Text Summarization and Information Extraction

Text and Web Mining (H6751)

School of Communication and Information

Text Summarization

- The goal of text summarization is to get an idea of key important topics and summarize huge documents of information into a few lines that can be read, understood, and interpreted.
- The end goal is to be able to make well informed decisions in shorter timeframes through efficient and scalable processes and techniques that can be performed on text data.
- We will be looking at following methods:
    - Keyphrase Extraction
    - Topic Modeling
    - Information Extraction
        - Named Entities Recognition (NER)
        - Relationship Extraction

Keyphrase Extraction

- Often the starting point for carrying out more complex text mining processing where the output act as features for the more complex models.

- Two major techniques:
  - Collocations
  - Weighted tag-based phrase extraction

- Phrases – a collection of words standing together as a conceptual unit.
  - Noun phrase – "the movie" [head word – movie]
  - Verb phrase – "we all laughed" [head word – laughed]
  - Prepositional phrase – "with her" [head word – with]

Keyphrase Extraction

- Collocation - a phrase consisting of more than one word but these words more commonly co-occur in a given context than its individual word parts. E.g., 'CT scan' is more likely to co-occur than do 'CT' and 'scan' individually.
- How do we make good selections for collocations? Co-occurrences may not be sufficient as phrases such as 'of the' may co-occur frequently, but are not meaningful.

Keyphrase Extraction

- Technique (1) – construct n-grams out of a corpus and count frequency of each n-gram and rank them based on their frequency of occurrence to get the **most frequent n-grams** collocations.

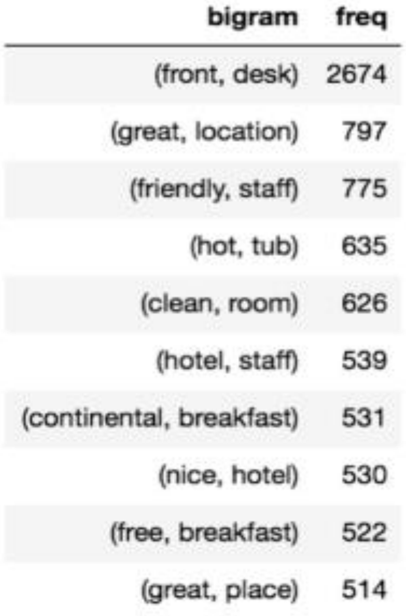
- Using any adjacent words as bigram or trigrams may not get meaningful phrases. E.g., the sentence ‘He uses social media’ contains bigrams: ‘He uses’, ‘uses social’, ‘social media’. ‘He uses’ and ‘uses social’ do not mean anything, while ‘social media’ is a meaningful bigram.

- Technique (2) **Pointwise Mutual Information** – measure of association between two features (x, y). Ratio of x and y occurring together under a joint distribution under the assumption that x and y are independent.

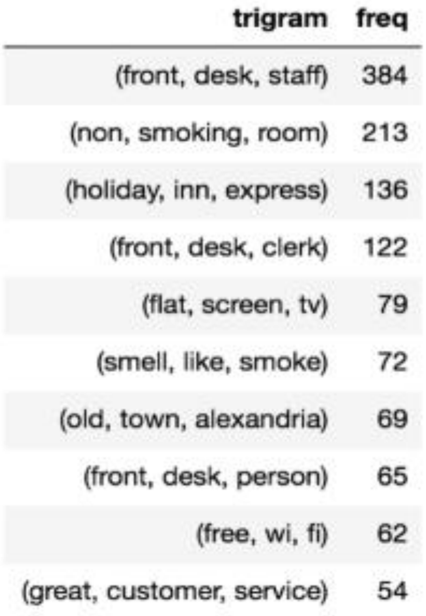
$$pmi(x,y) = log \frac{p(x,y)}{p(x)p(y)}$$

Collocations

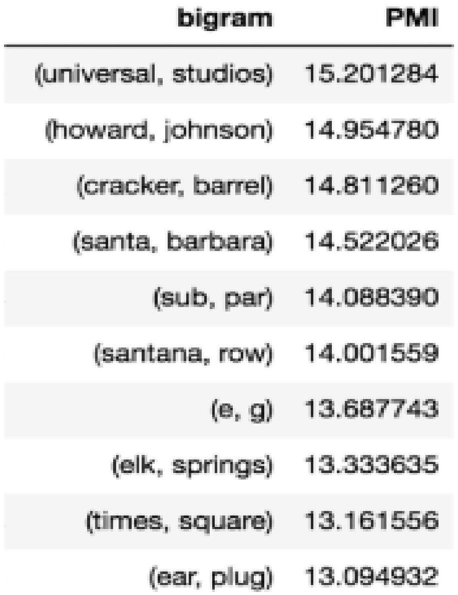
- Hotel Reviews Dataset: https://www.kaggle.com/datafiniti/hotel-reviews/data
- Choice of **adjacent words** as bigram or trigrams may not always yield meaningful phrases
- It can be seen **PMI** picks up bigrams and trigrams that consist of words that should co-occur together.



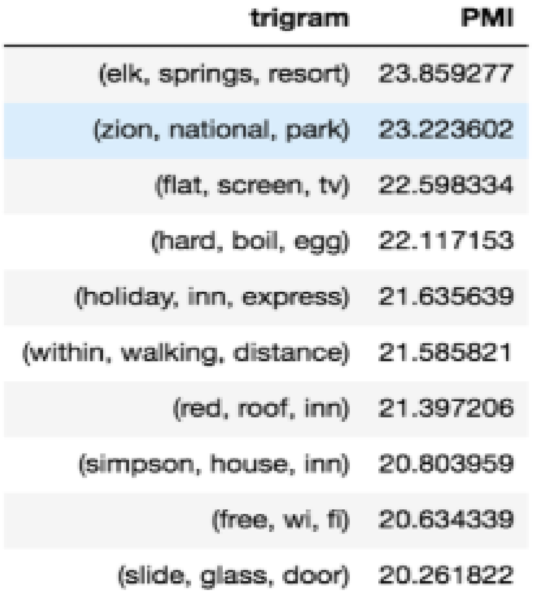
bigram	freq
(front, desk)	2674
(great, location)	797
(friendly, staff)	775
(hot, tub)	635
(clean, room)	626
(hotel, staff)	539
(continental, breakfast)	531
(nice, hotel)	530
(free, breakfast)	522
(great, place)	514



trigram	freq
(front, desk, staff)	384
(non, smoking, room)	213
(holiday, inn, express)	136
(front, desk, clerk)	122
(flat, screen, tv)	79
(smell, like, smoke)	72
(old, town, alexandria)	69
(front, desk, person)	65
(free, wi, fi)	62
(great, customer, service)	54



bigram	PMI
(universal, studios)	15.201284
(howard, johnson)	14.954780
(cracker, barrel)	14.811260
(santa, barbara)	14.522026
(sub, par)	14.088390
(santana, row)	14.001559
(e, g)	13.687743
(elk, springs)	13.333635
(times, square)	13.161556
(ear, plug)	13.094932



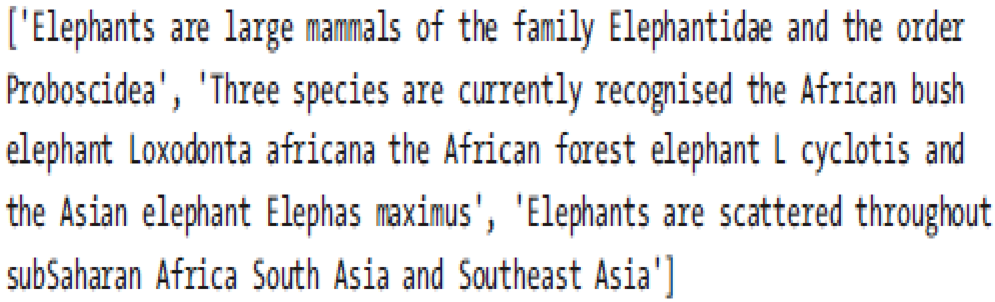
trigram	PMI
(elk, springs, resort)	23.859277
(zion, national, park)	23.223602
(flat, screen, tv)	22.598334
(hard, boil, egg)	22.117153
(holiday, inn, express)	21.635639
(within, walking, distance)	21.585821
(red, roof, inn)	21.397206
(simpson, house, inn)	20.803959
(free, wi, fi)	20.634339
(slide, glass, door)	20.261822

Weighted Tag-Based Phrase Extraction

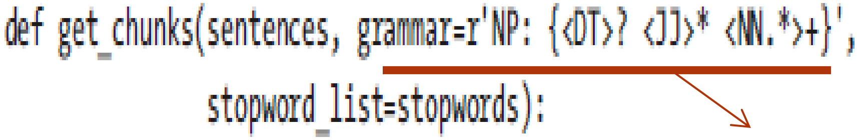
- Technique –
  1. Extract all **noun phrase** chunks using shallow parsing - based on parts of speech (POS) tags patterns.

Sample text

https://github.com/dipanjanS/text-analytics-with-python



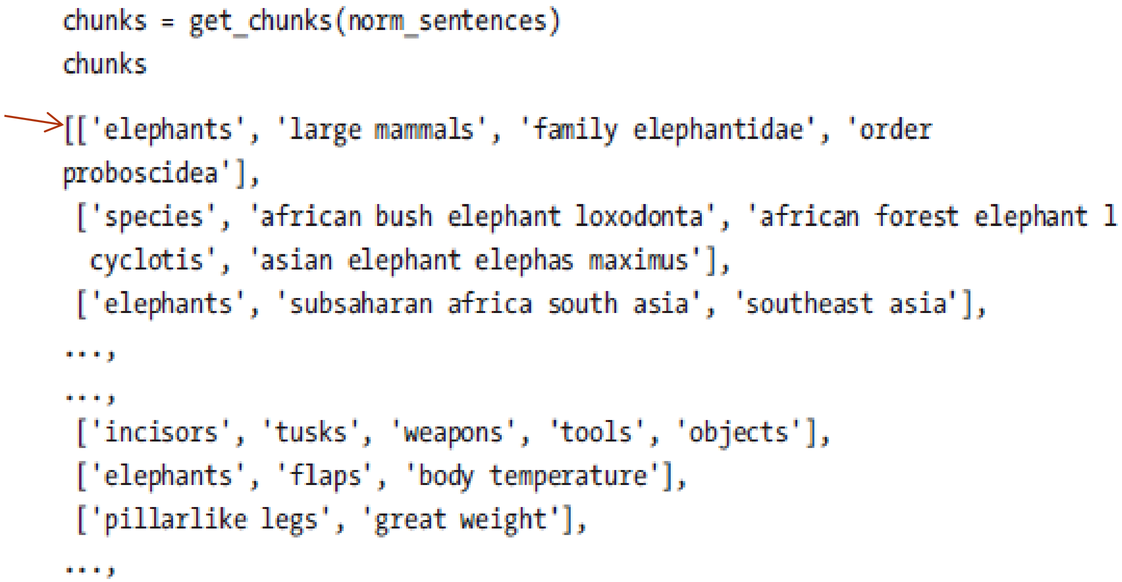
```
['Elephants are large mammals of the family Elephantidae and the order
Proboscidea', 'Three species are currently recognised the African bush
elephant Loxodonta africana the African forest elephant L cyclotis and
the Asian elephant Elephas maximus', 'Elephants are scattered throughout
subSaharan Africa South Asia and Southeast Asia']
```



```
def get_chunks(sentences, grammar=r'NP: {<DT>? <JJ>* <NN.*>+}',
                     stopword_list=stopwords):
```

Extract NP using regex

Noun phrases

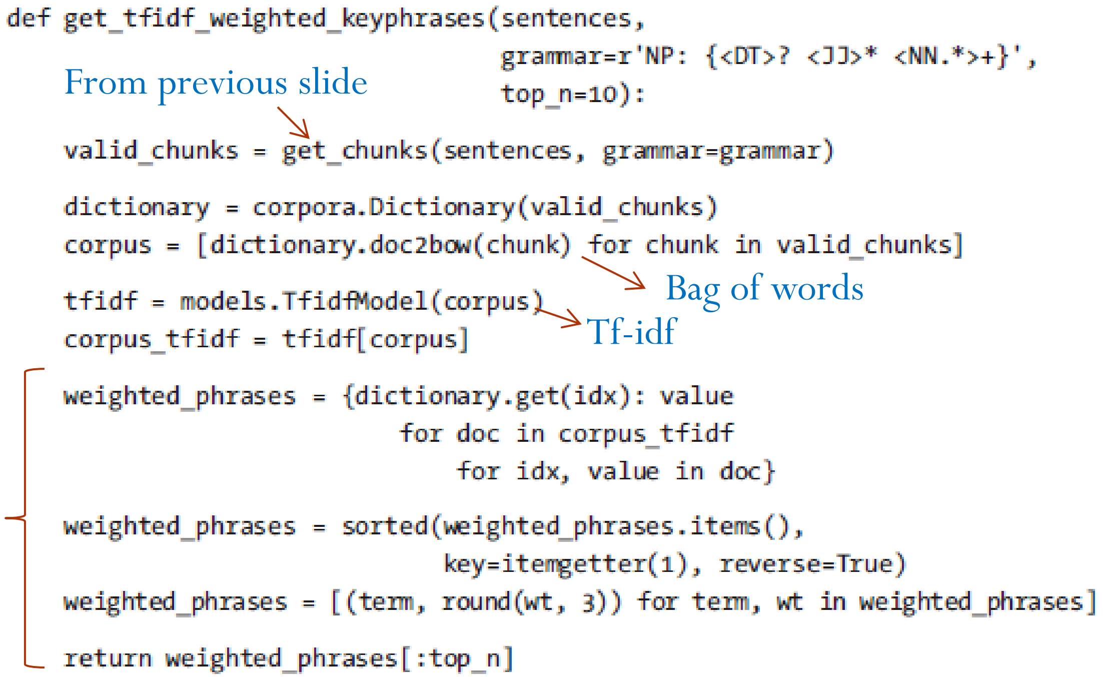


```
chunks = get_chunks(norm_sentences)
chunks

[['elephants', 'large mammals', 'family elephantidae', 'order
proboscidea'],
 ['species', 'african bush elephant loxodonta', 'african forest elephant l
  cyclotis', 'asian elephant elephas maximus'],
 ['elephants', 'subsaharan africa south asia', 'southeast asia'],
...,
...,
 ['incisors', 'tusks', 'weapons', 'tools', 'objects'],
 ['elephants', 'flaps', 'body temperature'],
 ['pillarlike legs', 'great weight'],
...,
```

Weighted Tag-Based Phrase Extraction

- Technique –
  1. Extract all **noun phrase** chunks using shallow parsing - based on parts of speech (POS) tags patterns.
  2. Compute TF-IDF weights for each chunk and return the top weighted phrases.

Better description of elephants



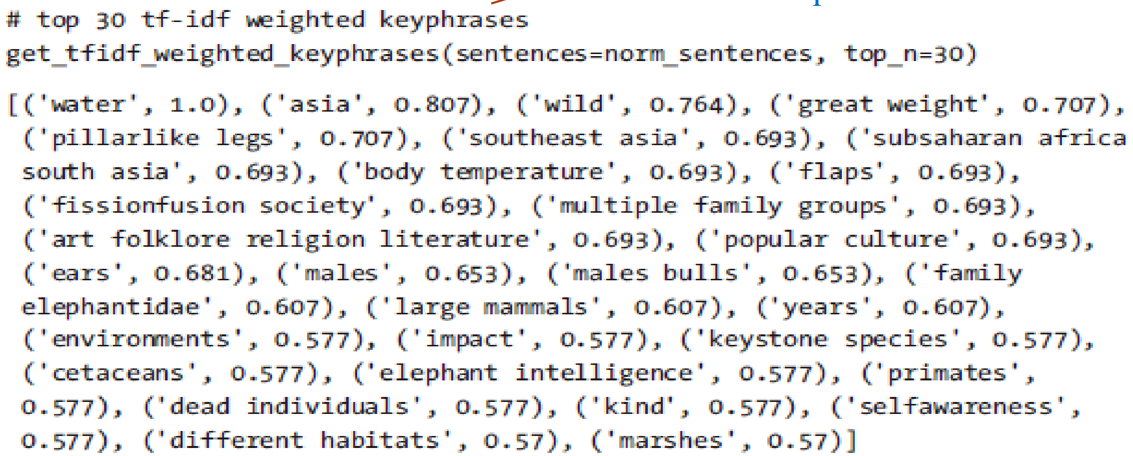
```
def get_tfidf_weighted_keyphrases(sentences,
                                  grammar=r'NP: {<DT>? <JJ>* <NN.*>+}',
                                  top_n=10):

    valid_chunks = get_chunks(sentences, grammar=grammar)

    dictionary = corpora.Dictionary(valid_chunks)
    corpus = [dictionary.doc2bow(chunk) for chunk in valid_chunks]

    tfidf = models.TfidfModel(corpus)
    corpus_tfidf = tfidf[corpus]

    weighted_phrases = {dictionary.get(idx): value
                           for doc in corpus_tfidf
                               for idx, value in doc}

    weighted_phrases = sorted(weighted_phrases.items(),
                              key=itemgetter(1), reverse=True)
    weighted_phrases = [(term, round(wt, 3)) for term, wt in weighted_phrases]

    return weighted_phrases[:top_n]
```

From previous slide

Bag of words

Tf-idf

Based on noun phrases from sentences



```
# top 30 tf-idf weighted keyphrases
get_tfidf_weighted_keyphrases(sentences=norm_sentences, top_n=30)

[('water', 1.0), ('asia', 0.807), ('wild', 0.764), ('great weight', 0.707),
 ('pillarlike legs', 0.707), ('southeast asia', 0.693), ('subsaharan africa
 south asia', 0.693), ('body temperature', 0.693), ('flaps', 0.693),
 ('fissionfusion society', 0.693), ('multiple family groups', 0.693),
 ('art folklore religion literature', 0.693), ('popular culture', 0.693),
 ('ears', 0.681), ('males', 0.653), ('males bulls', 0.653), ('family
 elephantidae', 0.607), ('large mammals', 0.607), ('years', 0.607),
 ('environments', 0.577), ('impact', 0.577), ('keystone species', 0.577),
 ('cetaceans', 0.577), ('elephant intelligence', 0.577), ('primates',
 0.577), ('dead individuals', 0.577), ('kind', 0.577), ('selfawareness',
 0.577), ('different habitats', 0.57), ('marshes', 0.57)]
```

Topic Modeling

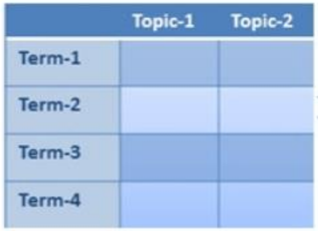
- Topic Modeling – a method of uncovering hidden structure in a collection of text (corpus). It entails:
- Dimensionality Reduction
  - Rather than representing a text in its feature (words) space, text is represented in its **topic (feature) space**
- Unsupervised Learning
  - Similar to clustering – as in the case of clustering, the **number of topics**, like the number of clusters, is a **hyperparameter**.
  - Topic modeling build **clusters of words** rather than **clusters of documents**. A **document is thus a mixture of all the topics**, each having a certain weight.
- A form of tagging
  - If document classification is assigning a single category to a text, topic modeling is **assigning tags to a text**. A human expert can label the resulting topics with human-readable labels and use heuristics to convert the weighted topics to a set of tags.

Topic Modeling

- Applications:
  - Text Classification – improve performance by **grouping similar words together in topics** rather than using each word as a feature.
  - Recommender System – using **similarity measure to recommend** articles for readers based on a topic similar to the articles the readers had already read.
  - Uncovering Themes in Texts – e.g., **detect trends** in online publications.
- Algorithms
  - LSA or LSI – **Latent Semantic Analysis** or **Latent Semantic Indexing** uses **Singular Value Decomposition (SVD)** on the Document-Term matrix. Based on Linear Algebra.
  - **Non-Negative Matrix Factorization** – based on Linear Algebra.
  - LDA – **Latent Dirichlet Allocation** – based on Probabilistic Graphical Models.

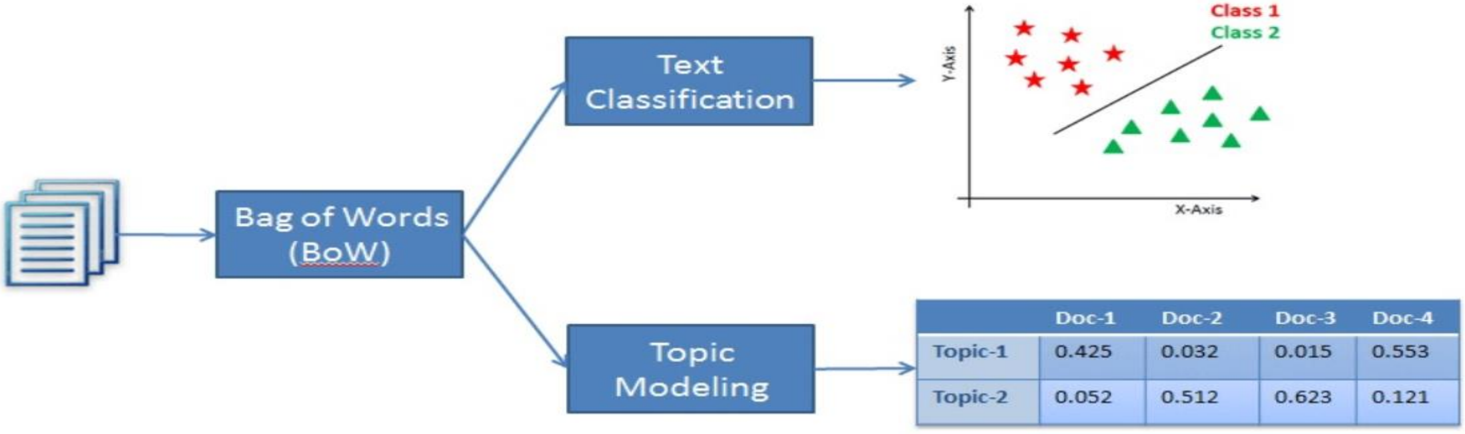
Comparison Between Text Classification and Topic Modeling

- Text classification is a supervised machine learning problem, where a text document or article classified into a **pre-defined** set of classes.
- Topic modeling is the process of discovering groups of **co-occurring** related words in text documents, which makes up the "topics".



	Topic-1	Topic-2
Term-1		
Term-2		
Term-3		
Term-4		

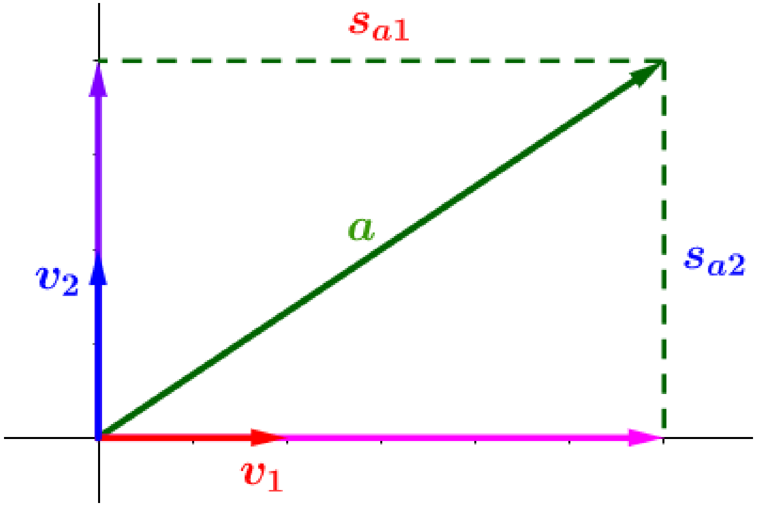
- A form of unsupervised learning, set of possible topics are unknown.
- Topic modeling can be used to solve the text classification problem, by identifying the topics in a document" used for further classification.



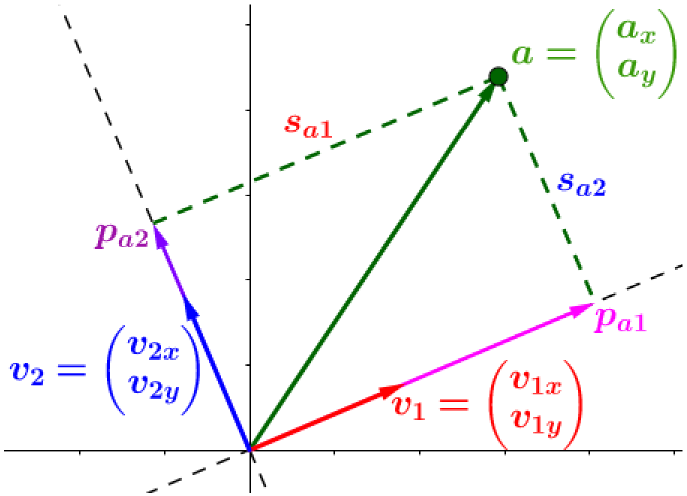
	Doc-1	Doc-2	Doc-3	Doc-4
Topic-1	0.425	0.032	0.015	0.553
Topic-2	0.052	0.512	0.623	0.121

Singular Value Decomposition

- When a vector (***a***) is decomposed, the following information is derived:
  - The **directions** of projection - the **unit** vector v_1 and v_2 **representing the directions** (x and y axes or any other orthogonal axes) onto which the vector is projected (decomposed).
  - The **lengths** of projection (the **line segments** s_{a1} and s_{a2}) - how much of the vector is **contained** in each direction of projection (more of vector ***a*** is leaning on the direction $\mathbf{v_1}$ than it is on $\mathbf{v_2}$, hence $s_{a1}>s_{a2}$).



SVD



$a = \begin{pmatrix} a_x \\ a_y \end{pmatrix}$
s_{a1}
s_{a2}
p_{a2}
p_{a1}
$v_2 = \begin{pmatrix} v_{2x} \\ v_{2y} \end{pmatrix}$
$v_1 = \begin{pmatrix} v_{1x} \\ v_{1y} \end{pmatrix}$

Same figure as before, but tilting the axes of projection to convince you they aren't confined to x and y. (a_x and a_y are the coordinates of vector **a**, put into a column matrix (aka column vector), as per convention. Same for v_1 and v_2).

- Projection is done by the **dot product** — it gives us the **lengths** of projection (s_{a1} and s_{a2}):

$$a^T \cdot v_1 = \begin{pmatrix} a_x & a_y \end{pmatrix} \cdot \begin{pmatrix} v_{1x} \\ v_{1y} \end{pmatrix} = s_{a1}$$

$$a^T \cdot v_2 = \begin{pmatrix} a_x & a_y \end{pmatrix} \cdot \begin{pmatrix} v_{2x} \\ v_{2y} \end{pmatrix} = s_{a2}$$

Projecting (a) onto v1 and v2.

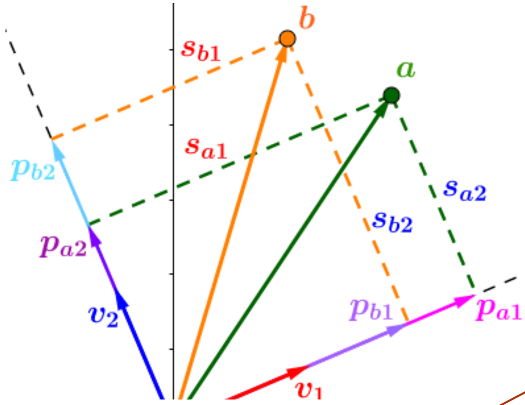
SVD

$$a^T \cdot V = \begin{pmatrix} a_x & a_y \end{pmatrix} \cdot \begin{pmatrix} v_{1x} & v_{2x} \\ v_{1y} & v_{2y} \end{pmatrix} = \begin{pmatrix} s_{a1} & s_{a2} \end{pmatrix}$$

...to write both equations in one go, by adding an **extra column** for each unit vector.

$$A \cdot V = \begin{pmatrix} a_x & a_y \\ b_x & b_y \end{pmatrix} \cdot \begin{pmatrix} v_{1x} & v_{2x} \\ v_{1y} & v_{2y} \end{pmatrix} = \begin{pmatrix} s_{a1} & s_{a2} \\ s_{b1} & s_{b2} \end{pmatrix} = S$$

...by adding an **extra row** for each point. **S** is the matrix containing the lengths of projections.



A = Matrix of data points

V = Matrix of unit vectors

$$A{\cdot}V = \underset{n \times d}{\begin{pmatrix} a_x & a_y & \cdots \\ b_x & b_y & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}} \cdot \underset{d \times d}{\begin{pmatrix} v_{1x} & v_{2x} & \cdots \\ v_{1y} & v_{2y} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}} = \underset{n \times d}{\begin{pmatrix} s_{a1} & s_{a2} & \cdots \\ s_{b1} & s_{b2} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}} = S$$

S = Matrix of projection lengths

n = no. of points, **d** = no. of dimensions, **A** = matrix containing points, **V** = matrix containing the decomposition axes, **S** = matrix containing lengths of projection.

SVD

- Now the act of projecting the dataset using SVD becomes a snap, since **all the points are *already projected*** (decomposed) on all the principal components (the ***v_i*** unit vectors):

$$A \cdot V = S$$

Matrix of points

The dot product performs the projection

Matrix of decomposition axes

Matrix of the lengths of projections

The dot product in this case is just **ordinary matrix multiplication.**

$$\begin{pmatrix} a_x & a_y & \cdots \\ b_x & b_y & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} = A = SV^T = \begin{pmatrix} s_{a1} & s_{a2} & \cdots \\ s_{b1} & s_{b2} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \begin{pmatrix} v_{1x} & v_{2x} & \cdots \\ v_{1y} & v_{2y} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}^T$$

The dataset

The projection lengths of the dataset on 1st principal component

The projection lengths of the dataset on 2nd principal component

1st principal component

2nd principal component

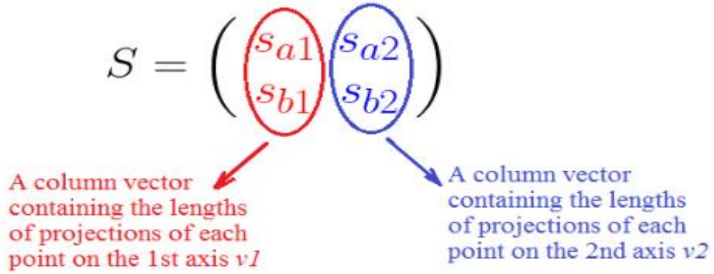
A.V = S => A.V.V^{-1} = SV^{-1} => A = SV^T ,

V^{-1} = V^T property of orthogonal vectors => V contains orthogonal columns (principal components)

SVD

$$A = SV^T$$

- Singular Value Decomposition formula: $A = U\ \Sigma\ V^T$
- Need to match: $S = U\ \Sigma$



$$S = \begin{pmatrix} s_{a1} & s_{a2} \\ s_{b1} & s_{b2} \end{pmatrix}$$

A column vector containing the lengths of projections of each point on the 1st axis *v1*

A column vector containing the lengths of projections of each point on the 2nd axis *v2*

- **normalize** these column vectors, i.e. make them of **unit length** by dividing each vector column vector by its magnitude, but in matrix form.

$$S = \begin{pmatrix} s_{a1} & s_{a2} \\ s_{b1} & s_{b2} \end{pmatrix}$$

$$\text{Magnitude of 1st column} = \sigma_1 = \sqrt{(s_{a1})^2 + (s_{b1})^2}$$

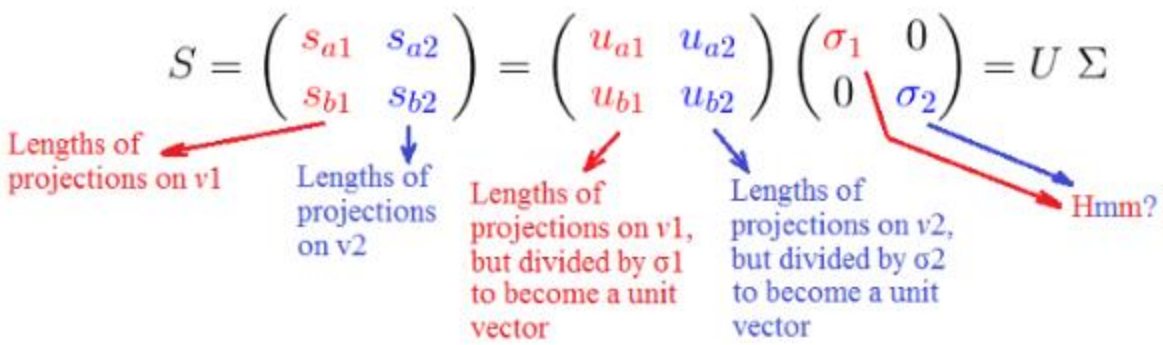
$$\text{Magnitude of 2nd column} = \sigma_2 = \sqrt{(s_{a2})^2 + (s_{b2})^2}$$

$$S = \begin{pmatrix} \frac{s_{a1}}{\sigma_1} & \frac{s_{a2}}{\sigma_2} \\ \frac{s_{b1}}{\sigma_1} & \frac{s_{b2}}{\sigma_2} \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix} = \underset{\substack{\big\downarrow \\ U}}{\begin{pmatrix} u_{a1} & u_{a2} \\ u_{b1} & u_{b2} \end{pmatrix}} \underset{\substack{\big\downarrow \\ \Sigma}}{\begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix}}$$

$$S = \begin{pmatrix} \frac{s_{a1}}{\sigma_1} & \frac{s_{a2}}{\sigma_2} \\ \frac{s_{b1}}{\sigma_1} & \frac{s_{b2}}{\sigma_2} \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix} = \begin{pmatrix} s_{a1} & s_{a2} \\ s_{b1} & s_{b2} \end{pmatrix}$$

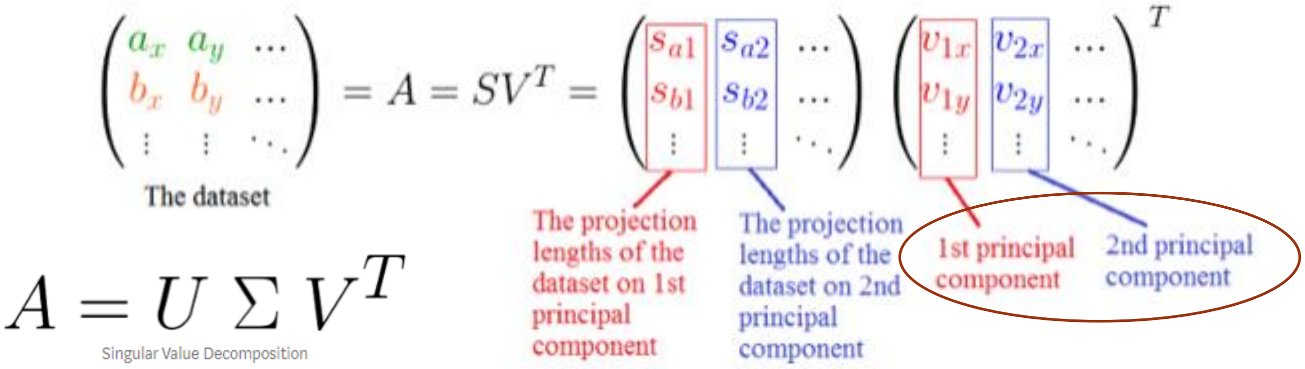
$$M = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix} = \begin{pmatrix} 2 & 3 \\ 2 & 3 \end{pmatrix}$$

SVD


$S = \begin{pmatrix} s_{a1} & s_{a2} \\ s_{b1} & s_{b2} \end{pmatrix} = \begin{pmatrix} u_{a1} & u_{a2} \\ u_{b1} & u_{b2} \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix} = U\,\Sigma$
Lengths of projections on v1
Lengths of projections on v2
Lengths of projections on v1, but divided by σ1 to become a unit vector
Lengths of projections on v2, but divided by σ2 to become a unit vector
Hmm?

(σ_i) is the **square root of the sum of squared projection lengths,** of all points, onto the ith unit vector $\boldsymbol{v_i}$.

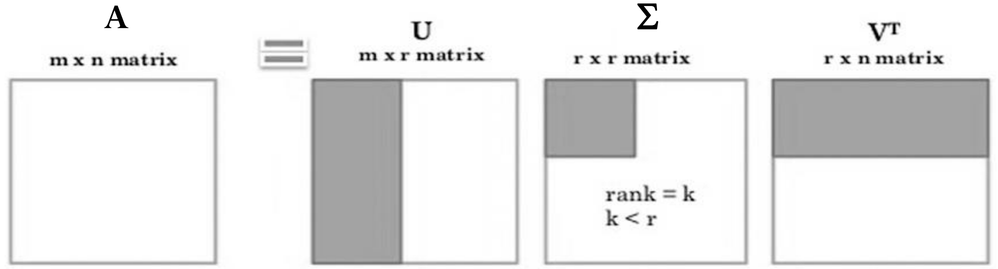
- Since the **sigmas** (Σ) contain the sum of projection lengths onto a specific axis, **they represent how close all the points are to that axis** => **influence each terms has on each topic.**
- E.g. if $\sigma_1 > \sigma_2$, then most points are closer to $\boldsymbol{v_1}$ than $\boldsymbol{v_2}$, and vice versa.


$\begin{pmatrix} a_x & a_y & \cdots \\ b_x & b_y & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} = A = SV^T = \begin{pmatrix} s_{a1} & s_{a2} & \cdots \\ s_{b1} & s_{b2} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \begin{pmatrix} v_{1x} & v_{2x} & \cdots \\ v_{1y} & v_{2y} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}^T$
The dataset
The projection lengths of the dataset on 1st principal component
The projection lengths of the dataset on 2nd principal component
1st principal component
2nd principal component
$A = U\,\Sigma\,V^T$
Singular Value Decomposition

SVD

Original Matrix : $A = U \times \Sigma \times V^T$

m = number of terms
n = number of docs
k or r = number of topics



Low Rank Matrix : $A_k = U_k \times \Sigma_k \times V^T{}_k$

- **k** singular values are the "topics" where the **number of topics (k)** is predefined and the original matrix **A** is decomposed into **U**, **Σ**, and **V^T** using SVD.
- **A** is the ***term-document*** matrix and is obtained after feature engineering on the preprocessed text data, where each row of the matrix represents a term and each column represents a text document.
- **U** is the ***term-topic*** matrix where each row of the matrix represents a term and each column represents a topic. It is used to get the **influential terms for each topic** when U is multiplied by the singular values in **Σ**.
- **Σ** is the matrix or array that consists of the ***singular values*** obtained after low-rank SVD, which is typically equal to the number of topics decided prior.
- **V^T** is the ***topic-document*** matrix, when after transpose, gives the ***document-topic*** matrix, which is used to determine the **influence each topic has on each document**.

SVD – example

- Imagine a matrix A whose columns represent movies and the rows different users. The entries of the matrix are numbers 0 to 5 where 0 means a user does not like a certain movie and 5 means they really like a given movie as illustrated below:

$$A = \overset{\text{movies}}{\left[\begin{array}{ccc|cc} 1 & 1 & 1 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 2 & 0 & 4 & 4 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 1 & 0 & 2 & 2 \end{array}\right]} \text{users}$$

- Now imagine that the first 3 columns are the movies Avengers, StarWars and IronMan respectively(Sci-Fi movies). While the last 2 columns are the movies Titanic and Notebook (Romance movies).

SVD – example

- After performing SVD on matrix A we get the matrices U**Σ**V as illustrated below(using a tool like scikit-learn):

Topics (Movie Categories)

Terms (Users)

$$U = \begin{bmatrix} 0.13 & 0.02 & -0.01 \\ 0.41 & 0.07 & -0.03 \\ 0.55 & 0.09 & -0.04 \\ 0.68 & 0.11 & -0.05 \\ 0.15 & -0.59 & 0.65 \\ 0.07 & -0.73 & -0.67 \\ 0.07 & -0.29 & -0.32 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 12.4 & 0 & 0 \\ 0 & 9.5 & 0 \\ 0 & 0 & 1.3 \end{bmatrix} \quad V^T = \begin{bmatrix} 0.56 & 0.59 & 0.56 & 0.09 & 0.09 \\ 0.12 & -0.02 & 0.12 & -0.69 & -0.69 \\ 0.40 & -0.80 & 0.40 & 0.09 & 0.09 \end{bmatrix}$$

- The first column of U represents weights that would match each **user's preference** to movies categorized under **Sci-Fi** while the second column of U represents weights that would match each user's preference to movies under the R**omance** category.
- For example, the first user greatly prefers sci-fi movies(0.13 score) compared to romance(0.02 score). KIV third column for now.

SVD – example

Topics (Movie Categories)

Terms (Users)

$$U = \begin{bmatrix} 0.13 & 0.02 & -0.01 \\ 0.41 & 0.07 & -0.03 \\ 0.55 & 0.09 & -0.04 \\ 0.68 & 0.11 & -0.05 \\ 0.15 & -0.59 & 0.65 \\ 0.07 & -0.73 & -0.67 \\ 0.07 & -0.29 & -0.32 \end{bmatrix}$$

Singular Values

$$\Sigma = \begin{bmatrix} 12.4 & 0 & 0 \\ 0 & 9.5 & 0 \\ 0 & 0 & 1.3 \end{bmatrix}$$

Documents (Movies)

Topics (Movie Categories)

$$V^T = \begin{bmatrix} 0.56 & 0.59 & 0.56 & 0.09 & 0.09 \\ 0.12 & -0.02 & 0.12 & -0.69 & -0.69 \\ 0.40 & -0.80 & 0.40 & 0.09 & 0.09 \end{bmatrix}$$

And for **Σ**,

- The **first diagonal entry** represents the **weight** of the **Sci-Fi** category and the **second diagonal entry** represents the **weight** of the **romance** category.

And for V^T,

- The columns depict the **degree** to which a **movie** belongs to a **category**. So, for example, we can see from the first column of **V^T** that the first movie(this would be **Avengers**) belongs heavily to the **Sci-Fi** (0.56 score) category and very little to the romance category (0.12 score).

SVD - example

$$\Sigma = \begin{bmatrix} 12.4 & 0 & 0 \\ 0 & 9.5 & 0 \\ 0 & 0 & 1.3 \end{bmatrix}$$

- Looking at matrix **Σ**, the third diagonal entry which represents the weight of a movie category has a small value(1.3 score). This is understandable because there are only two categories of movies. So most of the third dimension is considered as noise.
- Hence, perform dimensionality reduction to matrix A by eliminating the third dimension of **Σ**, this would also mean eliminating the third column of U and the third row of V to produce the following new U, **Σ** and, V:

$$U = \begin{bmatrix} 0.13 & 0.02 \\ 0.41 & 0.07 \\ 0.55 & 0.09 \\ 0.68 & 0.11 \\ 0.15 & -0.59 \\ 0.07 & -0.73 \\ 0.07 & -0.29 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 12.4 & 0 \\ 0 & 9.5 \end{bmatrix} \quad V^T = \begin{bmatrix} 0.56 & 0.59 & 0.56 & 0.09 & 0.09 \\ 0.12 & -0.02 & -0.12 & -0.69 & -0.69 \end{bmatrix}$$

SVD vis-à-vis PCA

- PCA – computes the eigenvalues and eigenvectors of the **covariance matrix** (max variance describes data most), which is the **product $XX^{\top}$**, where X is the data matrix:
  - $\underset{\textbf{Covariance matrix}}{XX^{\top}} = \overset{\text{Eigenvectors}}{W}\overset{\text{Eigenvalues}}{\mathbf{D}}W^{\top}$ (eigenvalue decomposition – one of the other possible method)
- On the other hand, applying SVD to the data matrix X as follows:

$$X = \mathbf{U\Sigma V^{\top}}$$

and constructing the covariance matrix from SVD decomposition gives

$$XX^{\top} = (\mathbf{U\Sigma V^{\top}})(\mathbf{U\Sigma V^{\top}})^{\top}$$

transpose

$$XX^{\top} = (U\Sigma \mathbf{V^{\top}})(\mathbf{V}\Sigma U^{\top})$$

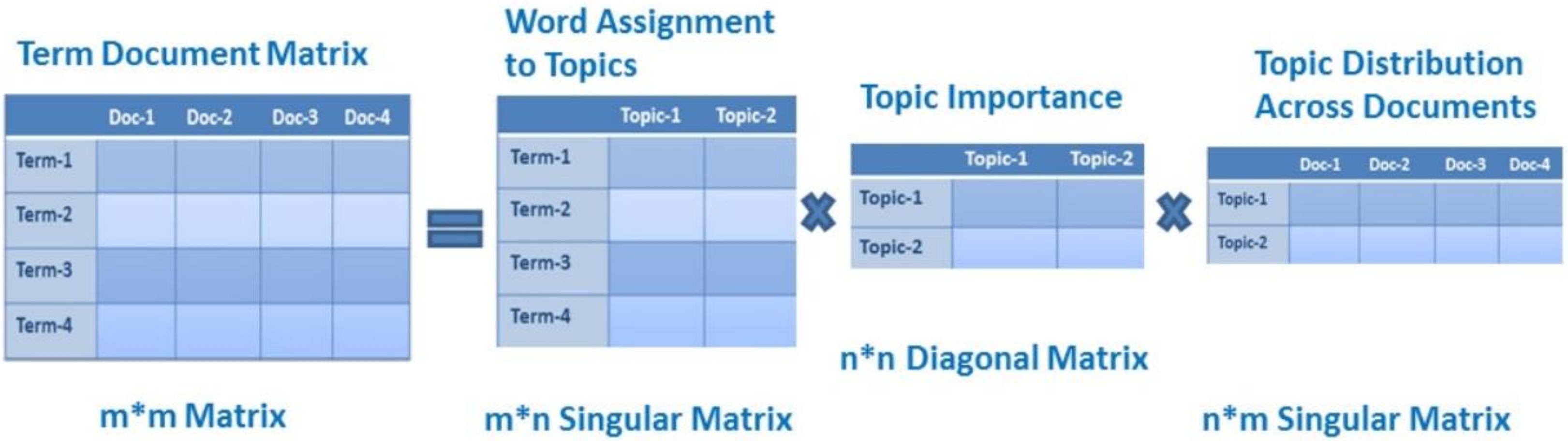
and since V is an orthogonal matrix ($\mathbf{V^{\top}V} = I$),

$XX^{\top} = U\mathbf{\Sigma^{2}}U^{\top}$ => eigenvalues ($\mathbf{D}$) = squared ($\mathbf{\Sigma^{2}}$) singular values of X.

- Hence, PCA can also be computed using SVD, where **eigenvectors** are **orthogonal** to each other (a special case of SVD, as the input matrix is symmetric – covariance measurement). Basically, SVD is the eigenvalue decomposition for general m×n matrices (rather than a strictly square n×n matrix – for PCA)

Latent Semantic Indexing (LSI)

- LSI also known as Latent Semantic Analysis (LSA) uses bag of word (other term features e.g., TF-IDF can also be used) model, which results in a term-document matrix (occurrence of terms in a document).
- LSI learns **latent topics** by performing a matrix decomposition on the document-term matrix using Singular Value Decomposition.



Term Document Matrix

	Doc-1	Doc-2	Doc-3	Doc-4
Term-1				
Term-2				
Term-3				
Term-4				

m*m Matrix

Word Assignment to Topics

	Topic-1	Topic-2
Term-1		
Term-2		
Term-3		
Term-4		

m*n Singular Matrix

Topic Importance

	Topic-1	Topic-2
Topic-1		
Topic-2		

n*n Diagonal Matrix

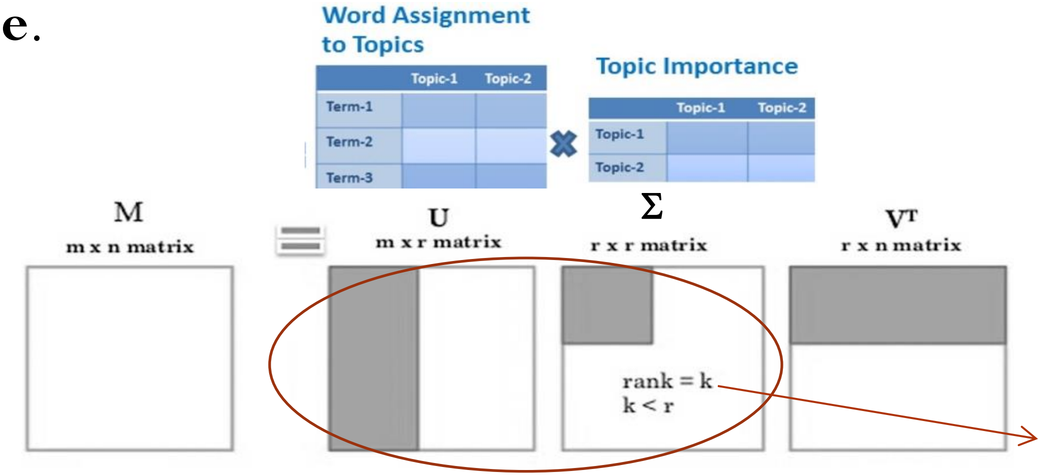
Topic Distribution Across Documents

	Doc-1	Doc-2	Doc-3	Doc-4
Topic-1				
Topic-2				

n*m Singular Matrix

Latent Semantic Indexing (LSI)

- LSI uses Singular Value Decomposition (SVD) with the main principle that **similar terms** tend to be used in the same context and hence tend to **co-occur more**.



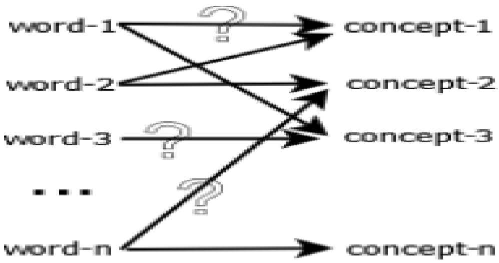
Word Assignment to Topics

	Topic-1	Topic-2
Term-1		
Term-2		
Term-3		

Topic Importance

	Topic-1	Topic-2
Topic-1		
Topic-2		

Specify k value (number of topics)

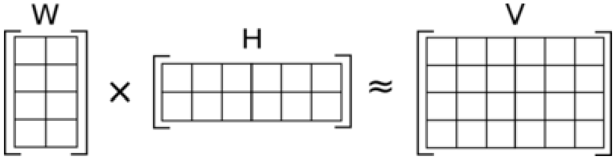


U is the ***term-topic*** matrix where each row of the matrix represents a term and each column represents a topic. It is used to get the **influential terms for each topic** when U is multiplied by the singular values in **Σ**.

- The term LSI comes from the fact that this technique has the ability to **uncover latent hidden terms** that correlate semantically with the **topics** => Sort influence terms for each topic to discover latent terms that identify the topics.
- LSI is based on the principle that SVD is used to perform dimensionality reduction to find terms with **similar co-occurrences**.

Non-negative Matrix Factorization

- Non-negative matrix factorization – a matrix decomposition technique that operates on **non-negative** matrices.
- Given a non-negative matrix *V*, the objective of NMF is to find two non-negative matrix factors, **W** and **H**, such that when they are multiplied, they can approximately reconstruct **V**, represented as follows:
  - $V \approx WH$, such that all three matrices are non-negative.
- To get to this approximation, **optimization** over an objective function using Euclidean distance or L2 norm between two matrices is performed



V (Document-word matrix) – **input** that contains which words appear in which documents.

W (Basis vectors) – the **topics** (clusters) discovered from the documents.

H (Coefficient matrix) - the **weights** for the topics in each document.

Non-negative Matrix Factorization

- NMF will produce two matrices W and H. The columns of W can be interpreted as basis documents (bags of words). What interpretation can we give to such a basis document in this case? They represent ***topics***! Sets of words found simultaneously in different documents.
- H tells us how to sum contributions from different topics to reconstruct the word mix of a given original document.

$$\underbrace{X(:,j)}_{j\text{th document}} \approx \sum_{k=1}^{r} \underbrace{W(:,k)}_{k\text{th topic}} \underbrace{H(k,j)}_{\substack{\text{importance of } k\text{th topic} \\ \text{in } j\text{th document}}}, \qquad \text{with } W \geq 0 \text{ and } H \geq 0.$$

- Therefore, given a set of documents, NMF **identifies topics** and simultaneously **classifies the documents among these different topics.**

Non-negative Matrix Factorization

- NMF decomposition of the term-document matrix would yield components that could be considered "**topics**", and decompose each document into a **weighted sum of topics**. This is called topic modeling and is an important application of NMF.

- The interpretation would not be possible with other decomposition methods - **cannot interpret what it means to have a "negative" weight of the "Movie" topic**. This is why the underlying components (topics) and their weights should be non-negative.

- Another interesting property of NMF is that it naturally produces **sparse** representations. This makes sense in the case of topic modeling: **documents generally do not contain a large number of topics.**

NMF vis-à-vis SVD

$$A = U \Sigma V^T$$

Singular Value Decomposition

- NMF gives only U and V matrices, but SVD gives a **Sigma matrix** also along with these two. Sigma gives insights into the amount of information each eigenvector holds, which is not available in NMF.

- SVD computes **eigenvectors**, which are deterministic for a given matrix. So they don't change much based on the package used or what are the initial conditions. Finding eigenvectors is key consideration in SVD (PCA use case).

- NMF on the other hand tries to get best fit decomposed matrices, which are **trained on some training data** and with evaluation done on test data. Based on technique used to get these smaller matrices you may end up with different results.

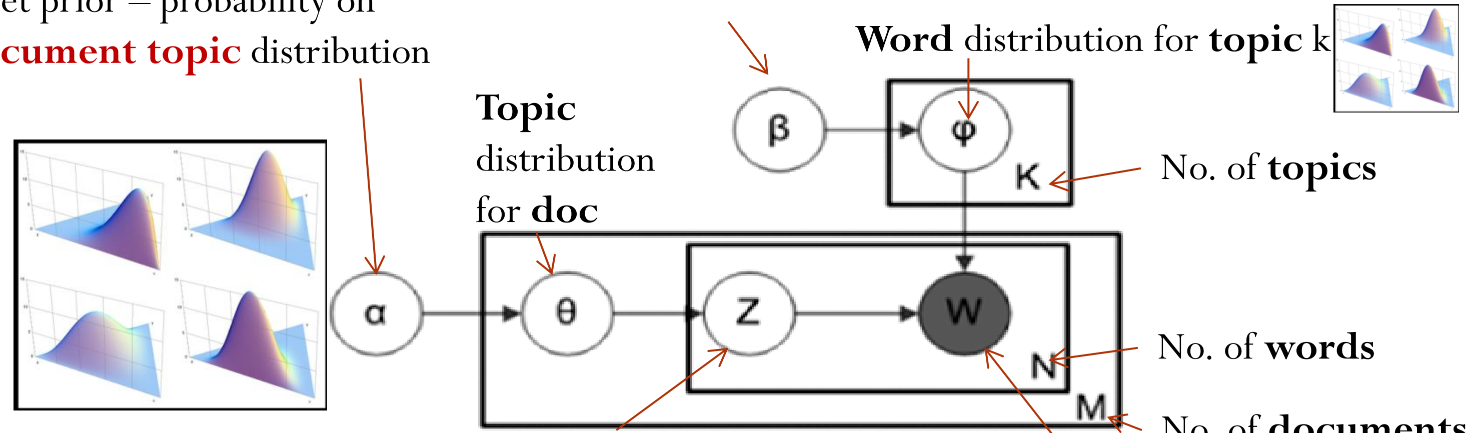
- Have to consider **regularization** in NMF to prevent overfitting while there's no such problem in SVD.

- NMF is **fast** and able to extract **sparse** and easily **interpretable** factors (topics / documents)

Latent Dirichlet Allocation (LDA)

- The Latent Dirichlet Allocation (LDA) technique is a ***generative probabilistic*** model in which each document is assumed to have a combination of topics. In this case, the latent topics contain a **Dirichlet** prior over them.

Dirichlet prior - encode the intuition that **documents** cover only a **small** set of **topics** and that **topics** use only a **small set** of **words** frequently



Probability on per **topic word** distribution

Dirichlet prior – probability on per **document topic** distribution

Word distribution for **topic** k

Topic distribution for **doc**

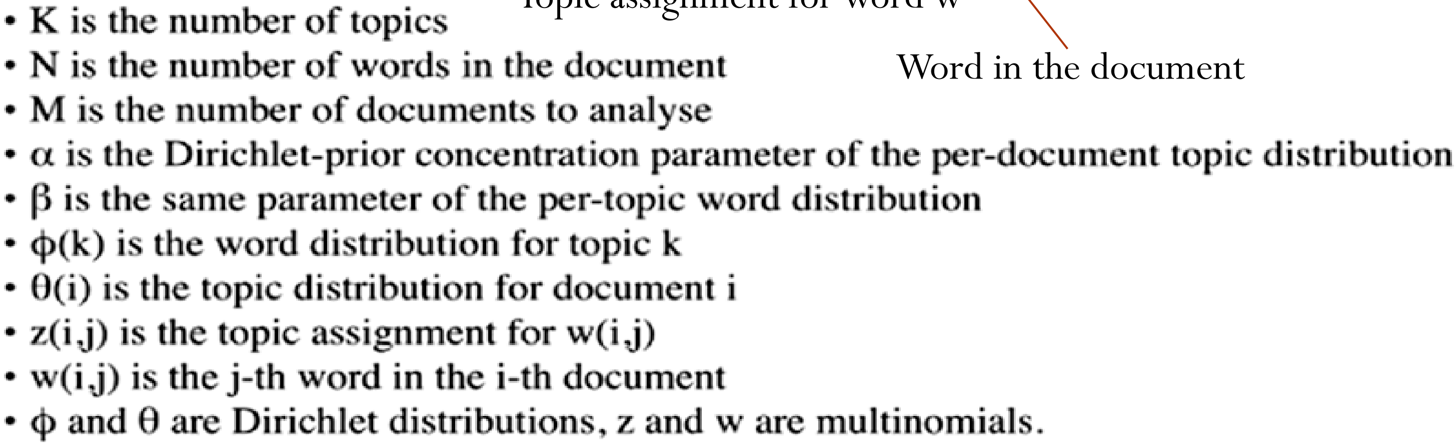
No. of **topics**

No. of **words**

No. of **documents**

Topic assignment for word w

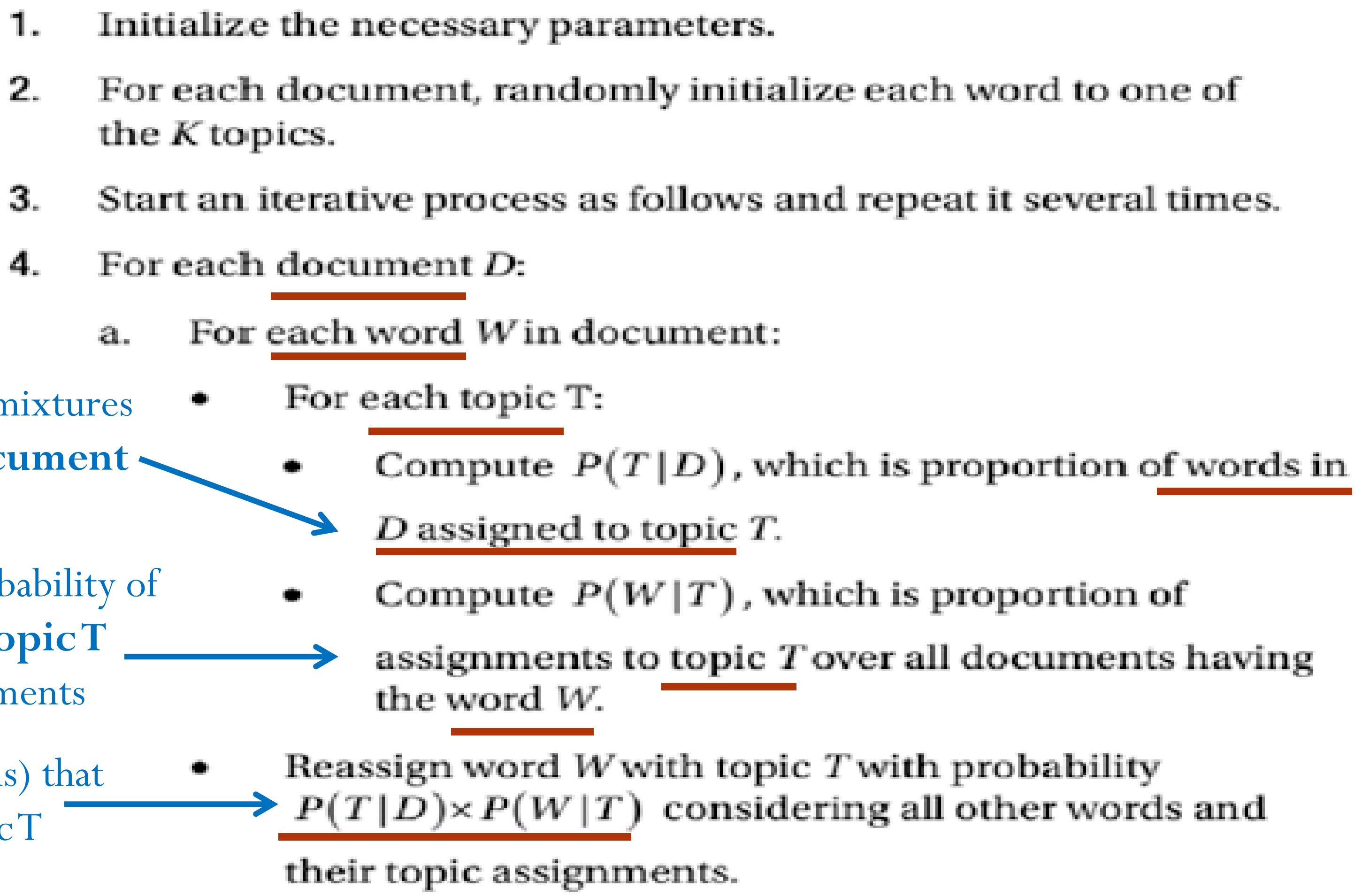
Word in the document



- K is the number of topics
- N is the number of words in the document
- M is the number of documents to analyse
- α is the Dirichlet-prior concentration parameter of the per-document topic distribution
- β is the same parameter of the per-topic word distribution
- ϕ(k) is the word distribution for topic k
- θ(i) is the topic distribution for document i
- z(i,j) is the topic assignment for w(i,j)
- w(i,j) is the j-th word in the i-th document
- ϕ and θ are Dirichlet distributions, z and w are multinomials.

LDA

- Steps in the LDA topic modeling algorithm



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.
4. For each document *D*:
   - a. For each word *W* in document:
     - 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.

Gives **topic** mixtures for **each document**

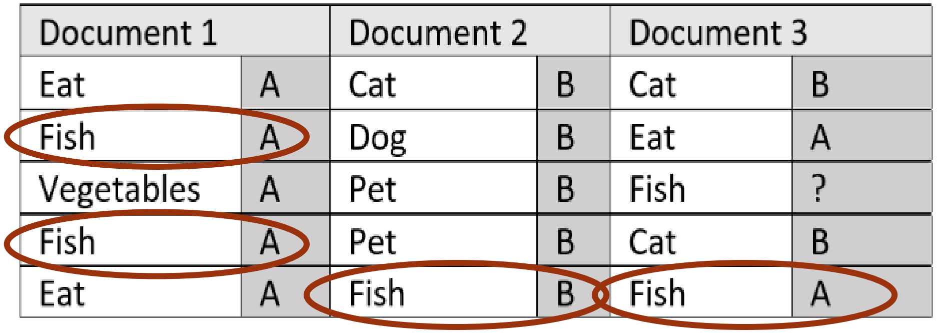
Gives the probability of **word W** in **topic T** over all documents

Words (terms) that point to topic T

The algorithm runs several iterations to give the **topic mixtures for each document**, which is then used to generate the constituents of each topic from the **terms that point to that topic**.

LDA - example

- Three documents with 2 topics: Food and Pets.



Document 1		Document 2		Document 3	
Eat	A	Cat	B	Cat	B
Fish	A	Dog	B	Eat	A
Vegetables	A	Pet	B	Fish	?
Fish	A	Pet	B	Cat	B
Eat	A	Fish	B	Fish	A

Example of topic distribution in documents

P(word | topics) and
P(topics | documents)

First calculate what the probability is of the word appearing in the different topics:

P('Fish' | topic A) = 0.75 3/4 = 0.75

P('Fish' | topic B) = 0.25 1/4 = 0.25

Next, calculate the probability of the topics in the document with the word in it, which is going to be: P(topic A | Document 3) = P(topic B | Document 3) = 0.5 because they are evenly split.

Weighing the conclusions from both probabilities, we will assign the word 'Fish' in Document 3 to topic A.

Repeat the step for all words in each document, and then repeat the entire classification multiple times. With **multiple iterations**, the algorithm will obtain better and better topic classifications each time with more updated information for each document.

Information Extraction

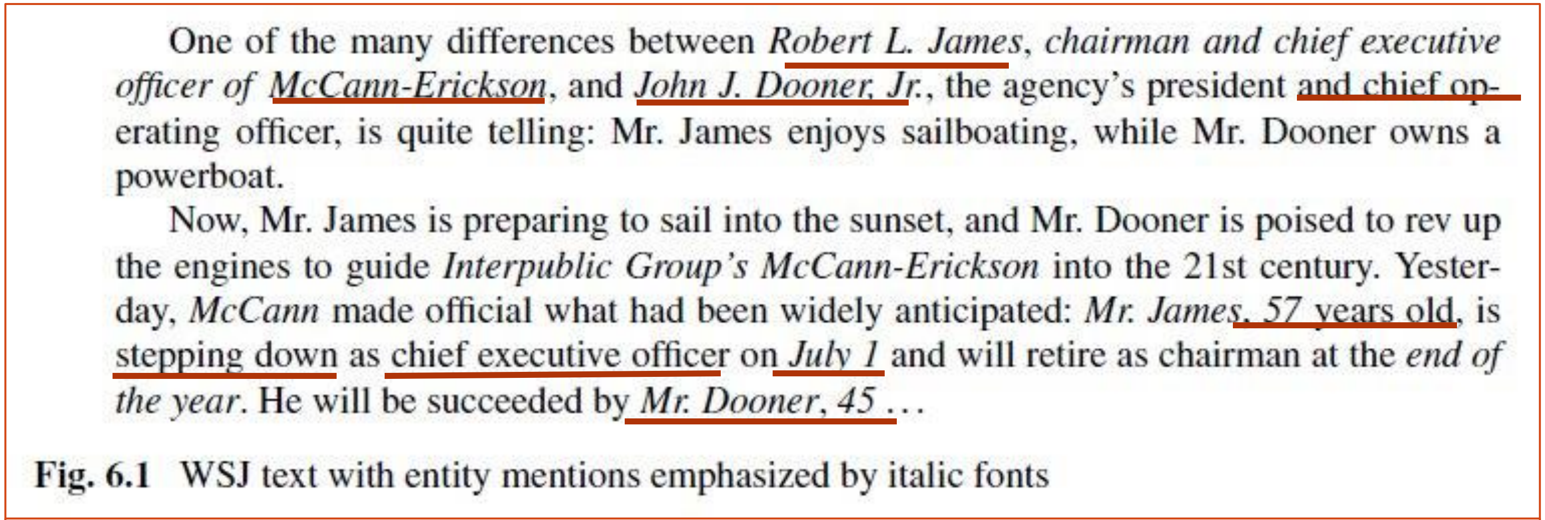
- An important research area for natural language processing and text mining is **the extraction and formatting of information from unstructured text**.

- Computers can be used to sift through a large amount of text and extract restricted forms of useful information.

- Information extraction can be regarded as a restricted form of full natural language understanding, where we know in advance what kind of **semantic information** we are looking for.

Goals of Information Extraction

- We illustrate the concept of information extraction using examples of identifying **executive position changes**.



One of the many differences between *Robert L. James*, *chairman and chief executive officer of McCann-Erickson*, and *John J. Dooner, Jr.*, the agency's president and chief operating officer, is quite telling: Mr. James enjoys sailboating, while Mr. Dooner owns a powerboat.

Now, Mr. James is preparing to sail into the sunset, and Mr. Dooner is poised to rev up the engines to guide *Interpublic Group's McCann-Erickson* into the 21st century. Yesterday, *McCann* made official what had been widely anticipated: *Mr. James, 57 years old*, is stepping down as chief executive officer on *July 1* and will retire as chairman at the *end of the year*. He will be succeeded by *Mr. Dooner, 45* . . .

Fig. 6.1 WSJ text with entity mentions emphasized by italic fonts

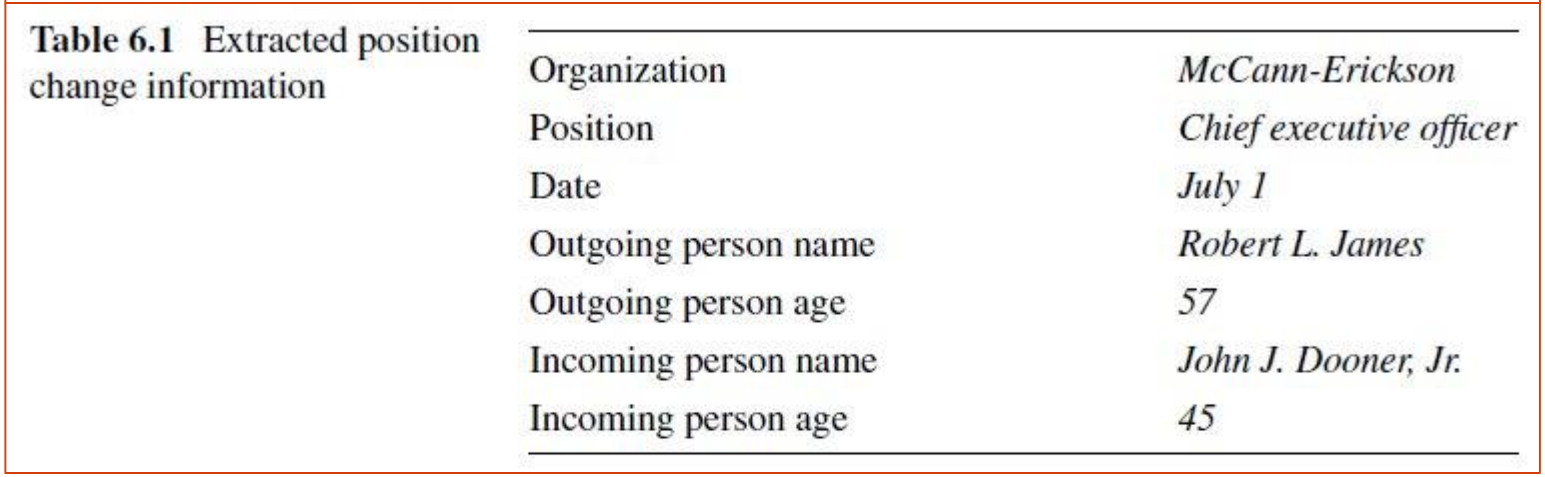


Table 6.1 Extracted position change information

Organization	*McCann-Erickson*
Position	*Chief executive officer*
Date	*July 1*
Outgoing person name	*Robert L. James*
Outgoing person age	*57*
Incoming person name	*John J. Dooner, Jr.*
Incoming person age	*45*

Goals of Information Extraction

- The most accurate information extraction systems often involve **handcrafted** language processing modules.
    - People's names have prefixes such as ***Mr., Ms., Miss., Dr., Jr., Sr. …***
    - People's names are recognized by phrases such as ***"according to …"*** or ***" … said"***.
- The application of machine-learning techniques to information extraction is motivated by **the time-consuming process needed to handcraft these systems**.

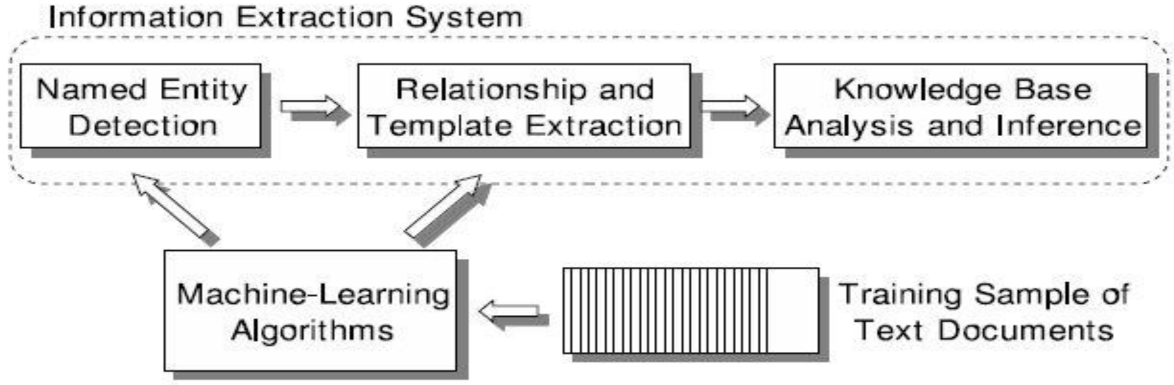

Information Extraction System
Named Entity Detection
Relationship and Template Extraction
Knowledge Base Analysis and Inference
Machine-Learning Algorithms
Training Sample of Text Documents

Fig. 6.3 Learnable information extraction system

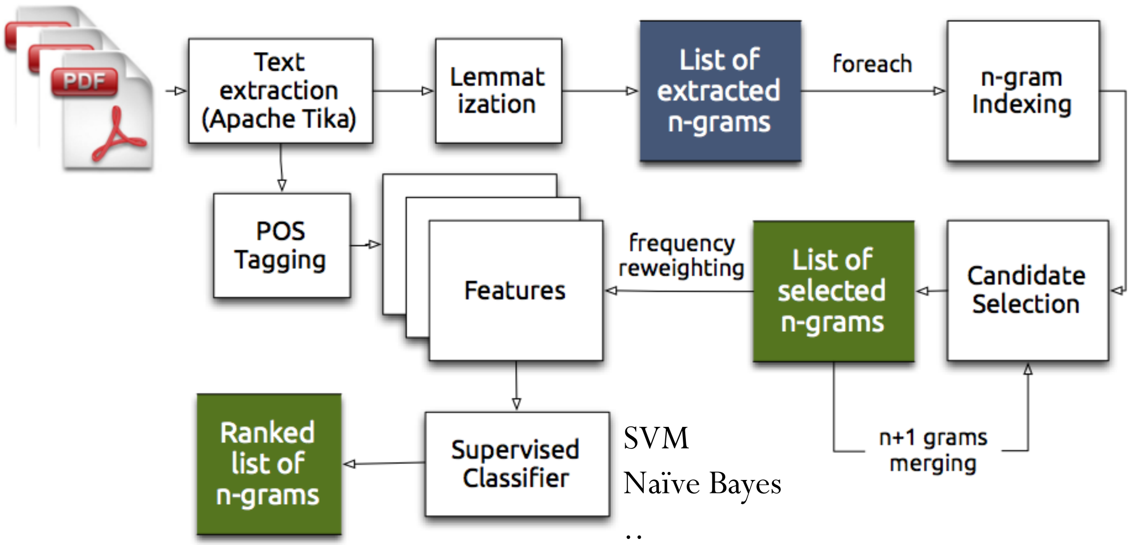
Goals of Information Extraction

- There are typically **two main modules** involved in an Information Extraction system.
- The purpose of the first module is to annotate the text document and find portions of the text that interest us (**Name Entity Recognition (NER)**).
  - For example, we want to identify the string ***Robert L. James*** as a **person** and the string ***McCann-Erickson*** as an **organization**.
- Once such entity mentions are extracted, another module is invoked to extract high-level information based on the entity mentions (**Relationship extraction**).
  - In the example, we want to identify that the person ***Robert L. James*** **belongs to** the organization ***McCann-Erickson***, and his age is **57**.

Named Entity Recognition (NER)

- Objective: **find** and **classify** entities in text: Person, Date, Organization
  - The decision by the independent MP Andrew Wilkie to withdraw his support for the minority Labor government sounded dramatic but it should not further threaten its stability. When, after the 2010 election, Wilkie, Rob Oakeshott, Tony Windsor and the Greens agreed to support Labor, they gave just two guarantees: confidence and supply.

Example of an NER system



https://exascale.info/named/entity/recognition/ner/Named-Entity-Recognition-for-Idiosyncratic-Web-Collections/

Named Entity Recognition (NER)

- The uses:
  - Named entities can be indexed, linked off, etc.
  - A lot of IE relations are associations between named entities
  - For question answering, answers are often named entities.
  - Sentiment can be attributed to companies or products
- Concretely:
  - Many web pages tag various entities, with links to bio or topic pages, etc.
    - Reuters' OpenCalais, AlchemyAPI (IBM Watson), Yahoo's Term Extraction, …
  - Microsoft: smart recognizers for document content
    - E.g., recognize a name, can take actions, such as add to contacts and open contacts.

Precision/Recall/F1 for IE/NER

- Recall and precision are straightforward for tasks like text categorization.
- The measure is a bit different for IE/NER when there are *boundary errors* (which are *common*):
  - First Bank of Chicago announced earnings …
- This counts as both a FP and a FN.
- Selecting *nothing* would have been better (just FN).
- Some other metrics (e.g., MUC scorer) give partial credit (according to complex rules)
  - MUC – Message Understanding Conference

Relationship Extraction

- The relation extraction module in a typical information extraction system often is constrained to **relationships among entities in a single sentence**.
- The most interesting relationships are often **binary relations**.
- For example, consider the sentence below, with the relationship that **the person** ***Robert L. James*** **belongs to organization** ***McCann-Erickson***.
- A binary relationship takes **two typed entities** as arguments.
- In the example above, we have **a belong-to relationship** that takes its first argument as a **person** and its second argument as an **organization**.

Hearst's Patterns for extracting IS-A relations

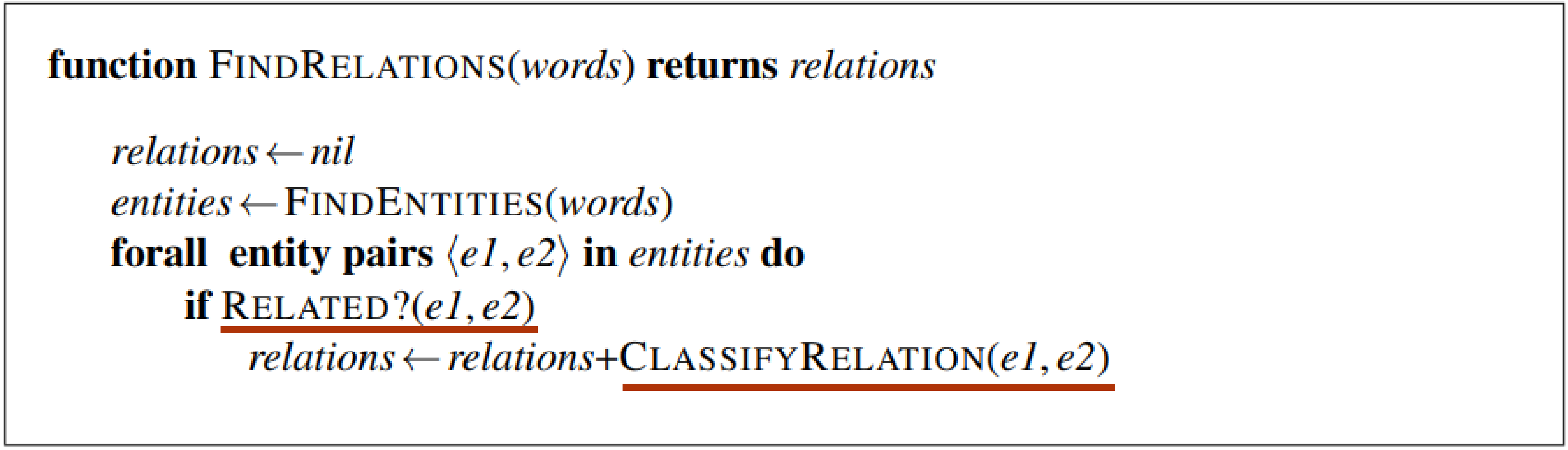
(Hearst, 1992): Automatic Acquisition of Hyponyms (i.e. subordinate categories)

Hearst pattern	Example occurrences
X and other Y	...temples, treasuries, and other important civic buildings.
X or other Y	Bruises, wounds, broken bones or other injuries...
Y such as X	The bow lute, such as the Bambara ndang...
Such Y as X	...such authors as Herrick, Goldsmith, and Shakespeare.
Y including X	...common-law countries, including Canada and England...
Y , especially X	European countries, especially France, England, and Spain...

- Rules based: relations often hold between specific entities.
  - located-in (ORGANIZATION, LOCATION)
  - founded (PERSON, ORGANIZATION)
  - cures (DRUG, DISEASE)
- Start with Named Entity tags to help extract relation!

Classifiers for supervised methods

- Possible to use any good classifier:
  - SVM
  - Naïve Bayes, …
- Train it on the training set, and test on the test set



```
function FINDRELATIONS(words) returns relations

    relations ← nil
    entities ← FINDENTITIES(words)
    forall entity pairs $\langle e1, e2\rangle$ in entities do
        if RELATED?(e1, e2)
            relations ← relations+CLASSIFYRELATION(e1, e2)
```

Figure 21.12 Finding and classifying the relations among entities in a text.

Related?(e1,e2): decide whether there is a relationship.
ClassifyRelation(e1,e2): classify them into one of multiple category relations, such as founder, employment, etc.

Parse Features for Relation Extraction

***American Airlines**, a unit of AMR, immediately **matched** the move, spokesman **Tim Wagner** said*

Mention 1 Mention 2

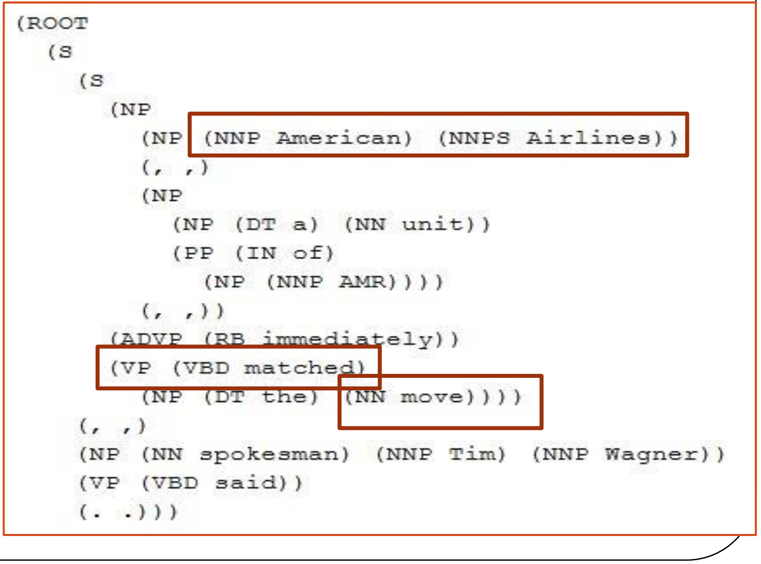
- Base syntactic chunk sequence from one to the other
- Constituent path through the tree from one to the other

  Airlines [**NNP**] matched [**VBD**] move [**NN**]

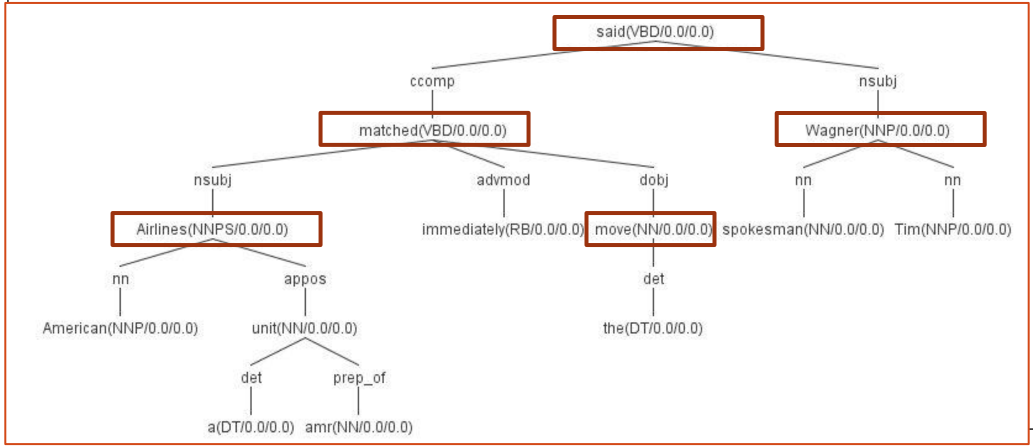
- Dependency path

  (Airlines matched move) [CCOMP] said [VBD] Wagner [NNP]

  Clausal complement



```
(ROOT
  (S
    (S
      (NP
        (NP (NNP American) (NNPS Airlines))
        (, ,)
        (NP
          (NP (DT a) (NN unit))
          (PP (IN of)
            (NP (NNP AMR))))
        (, ,))
      (ADVP (RB immediately))
      (VP (VBD matched)
        (NP (DT the) (NN move))))
    (, ,)
    (NP (NN spokesman) (NNP Tim) (NNP Wagner))
    (VP (VBD said))
    (. .)))
```



Evaluation of Supervised Relation Extraction

- Compute P/R/F_1 for each relation

$$\mathrm{P} = \frac{\text{\# of correctly extracted relations}}{\text{Total \# of extracted relations}}$$

$$R = \frac{\text{\# of correctly extracted relations}}{\text{Total \# of gold relations}} \qquad F_1 = \frac{2PR}{P+R}$$

+ Can get high accuracies with **enough hand-labeled training data**, if test similar enough to training

- Labeling a large training set is expensive

- Supervised models don't generalize well to different genres

Referenced Materials

- Fundamentals of Predictive Text Mining, Sholom M. Weiss, Nitin Indurkhya, and Tong Zhang, Springer.
  - Chapter 6
- NLP, Dan Jurafsky and Christopher Manning, http://www.stanford.edu/~jurafsky/NLPCourseraSlides.html
- Text Analytics with Python. A Practitioner's Guide to Natural Language Processing (2nd Edition). Dipanjan Sarkar