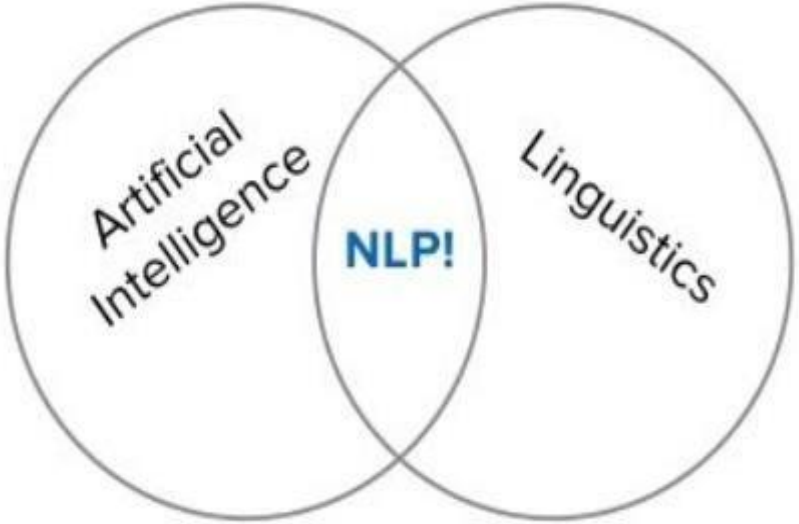
greatlearning
Learning for Life

Text Mining & Analytics

Natural language processing

Natural language processing (NLP) is an intersection between the fields of **computer science, linguistics** and **artificial intelligence. NLP** is concerned with the interactions between computers and human (natural) languages, in particular how to program computers to process, analyze and model large amounts of natural language data.



Artificial Languages: Java, C++, Binary...

Natural Language: Language spoken by *people*.

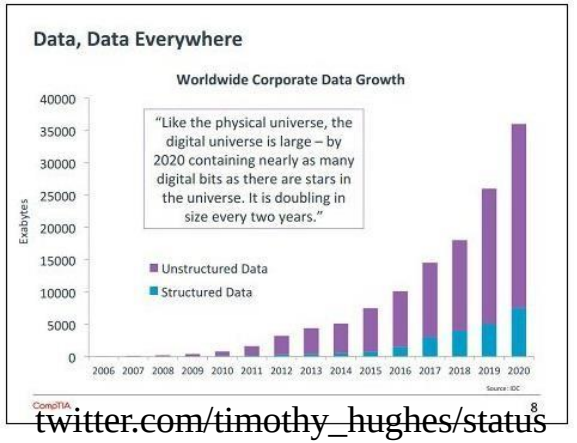
Motivation: Sophisticated linguistic analysis for human-like sophistication for a range of tasks or applications.

Goal: have computers *understand* natural language in order to perform useful tasks

What is Text Mining?

1. Text mining is the process of extracting useful insights / information from a body of text (classified as unstructured data)

2. Volume of unstructured data generated today is much more than the volume of structured data (90:10)



twitter.com/timothy_hughes/status/619075227021090817

3. Source of text data include -
   a. e-mails, corporate Web pages, customer surveys,
   b. Social media and more…

4. Lots of information is held-up in the text data

Introduction

1. Text data requires special preparation before we can start using it for predictive modelling
2. The text must be parsed to remove words, called tokenization.
3. Then the words need to be encoded as integers or floating point values for use as input to a machine learning algorithm, called feature extraction (or vectorization).
4. The scikit-learn library offers easy-to-use tools to perform both tokenization and feature extraction of text data
5. We now learn to prepare text data for predictive modeling in Python with scikit-learn
6. Learn to use the following algorithms
    a. CountVectorizer : Convert text to word count vectors
    b. TfidfVectorizer : Convert text to word frequency vectors
    c. HashingVectorizer : Convert text to unique integers

Applications of Text Mining

greatlearning
Learning for Life

1. Analyze open ended responses where respondents give their view or opinion without any constraints.

2. Automatic processing of huge volumes of electronic messages, emails

3. Classify the emails (text) as spam & non-spam.

4. Analyze warranty or insurance claims for suspicious / anomalous claims

5. Diagnosis of situations described by people in form of text for e.g. customer experience

6. Investigate competitors by crawling their web sites (be careful as crawling is illegal in many websites)

Applications of Text Mining: Legal Domain

1. Judgements summarization

2. Similar judgements identification

3. Legal case outcome prediction
    1. Domain specific and making it generic is a challenge
    2. **Can focus on consumer complaints judgements if interested**

4. Auto documentation

5. Automation of judicial processes (this is a broader problem)

Text Mining Techniques

greatlearning
Learning for Life

1. Information Extraction - This is used to analyze the unstructured text by identifying entities and their relationships.

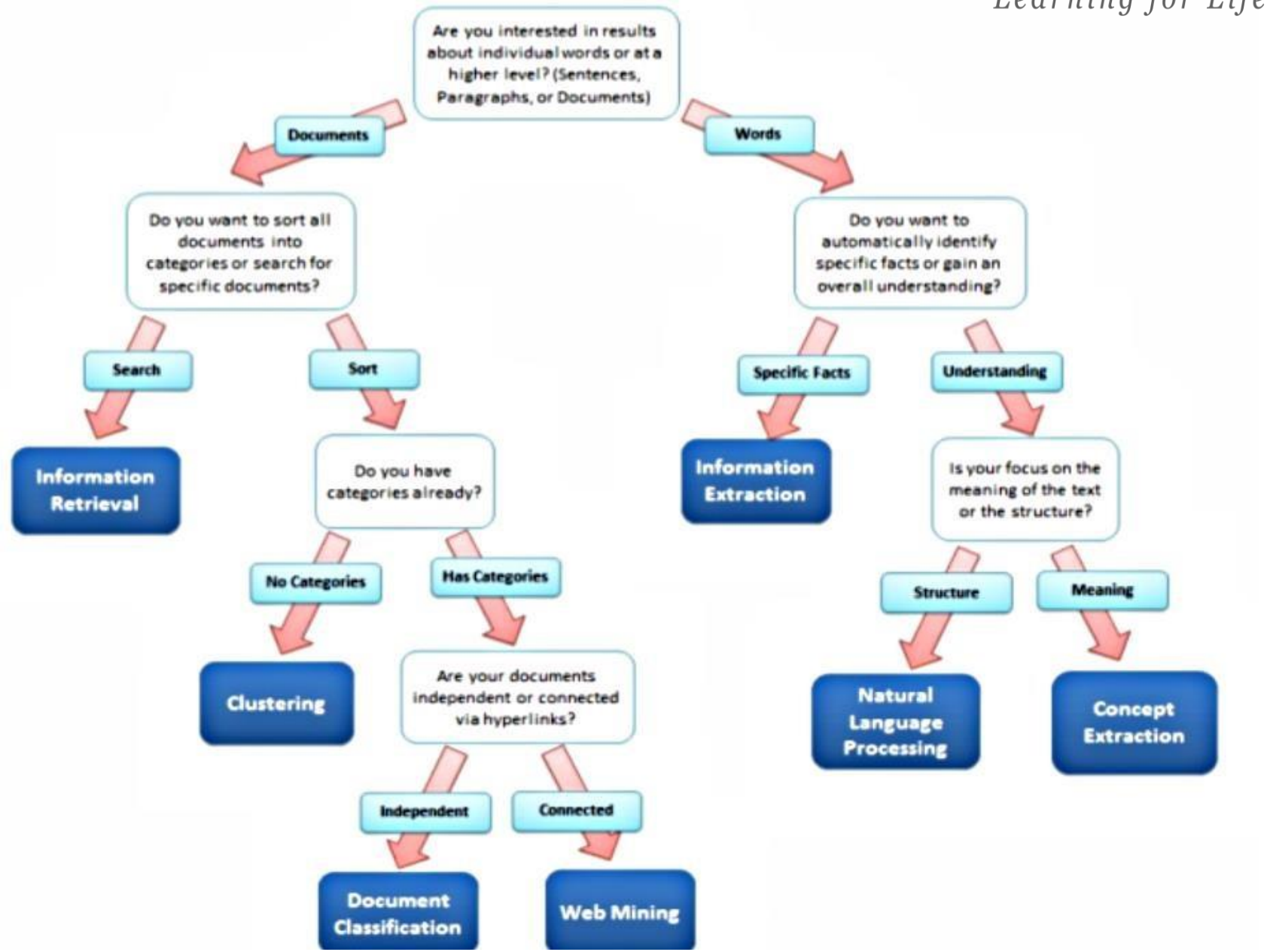
2. Categorization - Classifies the text document under one or more pre-determined categories such as spam or ham mails where each mail is a document

3. Clustering – Find similar document for instance similar queries in tech support databases for automated resolution

4. Summarization – Find the key parts of the document or what the document refers to and summarize the details

Formulating a Text Analytics Task

greatlearning
Learning for Life



Text Mining Challenges

1. Relations among word surface forms and their senses:
   a. Homonomy: same form, but different meaning (e.g. bank: river bank, financial institution)
   b. Polysemy: same form, related meaning (e.g. bank: blood bank, financial institution)
   c. Synonymy: different form, same meaning (e.g. singer, vocalist)
   d. Hyponymy: one word denotes a subclass of an another (e.g. breakfast, meal)

2. Word frequencies in texts have power distribution (Zipf's law):
   a. small number of very frequent words (usually useless words)
   b. big number of low frequency words (long tail of useful words)

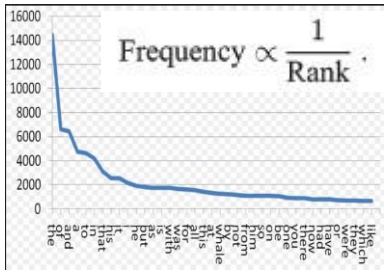


Image Source:
http://www.ruwhim.com/?p=4
7532

Stop words

1. Many of the most frequently used words in English are not useful in text analytics – these words are called stop words.
   - the, of, and, to, ….
   - Typically about 400 to 500 such words
   - For an application, an additional domain specific stop words list may be constructed

2. Why do we need to remove stop words
   - Reduce indexing (or data) file size
     - Stop words accounts 20-30% of total word counts.
   - Improve efficiency
     - Stop words are not useful for searching or text mining
     - Stop words always have a large number of hits

Bag Of Words model

greatlearning
Learning for Life

1. We cannot work with text directly when using machine learning algorithms, we need to convert the text to numbers.
2. We may want to perform classification of documents, so each document is an "input" and a class label is the "output" for our predictive algorithm.
3. Algorithms take vectors of numbers as input, therefore we need to convert documents to fixed-length vectors of numbers.
4. A simple and effective model for thinking about text documents in machine learning is called the Bag-of-Words Model, or BoW.
5. The model is simple in that it throws away all of the order information in the words and focuses on the occurrence of words in a document.

Bag Of Words model

6. This can be done by assigning each word a unique number. Then any document we see can be encoded as a fixed-length vector with the length of the vocabulary of known words. The value in each position in the vector could be filled with a count or frequency of each word in the encoded document.
7. This is the bag of words model, where we are only concerned with encoding schemes that represent what words are present or the degree to which they are present in encoded documents without any information about order.
8. The scikit-learn library provides 3 different APIs

Word Counts with CountVectorizer

6. CountVectorizer tokenizes a collection of text documents and builds a vocabulary of known words (textClassificationWithML.ipynb)
7. We can use it as follows:
   a. Create an instance of the CountVectorizer class.
   b. Call the fit() function in order to learn a vocabulary from one or more documents.
   c. Call the transform() function on one or more documents as needed to encode each document as a vector.
   d. An encoded vector is returned with a length of the entire vocabulary and an integer count for the number of times each word appeared in the document.
   e. The vectors returned from a call to transform() will be sparse vectors, and we can transform them back to numpy arrays to look and better understand what is going on by calling the toarray() function.

Document Term Matrix

- Most common form of representation in text mining is the *term - document* matrix
  - Term: typically a single word, but could be a word phrase like "data mining"
  - Document: a generic term meaning a collection of text to be retrieved
  - Can be large - terms are often 50k or larger, documents can be in the billions!
  - Can be binary, or use counts (frequency count)

textClassificationWithML.ipynb

Document Term Matrix

Example: 10 documents: 6 terms

	ML	Spark	Kafka	BigData	NoSql	SVM
D1	24	21	9	0	0	3
D2	32	10	5	0	3	0
D3	12	16	5	0	0	0
D4	6	7	2	0	0	0
D5	43	31	20	0	3	0
D6	2	0	0	18	7	6
D7	0	0	1	32	12	0
D8	3	0	0	22	4	4
D9	1	0	0	34	27	25
D10	6	0	0	17	4	23

$$D_1 = (d_{i1}, d_{i2}, \ldots, d_{it})$$

- Each document now is just a vector of terms, sometimes boolean

Document Term Matrix

greatlearning
Learning for Life

1. Semantic content of the text is ignored
2. Before creating DTM, all same terms should look same
3. Remove words with no information (Stop Words)
4. Express the words in their root form (Stem)

Feature Selection

- Performance of text classification algorithms can be optimized by selecting only a subset of the discriminative terms
  - Even after stopword removal.

- Greedy search
  - Start from full set and delete one at a time
  - Find the least important variable
    - Can use Gini index for this if a classification problem

- Often performance does not degrade even with orders of magnitude reductions
  - Only 140 out of 20,000 terms needed for classification!

Distances in DT matrices

- Given a doc term matrix representation, now we can define distances between documents (or terms!)
- Elements of matrix can be 0,1 or term frequencies (sometimes normalized)
- Can use Euclidean or cosine distance
- Cosine distance is the angle between the two vectors
- Not intuitive, but has been proven to work well

$$d_c(D_i, D_j) = \frac{\sum_{k=1}^{T} d_{ik} d_{jk}}{\sqrt{\sum_{k=1}^{T} d_{ik}^2 \sum_{k=1}^{T} d_{jk}^2}}$$

- If docs are the same, $d_c = 1$, if nothing in common $d_c = 0$

Bag Of Words / Boolean model

1. Documents (including query document) are treated as a collection / bag of words

2. Word sequence / semantics is not considered

3. Given a collection of documents D, let vocabulary V = {t1, t2, ..., t|V|} is set of distinctive words in D

4. A weight wij > 0 is associated with each term ti of a document dj ∈ D. For a term that does not appear in document dj, wij = 0

$$\mathbf{d}_j = (w_{1j}, w_{2j}, ..., w_{|V|j}),$$

BAG Of Words / Boolean Model

Document 1

The quick brown fox jumped over the lazy dog's back.

Document 2

Now is the time for all good men to come to the aid of their party.

Term	Document 1	Document 2
aid	0	1
all	0	1
back	1	0
brown	1	0
come	0	1
dog	1	0
fox	1	0
good	0	1
jump	1	0
lazy	1	0
men	0	1
now	0	1
over	1	0
party	0	1
quick	1	0
their	0	1
time	0	1

Stopword List

for
is
of
the
to

Source: http://lintool.github.io/UMD-courses/LBSC796-INFM718R-2006-Spring/syllabus.html

Bag Of Words / Boolean model

Term	Doc 1	Doc 2	Doc 3	Doc 4	Doc 5	Doc 6	Doc 7	Doc 8
aid	0	0	0	1	0	0	0	1
all	0	1	0	1	0	1	0	0
back	1	0	1	0	0	0	1	0
brown	1	0	1	0	1	0	1	0
come	0	1	0	1	0	1	0	1
dog	0	0	1	0	1	0	0	0
fox	0	0	1	0	1	0	1	0
good	0	1	0	1	0	1	0	1
jump	0	0	1	0	0	0	0	0
lazy	1	0	1	0	1	0	1	0
men	0	1	0	1	0	0	0	1
now	0	1	0	0	0	1	0	1
over	1	0	1	0	1	0	1	1
party	0	0	0	0	0	1	0	1
quick	1	0	1	0	0	0	0	0
their	1	0	0	0	1	0	1	0
time	0	1	0	1	0	1	0	0

Each column represents the view of a particular document: What terms are contained in this document?

Each row represents the view of a particular term: What documents contain this term?

To execute a query, pick out rows corresponding to query terms and then apply logic table of corresponding Boolean operator

Source: http://lintool.github.io/UMD-courses/LBSC796-INFM718R-2006-Spring/syllabus.html

BAG Of Words / Boolean model (contd)

1. Query terms are combined logically using the Boolean operators **AND**, **OR**, and **NOT.**
   a. E.g., ((*data* AND *mining*) AND (NOT *text*))

2. Retrieval
   a. Given a Boolean query, the system retrieves every document that makes the query logically true.
   b. Called **exact match**.

1. The retrieval results are usually quite poor because term frequency is not considered

BAG Of Words / Boolean model (contd)

Term	Doc 1	Doc 2	Doc 3	Doc 4	Doc 5	Doc 6	Doc 7	Doc 8
dog	0	0	1	0	1	0	0	0
fox	0	0	1	0	1	0	1	0
dog ∧ fox	0	0	1	0	1	0	0	0

dog AND fox → Doc 3, Doc 5

dog ∨ fox	0	0	1	0	1	0	1	0

dog OR fox → Doc 3, Doc 5, Doc 7

dog ¬ fox	0	0	0	0	0	0	0	0

dog NOT fox → empty

fox ¬ dog	0	0	0	0	0	0	1	0

fox NOT dog → Doc 7

Term	Doc 1	Doc 2	Doc 3	Doc 4	Doc 5	Doc 6	Doc 7	Doc 8
good	0	1	0	1	0	1	0	1
party	0	0	0	0	0	1	0	1
g ∧ p	0	0	0	0	0	1	0	1
over	1	0	1	0	1	0	1	1
g ∧ p ¬ o	0	0	0	0	0	1	0	0

good AND party → Doc 6, Doc 8

good AND party NOT over → Doc 6

Source: http://lintool.github.io/UMD-courses/LBSC796-INFM718R-2006-Spring/syllabus.html

BAG Of Words / Vector Space model

1. Each document is represented as a vector. The term weights are no longer 0 or 1.
2. Each term weight is computed based on term frequency. But term frequency can mislead as they may occur frequently across all documents in all classes (poor classifiers)
3. In TFIDF, using this concept weightage to frequent terms across all documents is reduced relative to other words

"The quick brown fox jumped over the lazy dog's back"
→ [1 1 1 1 1 1 1 1 2]

document

Vector in feature space

1st position corresponds to "back"
2nd position corresponds to "brown"
3rd position corresponds to "dog"
4th position corresponds to "fox"
5th position corresponds to "jump"
6th position corresponds to "lazy"
7th position corresponds to "over"
8th position corresponds to "quick"
9th position corresponds to "the"

Image Source: http://lintool.github.io/UMD-courses/LBSC796-INFM718R-2006-Spring/syllabus.html

BAG Of Words / Vector Space model

- Represent a doc by a term vector
  - Term: basic concept, e.g., word or phrase
  - Each term defines one dimension
  - N terms define a N-dimensional space
  - Element of vector corresponds to term weight
  - E.g., d = $(x_1,\ldots,x_N)$, x_i is "importance" of term i
- New document is assigned to the most likely category based on vector similarity.

BAG Of Words / Vector Space model

1. How to select terms to represent the documents as vectors
   - Remove fluff words (Stopwords)
     - e.g. “and”, “the”, “always”, “along”
   - Use word stem to prevent same word becoming multi dimension
     - e.g. “training”, “trainer”, “trained” => “train”
   - Latent semantic indexing
2. How to assign weights to terms
   - Not all words are equally important: Some are more indicative than others
     - e.g. “Automobile” vs. “Car”
3. How to measure the similarity between document vectors

BAG Of Words / Vector Space model

1. Given two document

$$D_i = (w_{i1}, w_{i2}, \cdots, w_{iN}) \qquad D_j = (w_{j1}, w_{j2}, \cdots, w_{jN})$$

2. Similarity definition
   - dot product

$$Sim(D_i, D_j) = \sum_{t=i}^{N} w_{it} * w_{jt}$$

   - normalized dot product (or cosine)

$$Sim(D_i, D_j) = \frac{\sum_{t=i}^{N} w_{it} * w_{jt}}{\sqrt{\sum_{t=1}^{N} (w_{it})^2 * \sum_{t=1}^{N} (w_{jt})^2}}$$

Lab: SMS classification

textClassificationWithML.ipynb

Word Frequencies with TFIDFVectorizer

1. One issue with simple counts is that some words other than stopwords appear many times
2. Doc vectors with large counts attributes will not be very meaningful in the encoded vectors. It is like a dimension with higher scale overwhelming others
3. An alternative is to calculate word frequencies and standardizing which is done using TF-IDF.
4. This is an acronym than stands for "Term Frequency – Inverse Document" Frequency which are the components of the resulting scores assigned to each word.
5. Term Frequency: This summarizes how often a given word appears within a document.
6. Inverse Document Frequency: This downscales words that appear a lot across documents.
7. Without going into the math, TF-IDF are word frequency scores that try to highlight words that are more interesting, e.g. frequent in a document but not across documents.

BAG Of Words / Vector Space model

1. **TF:** Term Frequency, which measures how frequently a term occurs in a document.
   a. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones.
   b. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:
   c. TF(t) = (Number of times term t appears in a document) / (Total number of terms
   in the document)

2. **IDF:** Inverse Document Frequency measures how important a term is.
   a. Certain terms, may appear a lot of times across documents but have little importance.
   b. Weigh down the frequent terms while scale up the rare ones, by computing the following:
      a. IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

Source: http://www.tfidf.com/

BAG Of Words / Vector Space model

$$w_{i,j} = \mathrm{tf}_{i,j} \cdot \log \frac{N}{n_i}$$

$w_{i,j}$ **weight assigned to term *i* in document *j***

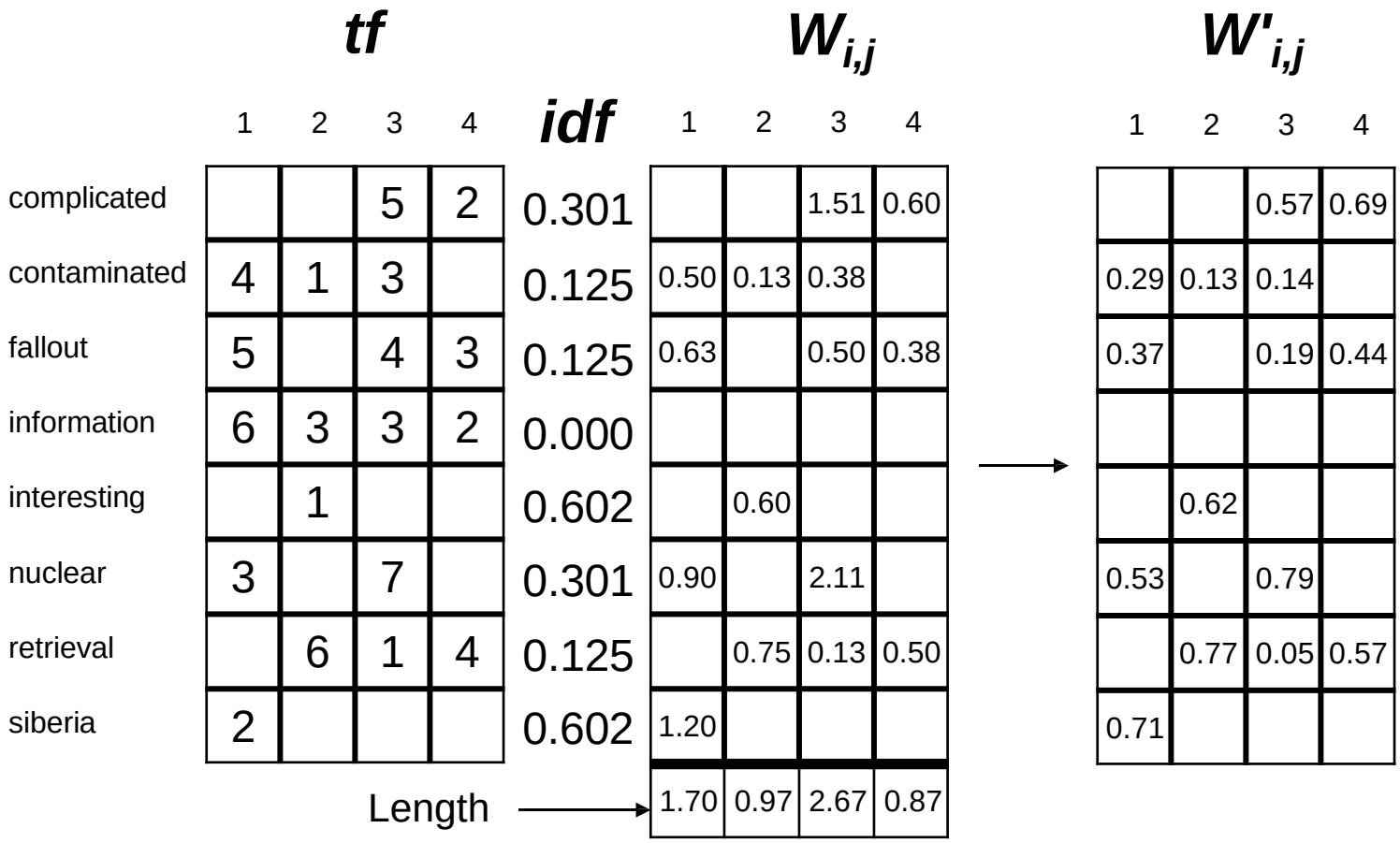
$\mathrm{tf}_{i,j}$ **number of occurrence of term *i* in document *j***

N **number of documents in entire collection**

n_i **number of documents with term *i***

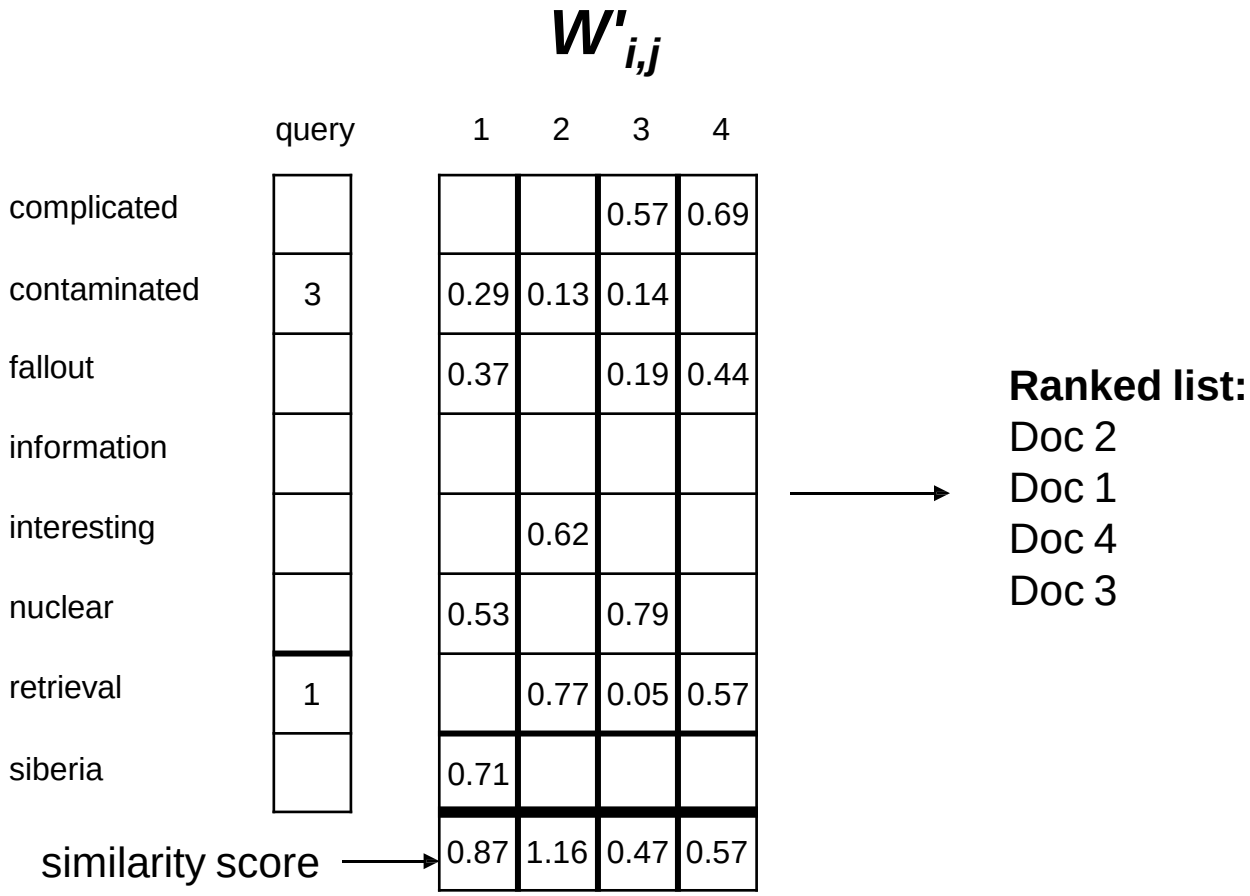
- Consider a document containing 1000 words wherein the word ML appears 30 times. The term frequency (i.e., tf) for ML is then (30 / 1000) = 0.03. Now, assume we have 10 million documents and the word ML appears in one thousand of these. Then, the inverse document frequency (i.e., idf) is calculated as log(10,000,000 / 1,000) = 4. Thus, the Tf-idf weight is the product of these quantities: 0.03 * 4 = 0.12.
- This is the coefficient of the given document

BAG Of Words / Vector Space model



Source: http://lintool.github.io/UMD-courses/LBSC796-INFM718R-2006-Spring/syllabus.html

BAG Of Words / Vector Space model



Source: http://lintool.github.io/UMD-courses/LBSC796-INFM718R-2006-Spring/syllabus.html

Word Frequencies with TFIDFVectorizer

1. The TfidfVectorizer will tokenize documents, learn the vocabulary and inverse document frequency weightings, and allow you to encode new documents.
2. Alternately, if we already have a learned CountVectorizer, we can use it with a TfidfTransformer to just calculate the inverse document frequencies and start encoding documents.

Questions?