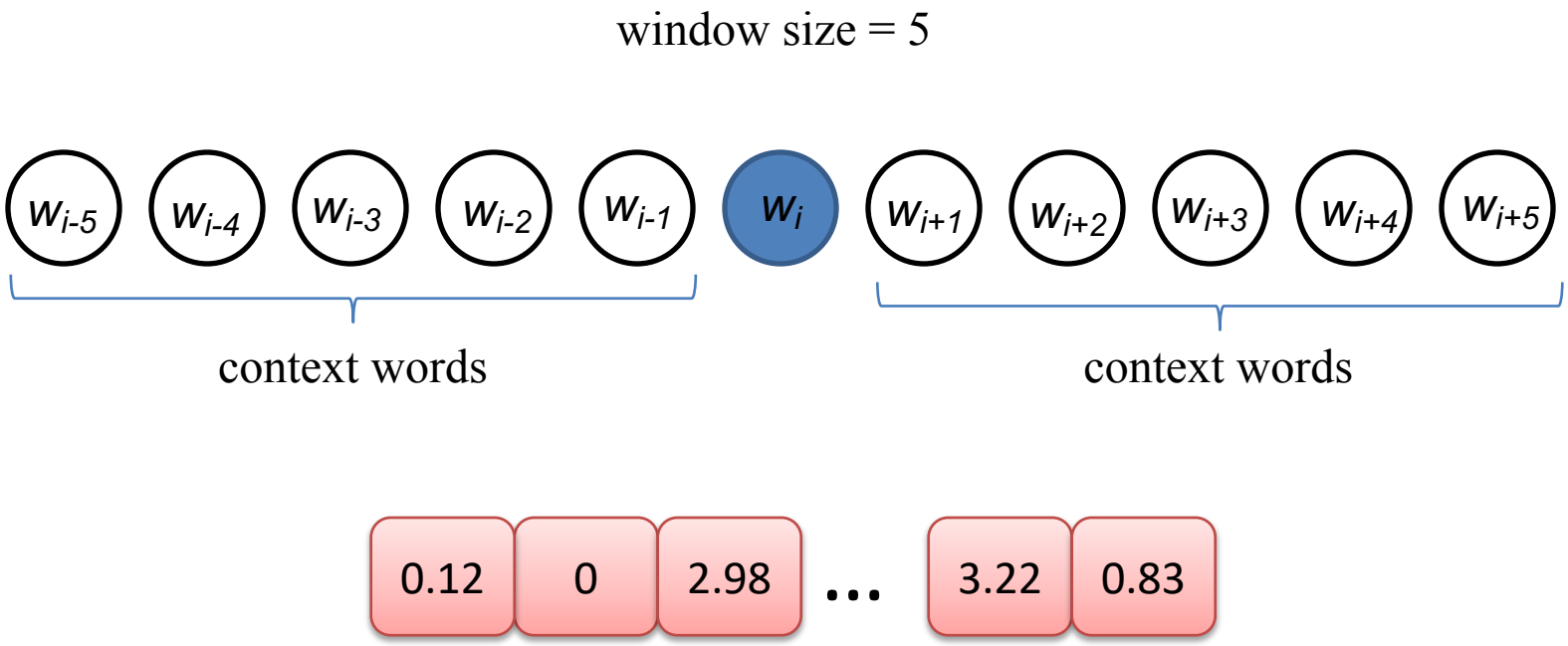




  - TF-IDF:
    - from sklearn.feature_extraction.text import TfidfVectorizer
    - vectorizer = **TfidfVectorizer**()
    - vectors_training = vectorizer.fit_transform(trainingData.data)

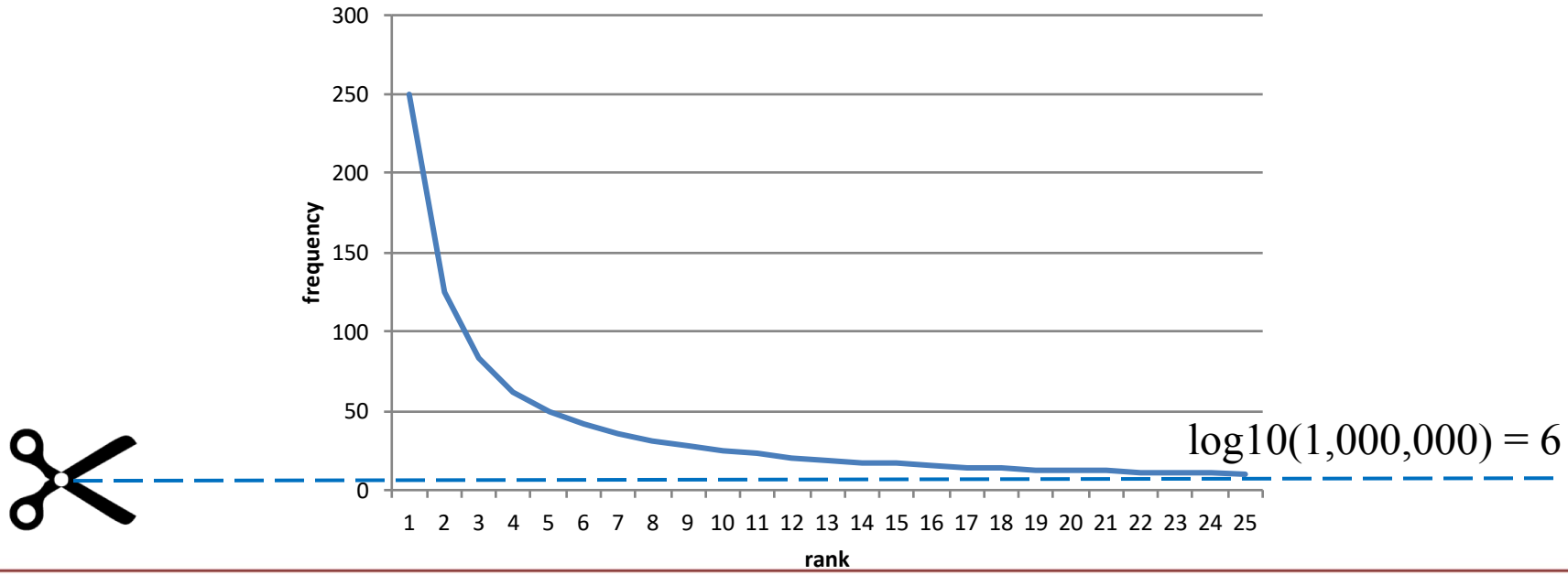
Word Representation

- How about the vector representation of a word?

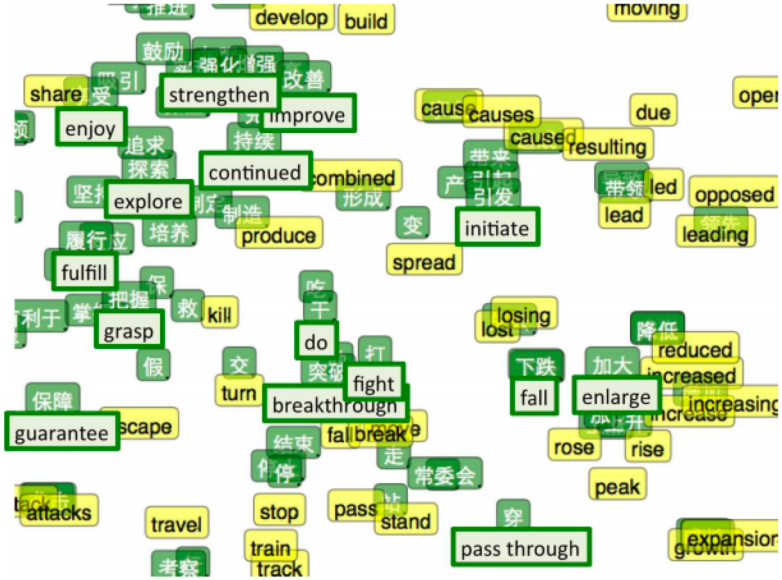


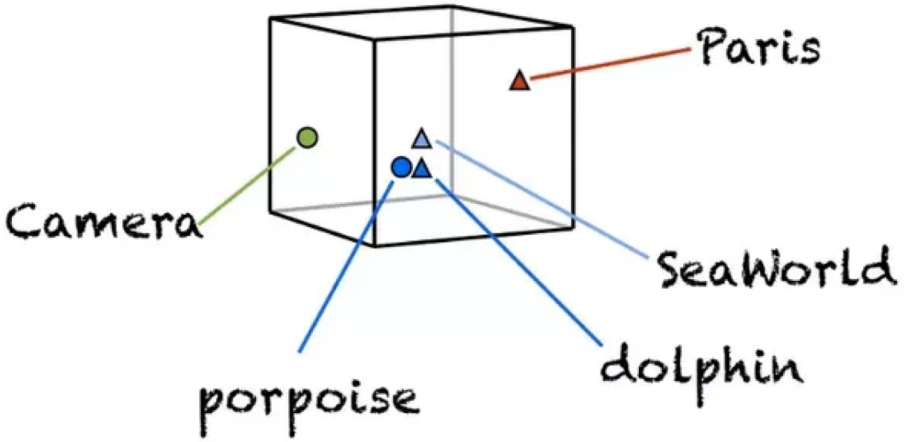
Dimension Reduction for Text Representation

- To avoid noise from rarely occuring words and reduce the size of the vectors, **we remove any feature with a count below a threshold** of log10(Σ) where Σ is the sum of all feature counts in the vector.



Advanced Text Representation – Word Embeddings





Mikolov 2013, Distributed Representations of Words and Phrases and their Compositionality, NIPS'13 Proceedings of the 26th International Conference on Neural Information Processing Systems, pp.3111-3119.

Advanced Text Representation – Word Embeddings (cont.)

The concept of CBOW is motivated by the distributional hypothesis [Miller and Charles 1991], which states that words with similar meanings often occur in similar contexts and thus suggests to look for word representations that capture their context distributions.

In contrast to the CBOW model, the SG model employs an inverse training objective for learning word representations with a simplified feed-forward neural network [Mikolov et al. 2013a].

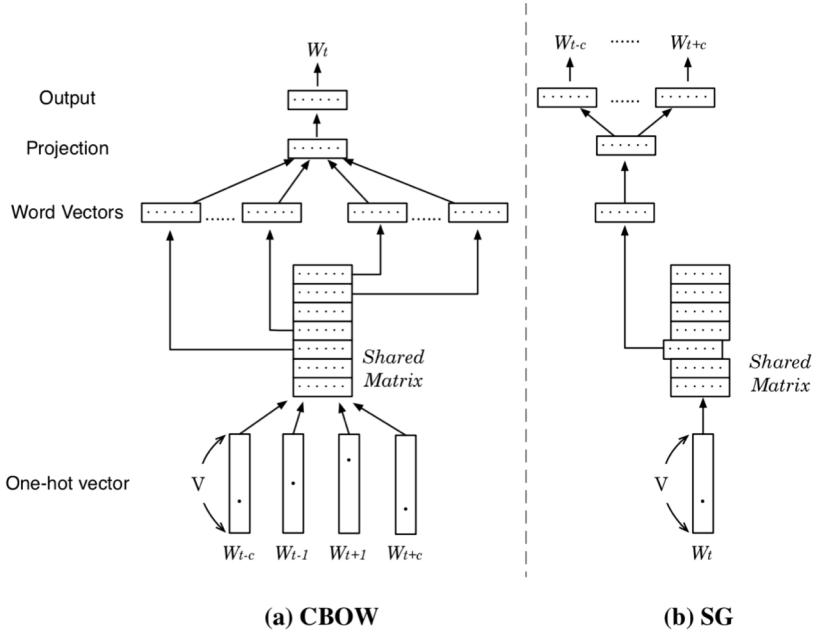


Fig. 1. (a) The CBOW model uses the context words $W_{t-c}, \cdots, W_{t+c}$ in the window as inputs to predict the current word W_t. (b) The SG model predicts words $W_{t-c}, \cdots, W_{t+c}$ in the context using the current word W_t as the input.

Mikolov et al. Distributed Representations of Words and Phrases and their Compositionality, NIPS 2013

Advanced Text Representation – Word Embeddings (cont.)

- Distributional hypothesis: two similar words have similar context.

「我們 去 六福村 玩 雲霄飛車 好不好」
「我們 去 麗寶樂園 玩 雲霄飛車 好不好」
「我們 去 九族文化村 玩 雲霄飛車 好不好」

https://fgc.stpi.narl.org.tw/activity/videoDetail/4b1141305ddf5522015de5479f4701b1

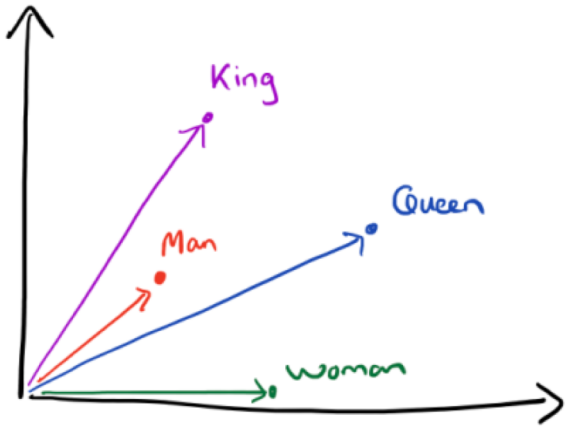
Advanced Text Representation – ELMo & BERT

Peter et al., “Deep contextualized word representations”, NAACL 2018

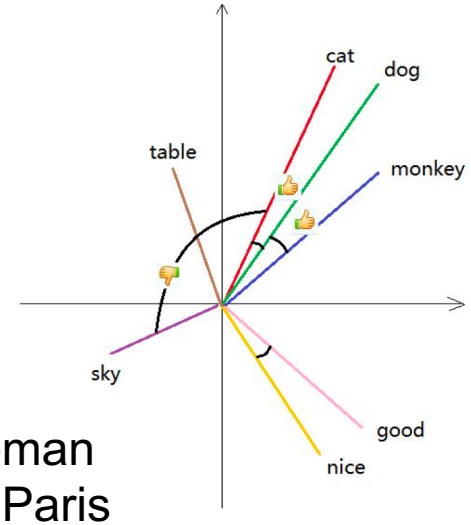
Devlin et al., “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”, NAACL 2018

Word2Vec

- Download word2vec from https://github.com/tmikolov/word2vec
- Run 'make' to compile word2vec tool
- Run the demo scripts: **./demo-word.sh** and **./demo-phrases.sh**



Word Vectors



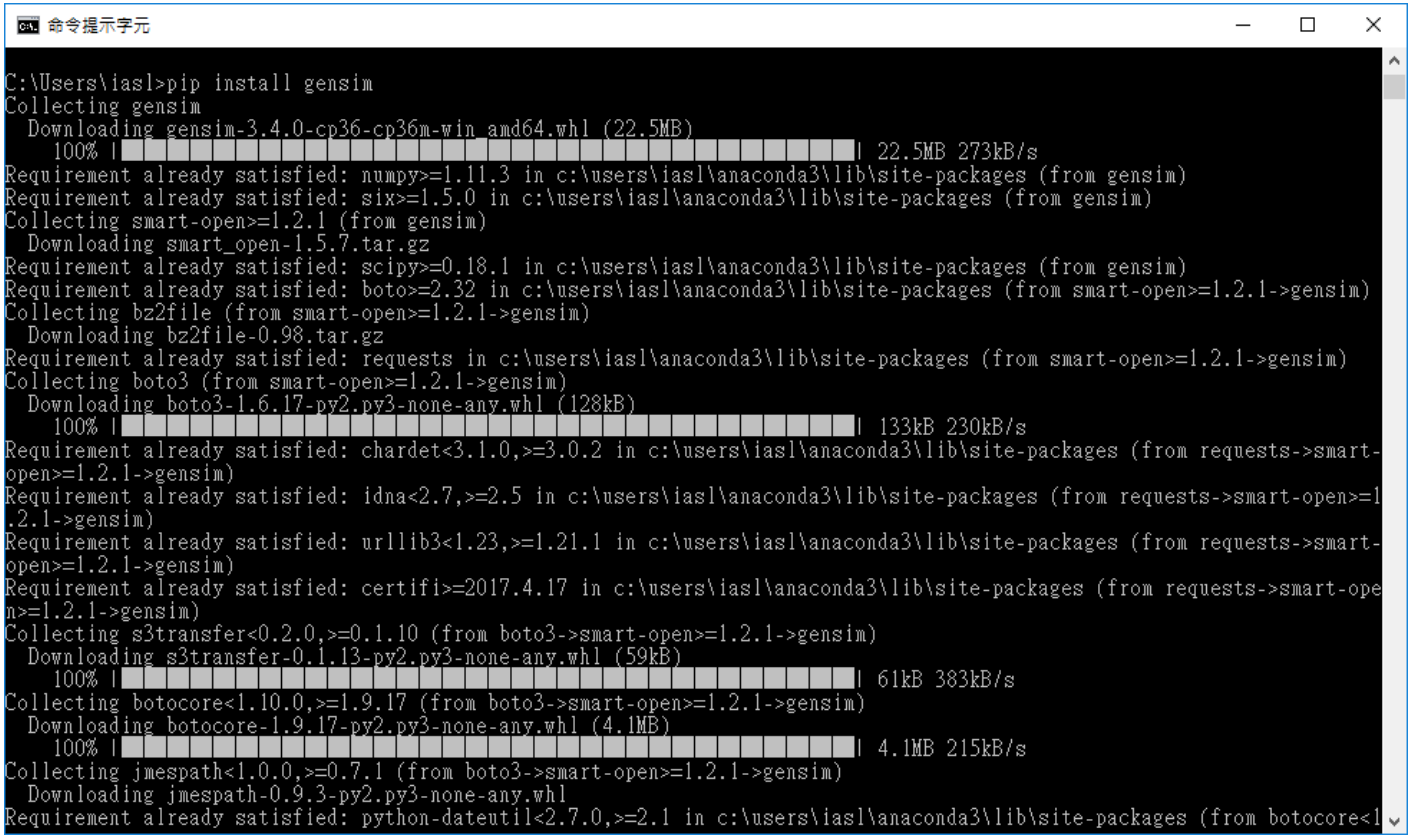
King + Queen = Man + Woman
Japan + Tokyo = France + Paris

Mikolov 2013, Distributed Representations of Words and Phrases and their Compositionality, NIPS'13 Proceedings of the 26th International Conference on Neural Information Processing Systems, pp.3111-3119.

https://code.google.com/archive/p/word2vec/

Word2Vec in Gensim (1/6)

- Install Gensim (pip install gensim)
- Download text8 for preparing training dataset



Word2Vec in Gensim (2/6)

```python
import logging
from gensim.models import word2vec

logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)
sentences = word2vec.LineSentence("text8")
model = word2vec.Word2Vec(sentences, size=300)
model.save("word2vec.model")
```

```
2018-03-27 16:24:32,373 : INFO : collecting all words and their counts
2018-03-27 16:24:34,011 : INFO : PROGRESS: at sentence #0, processed 0 words, keeping 0 word types
2018-03-27 16:24:37,177 : INFO : collected 253854 word types from a corpus of 17005207 raw words and 1701 sentences
2018-03-27 16:24:37,178 : INFO : Loading a fresh vocabulary
2018-03-27 16:24:37,417 : INFO : min_count=5 retains 71290 unique words (28% of original 253854, drops 182564)
2018-03-27 16:24:37,418 : INFO : min_count=5 leaves 16718844 word corpus (98% of original 17005207, drops 286363)
2018-03-27 16:24:37,596 : INFO : deleting the raw counts dictionary of 253854 items
2018-03-27 16:24:37,658 : INFO : sample=0.001 downsamples 38 most-common words
2018-03-27 16:24:37,660 : INFO : downsampling leaves estimated 12506280 word corpus (74.8% of prior 16718844)
2018-03-27 16:24:37,889 : INFO : estimated required memory for 71290 words and 300 dimensions: 206741000 bytes
2018-03-27 16:24:37,890 : INFO : resetting layer weights
2018-03-27 16:24:38,735 : INFO : training model with 3 workers on 71290 vocabulary and 300 features, using sg=0 hs=0 sample=0.0
01 negative=5 window=5
2018-03-27 16:24:40,438 : INFO : EPOCH 1 - PROGRESS: at 0.06% examples, 4558 words/s, in_qsize 5, out_qsize 0
2018-03-27 16:24:41,438 : INFO : EPOCH 1 - PROGRESS: at 5.53% examples, 255335 words/s, in_qsize 6, out_qsize 0
2018-03-27 16:24:42,446 : INFO : EPOCH 1 - PROGRESS: at 10.99% examples, 367755 words/s, in_qsize 6, out_qsize 0
2018-03-27 16:24:43,456 : INFO : EPOCH 1 - PROGRESS: at 16.34% examples, 430861 words/s, in_qsize 5, out_qsize 0
2018-03-27 16:24:44,457 : INFO : EPOCH 1 - PROGRESS: at 21.58% examples, 469937 words/s, in_qsize 5, out_qsize 0
2018-03-27 16:24:45,459 : INFO : EPOCH 1 - PROGRESS: at 26.98% examples, 501397 words/s, in_qsize 6, out_qsize 0
2018-03-27 16:24:46,460 : INFO : EPOCH 1 - PROGRESS: at 32.33% examples, 524127 words/s, in_qsize 6, out_qsize 0
2018-03-27 16:24:47,466 : INFO : EPOCH 1 - PROGRESS: at 37.21% examples, 534375 words/s, in_qsize 5, out_qsize 0
```

Word2Vec in Gensim (3/6)

- gensim.models.word2vec.Word2Vec(

sentences=None, size=100, alpha=0.025, window=5, min_count=5, max_vocab_size=None, sample=0.001, seed=1, workers=3, min_alpha=0.0001, sg=0, hs=0, negative=5, cbow_mean=1, hashfxn=<built-in function hash>, iter=5, null_word=0, trim_rule=None, sorted_vocab=1, batch_words=10000)

Parameters:

- **sentences** (*iterable of iterables*) – The *sentences* iterable can be simply a list of lists of tokens, but for larger corpora, consider an iterable that streams the sentences directly from disk/network. See `BrownCorpus`, `Text8Corpus` or `LineSentence` in `word2vec` module for such examples. If you don't supply *sentences*, the model is left uninitialized – use if you plan to initialize it in some other way.
- **sg** (*int {1, 0}*) – Defines the training algorithm. If 1, skip-gram is employed; otherwise, CBOW is used.
- **size** (*int*) – Dimensionality of the feature vectors.
- **window** (*int*) – The maximum distance between the current and predicted word within a sentence.
- **alpha** (*float*) – The initial learning rate.
- **min_alpha** (*float*) – Learning rate will linearly drop to *min_alpha* as training progresses.
- **seed** (*int*) – Seed for the random number generator. Initial vectors for each word are seeded with a hash of the concatenation of word + *str(seed)*. Note that for a fully deterministically-reproducible run, you must also limit the model to a single worker thread (*workers=1*), to eliminate ordering jitter from OS thread scheduling. (In Python 3, reproducibility between interpreter launches also requires use of the *PYTHONHASHSEED* environment variable to control hash randomization).
- **min_count** (*int*) – Ignores all words with total frequency lower than this.
- **max_vocab_size** (*int*) – Limits the RAM during vocabulary building; if there are more unique words than this, then prune the infrequent ones. Every 10 million word types need about 1GB of RAM. Set to *None* for no limit.
- **sample** (*float*) – The threshold for configuring which higher-frequency words are randomly downsampled, useful range is (0, 1e-5).
- **workers** (*int*) – Use these many worker threads to train the model (=faster training with multicore machines).

https://radimrehurek.com/gensim/models/word2vec.html

Word2Vec in Gensim (4/6)

```
1 res = model.most_similar('love', topn = 10)
2 for item in res:
3     print(item[0] + "," + str(item[1]))
```

```
affection,0.634809136390686
loving,0.6115965843200684
me,0.5884506702423096
passion,0.5881558656692505
lover,0.5817869305610657
praise,0.581366777420044
grace,0.5750666856765747
soul,0.5710604190826416
thee,0.5646052360534668
my,0.5626602172851562
```

Word2Vec in Gensim (5/6)

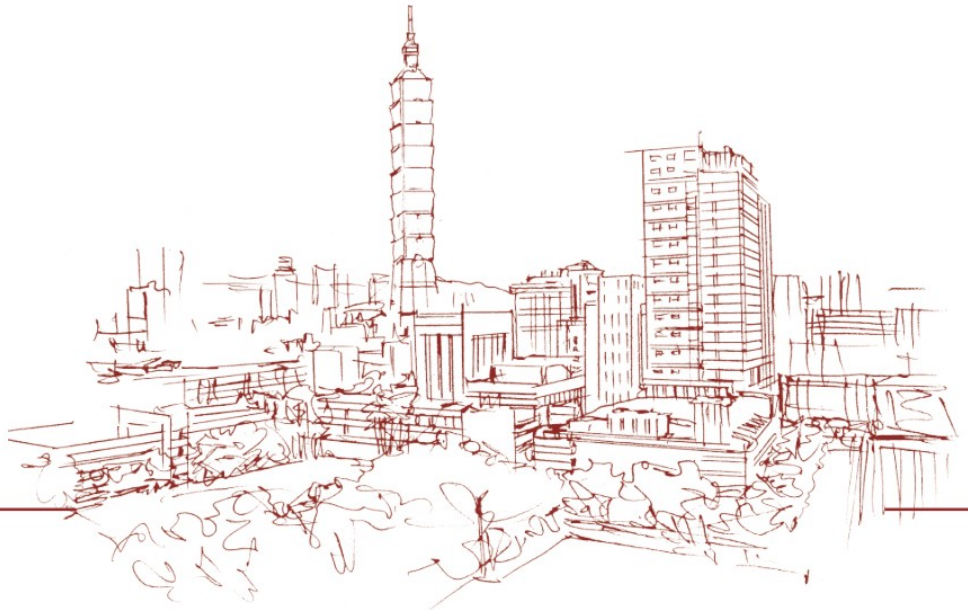
```
print("%s AND %s，LIKE %s AND... " % ('King' , 'Queen', 'Man'))
res = model.most_similar(['king','queen'], ['man'], topn= 10)
for item in res:
    print(item[0]+","+str(item[1]))
```

```
King AND Queen，LIKE Man AND...
prince,0.6096096634864807
aragon,0.5717663168907166
elizabeth,0.5674647092819214
burgundy,0.5611557364463806
vii,0.5567941665649414
dukes,0.5523853302001953
elector,0.544865608215332
scotland,0.5448097586631775
iii,0.5444616079330444
viii,0.5433887243270874
```

Word2Vec in Gensim (6/6)

```
1 print("Calculate Cosine Similarity")
2 res = model.similarity('king', 'prince')
3 print(res)
```

```
Calculate Cosine Similarity
0.67029500256
```



Thanks for your listening

Q & A

TAIPEI MEDICAL UNIVERSITY
臺北醫學大學
TAIPEI MEDICAL UNIVERSITY