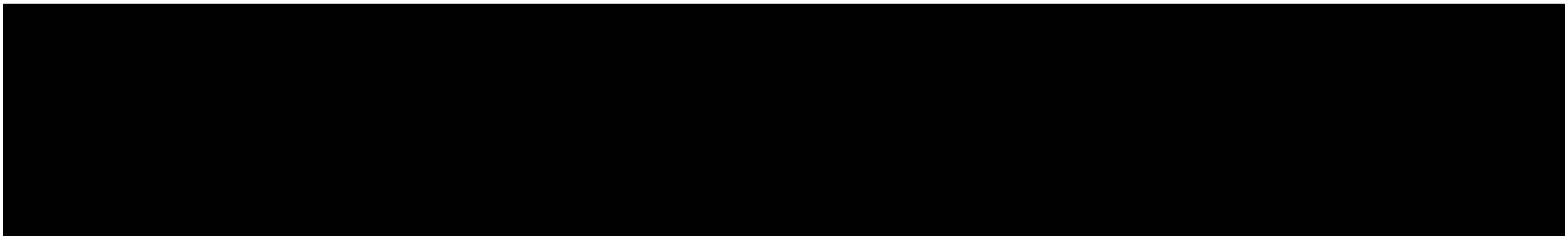
Machine Learning

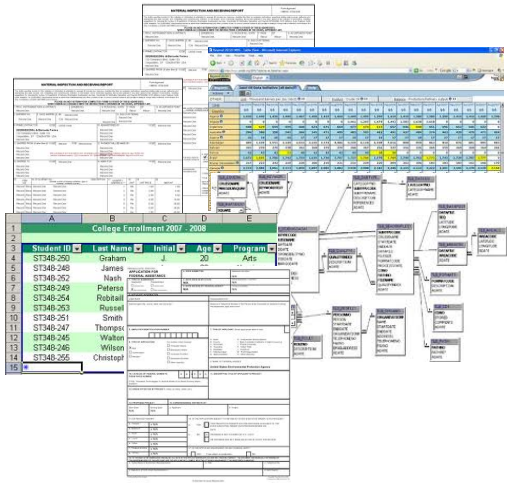
Working with Text

Sarah Jane Delany



Motivation

- Extracting insight from text data….



Identifying what the text is about

Grouping text documents into topics

Identifying the opinion expressed in text

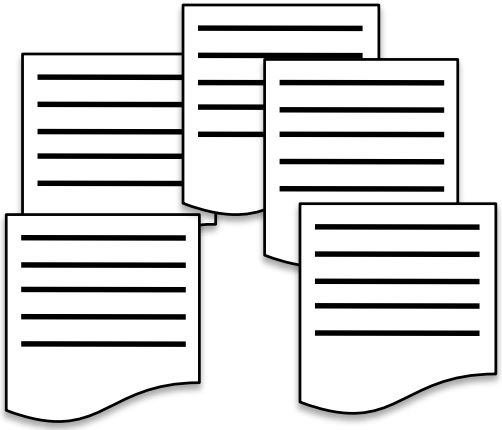
….

What's different about text?

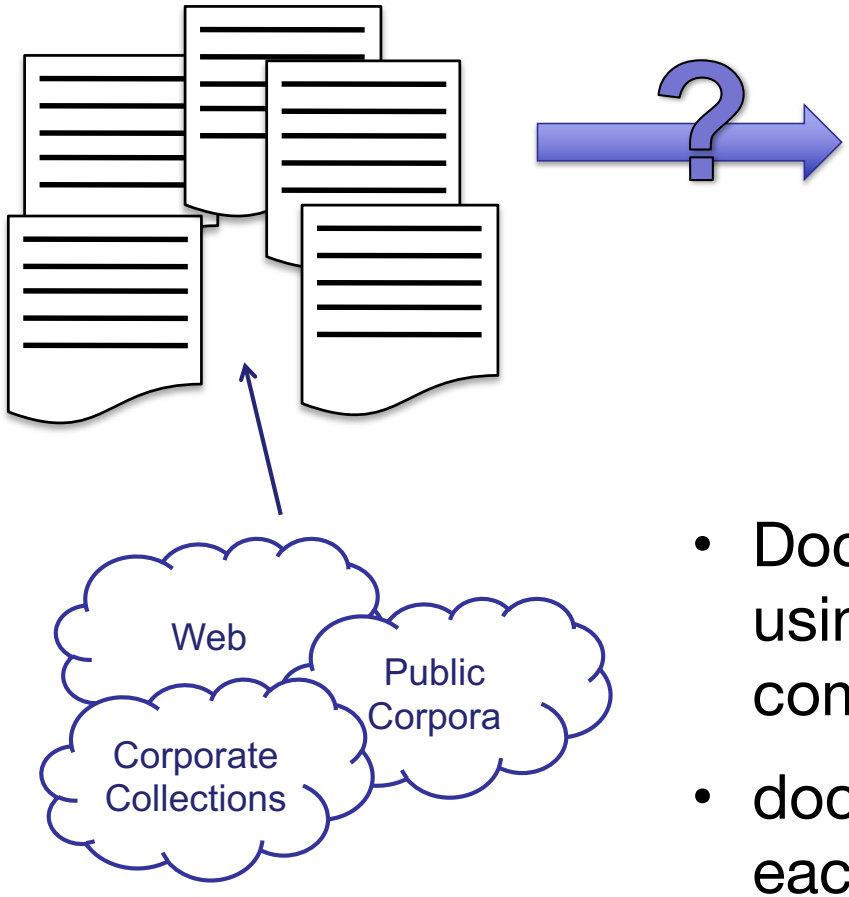
- Structured

NAME	SALARY:LOAN	HOME	DEFAULT
Mike	0.25	N	yes
Bill	0.2	Y	no
Betty	0.33	Y	no
Bob	0.6	N	no
Dave	0.11	N	yes
Anne	0.33	N	yes

- Unstructured



Representing Text



- Documents are typically represented using the vector-space model commonly known as bag-of-words
- doc = vector in n-dim space where each dim represents a word

Example

Patient Record 1

The patient is a male with a history of diabetes, hypertension, and CAD.

Patient Record 2

The patient presented with no signs of CAD.

term-document matrix

	the	patient	is	a	male	with	history	of	diabetes	hypertension	and	CAD	presented	no	signs
PR1	1	1	1	2	1	1	1	1	1	1	1	1			
PR2	1	1				1		1				1	1	1	1

term weight

Representation

Term weight can use:

- Boolean / Binary Representation

$$tf_{ij} = [0|1] \qquad tf_{ij} = \begin{cases} 1 & \text{if } \text{term}_j \in \text{doc}_i \\ 0 & \text{otherwise} \end{cases}$$

- Term / Word frequency
  - Raw Frequencies $tf_{ij} = f_{ij}$

Representation

Term weight can use:

- Boolean / Binary Representation

$$tf_{ij} = [0|1] \qquad tf_{ij} = \begin{cases} 1 & \text{if } \text{term}_j \in \text{doc}_i \\ 0 & \text{otherwise} \end{cases}$$

- Term / Word frequency
  - Raw Frequencies $tf_{ij} = f_{ij}$

Consider a word occurring 5 times in a 10 word document versus a word occurring 5 times in a 1000 word document…

Representation

Term weight can use:

- Boolean / Binary Representation

$$tf_{ij} = [0|1] \qquad tf_{ij} = \begin{cases} 1 & \text{if } \text{term}_j \in \text{doc}_i \\ 0 & \text{otherwise} \end{cases}$$

- Term / Word frequency
  - Raw Frequencies $tf_{ij} = f_{ij}$
  - Normalised Term Frequency

$$tf_{ij} = \frac{f_{ij}}{\max_k(f_{kj})}$$

Representation

Term weight can use:

- Boolean / Binary Representation

$$tf_{ij} = [0|1] \qquad tf_{ij} = \begin{cases} 1 & \text{if term}_j \in \text{doc}_i \\ 0 & \text{otherwise} \end{cases}$$

- Term / Word frequency
  - Raw Frequencies $tf_{ij} = f_{ij}$
  - Normalised Term Frequency

$$tf_{ij} = \frac{f_{ij}}{\max_k(f_{kj})}$$

Consider words that occur in every document in the corpus…

Consider words that occur in very few documents in the corpus…

Inverse Document Frequency

- Inverse Document Frequency (idf) is a measure across a corpus of documents D of how common or rare a particular term is…

$$\text{idf}_i = log \frac{|D|}{|d_j \in D : term_i \in d_j|}$$

Collection of 1000 documents

» term in 1000 docs - log(1000/1000) = 0

» term in 500 docs - log(1000/500) = .301

» term in 20 docs - log(1000/20) = 2.698

» term in 1 doc - log(1000/1) = 4

- Taking the log just scales the values
- Higher values for infrequent terms

Representation

Term weight can use:

- Boolean / Binary Representation

$$tf_{ij} = [0|1] \qquad tf_{ij} = \begin{cases} 1 & \text{if } \text{term}_j \in \text{doc}_i \\ 0 & \text{otherwise} \end{cases}$$

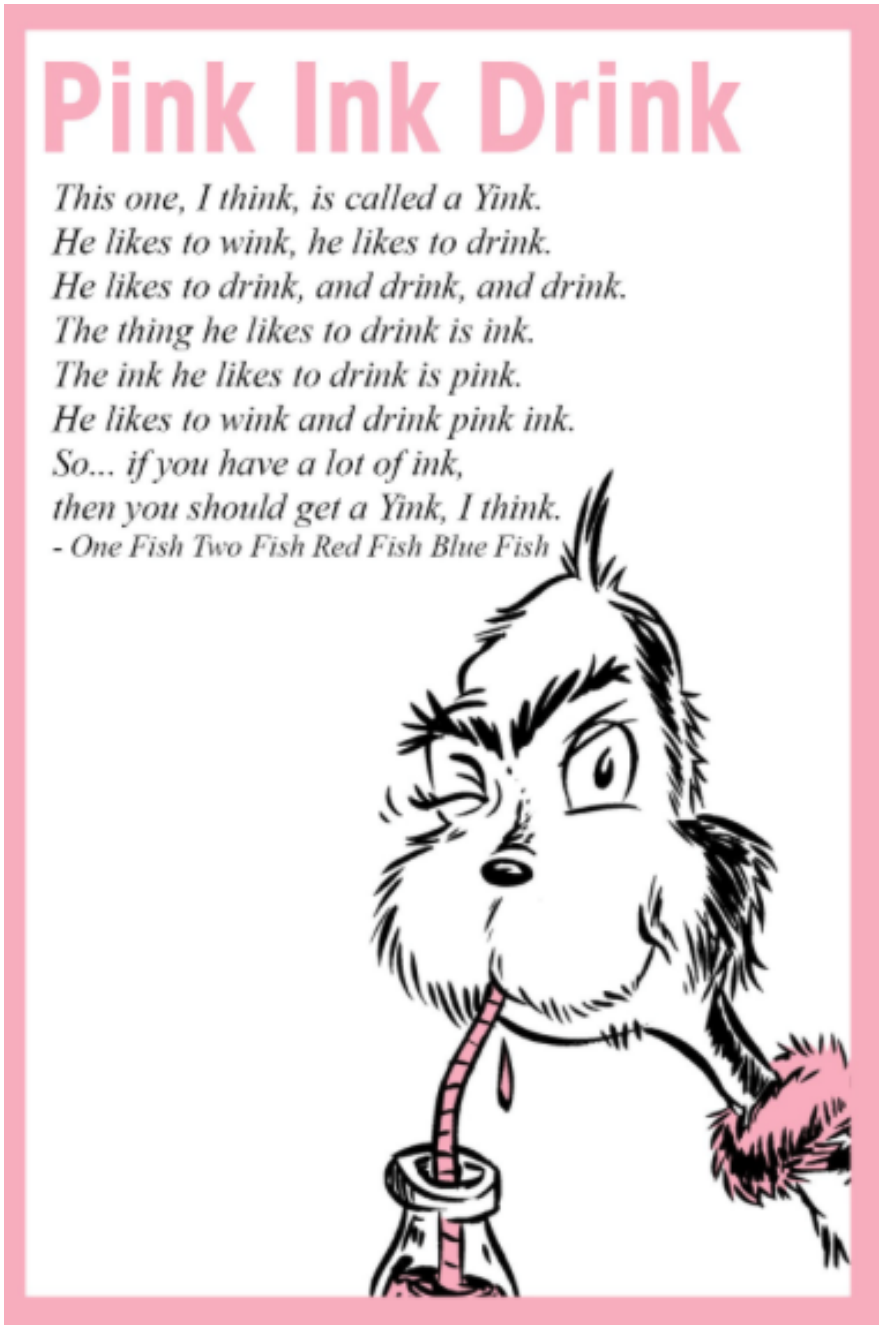
- Term / Word frequency
  - Raw Frequencies $tf_{ij} = f_{ij}$
  - Normalised Term Frequency

$$tf_{ij} = \frac{f_{ij}}{\max_k(f_{kj})}$$

  - Tf-idf

$$tf_{ij} = tf_{ij} \times \text{idf}_i$$

Represent this…



Doc 1
This one, I think, is called a Yink
He likes to wink, he likes to drink

Doc 2
He likes to drink, and drink, and drink
The thing he likes to drink is ink

Doc 3
The ink he likes to drink is pink
He likes to wink and drink pink ink

Doc 4
So… if you have a lot of ink
then you should get a Yink, I think.

Bag of Words

- BOW works well for a large range of text applications

No, the patient has cancer

vs.

The patient has no cancer

Bag of Words

- BOW works well for a large range of text applications

No, the patient has cancer

vs.

The patient has no cancer

Bag of Words

- BOW works well for a large range of text applications

No, the patient has cancer

vs.

The patient has no cancer

- But text is language and can be …

…multi-lingual

…semi-structured

…sequential

Text as Characters

Thx for ur msg, c u l8r

Text as Characters

bag of words

Thx for ur msg, c u l8r

Text as Characters

word trigrams…

word bigrams Thx_for for_ur ur_msg msg_c c_u u_l8r

bag of words Thx for ur msg c u l8r

Thx for ur msg, c u l8r

Text as Characters

word trigrams…

word bigrams Thx_for for_ur ur_msg msg_c c_u u_l8r

bag of words Thx for ur msg c u l8r

Thx for ur msg, c u l8r

bigrams Th hx x_ _f or r_ _m ms sg g, ,_ … 8r

Text as Characters

word trigrams…

word bigrams Thx_for for_ur ur_msg msg_c c_u u_l8r

bag of words Thx for ur msg c u l8r

Thx for ur msg, c u l8r

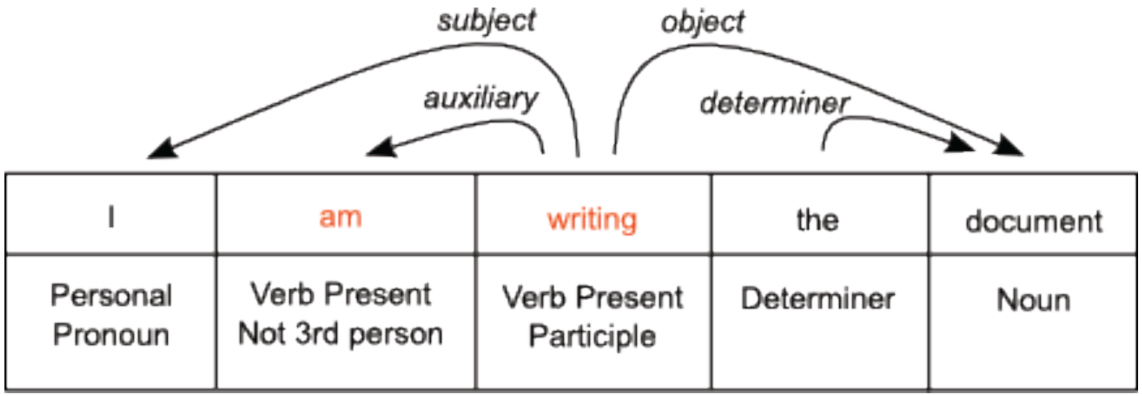
bigrams Th hx x_ _f or r_ _m ms sg g, ,_ … 8r

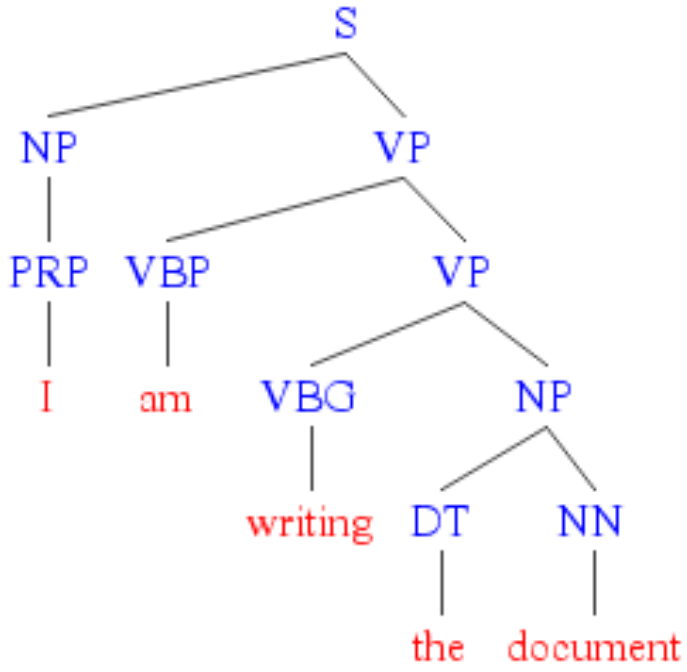
trigrams Thx hx_ x_f _fo or_ r_m _ms msg sg, … l8r

4-grams… Thx_ hx_f x_fo _for or_m r_ms _msg … _l8r

Text as Rich Data

- Natural Language Processing (NLP) tools such as parsers offer the power to create deeper linguistic features





- Part of speech (POS) tagging involves tagging a word with its part of speech
  - Separate '*book a holiday*' from '*read a book*'

Representation Challenges

Shallow Representation

Deep Representation

Low Effort

High Effort

Robust

Brittle

Other Features

- Consider using non-textual features such as
  - Structural features: headings, #words, #uppercase chars …
  - Vocabulary richness measures: type token ratio, readability…
  - Context: date, author, topic, location, meta-data…
  - Domain specific features: URL, @, #,…

Spam Filtering

Plagiarism Detection

Genre Identification

Medical Decision Support

Search

Question Answering

Topic Modelling

Information Retrieval

Spam Filtering

Language Identification

Sentiment Analysis

Author Attribution

Text Summarisation

Applications

Techniques

Lexicons

Parsing

Named Entity Recognition

Dictionaries

Embeddings

Textual Entailment

Part of Speech Tagging

Semantic Role Labelling

Co-reference Resolution

Latent Semantic Indexing

Working with Text

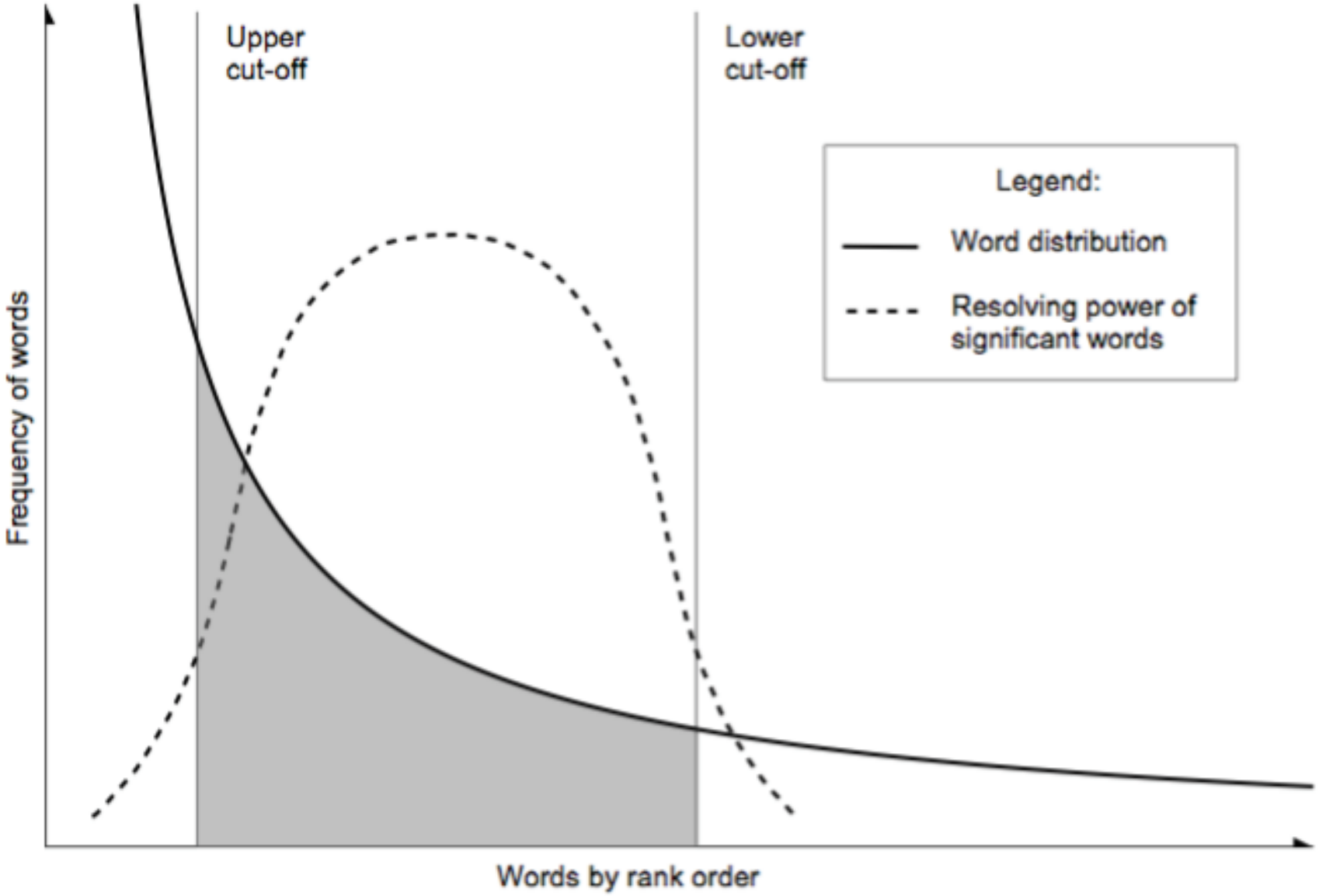
- Pre-processing of text data requires consideration of:
  - language identification?
  - normalisation e.g. reduce everything to lower case
  - tokenisation - determining how to split content into ‘features’
  - how to handle punctuation
  - how to domain specific features, e.g. URLs, hashtags, emoticons, etc…
  - what representation to use?

- Text data is sparse - no missing value issues
- Text data has high dimensionality - consider the size of the vocabulary of all unique words in a corpus of 100,000 documents

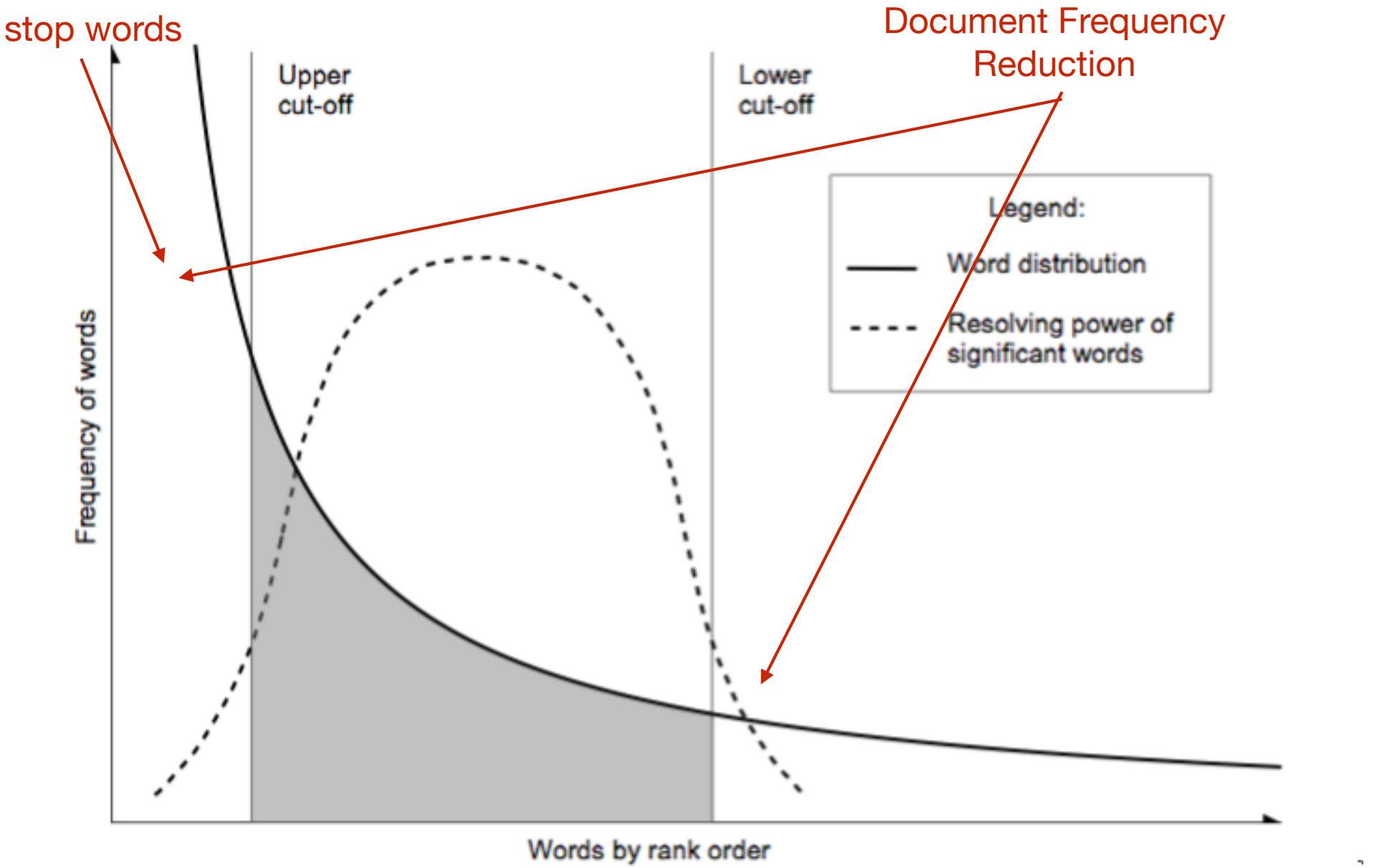
Dimensionality Reduction

- In BOW not all words are useful:
  - common words such as in, of, a, the,… called stopwords
  - infrequent words such as spelling mistakes
  - words which mean the same thing, e.g. walking, walk, walker… or better, good
- Can use stemming / lemmatisation to map similar words to one root
- For more general representations use Document Frequency Reduction
  - remove all features that occur in, e.g. >99.99% of docs
  - remove all features that occur in, e.g. < 3 docs
- Use Filter Feature Selection to reduce the dimensionality of the data

Word Frequency vs. Resolving Power



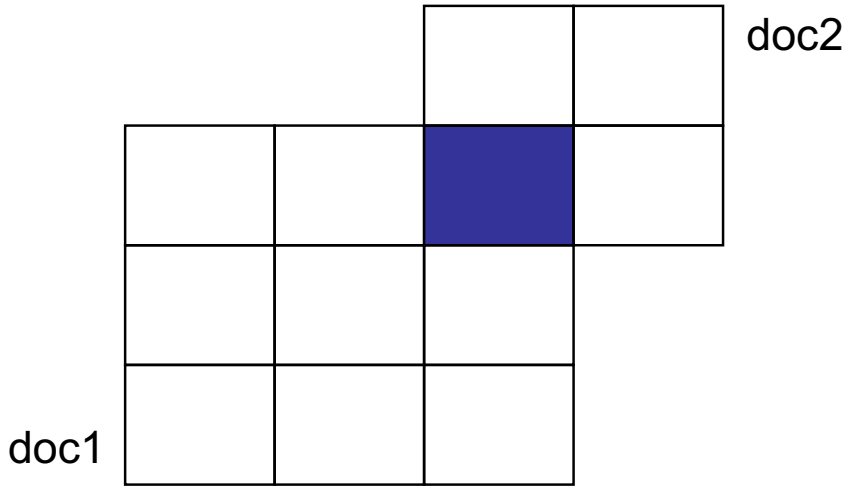
Word Frequency vs. Resolving Power



Measuring Similarity in Text

- Term weight is numeric but text representation is sparse
- Similarity is measured as the size of the intersection of two documents

$$Sim(d_1, d_2) = d_1 \cap d_2$$



- Quantified as the dot product of the vectors representing the docs

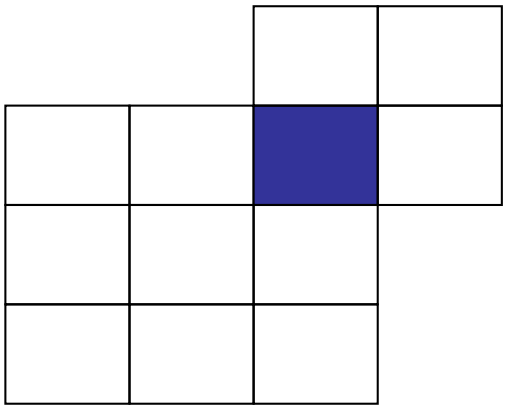
$$Sim(d_1, d_2) = \sum_{i=1}^{n} w_{d_1 i} \times w_{d_2 i}$$

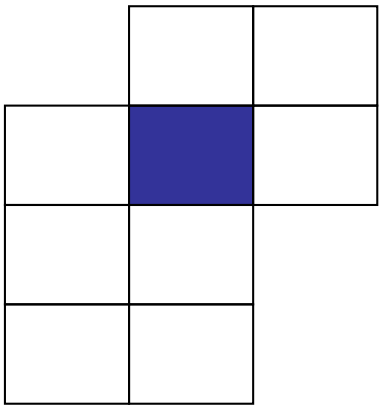
term weight of term in doc2

term weight of term in doc1

Measuring Similarity in Text

- Intersection doesn't take into account document sizes





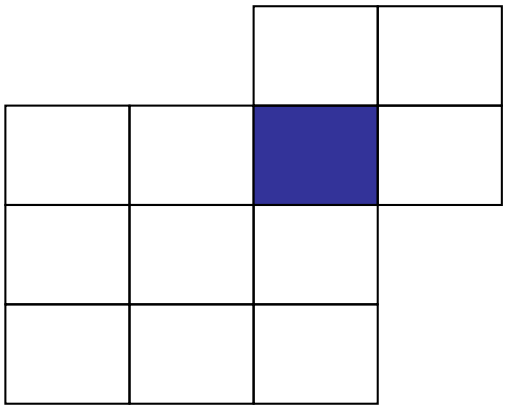
- Normalise by document lengths

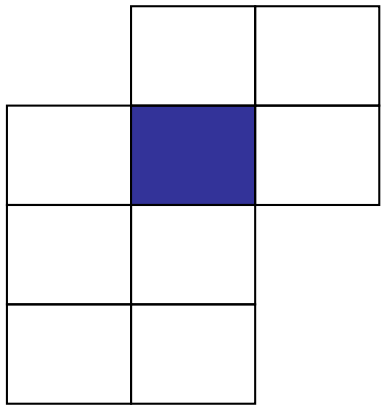
$$Sim(d_1, d_2) = \frac{d_1 \cap d_2}{|d_1||d_2|}$$

?

Measuring Similarity in Text

- Intersection doesn't take into account document sizes





- Normalise by document lengths

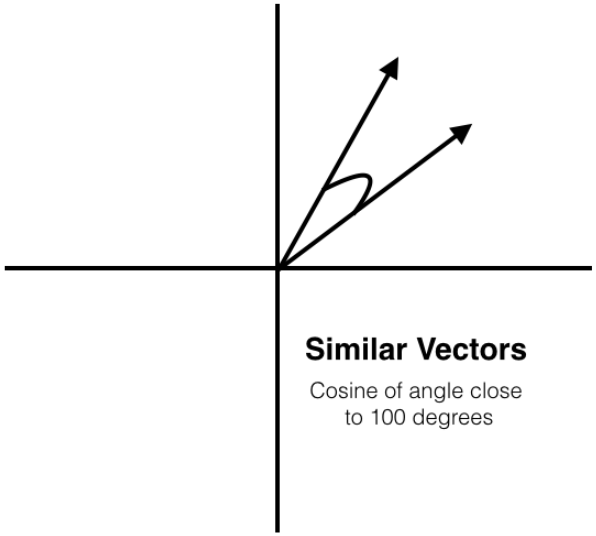
$$Sim(d_1, d_2) = \frac{d_1 \cap d_2}{|d_1||d_2|}$$

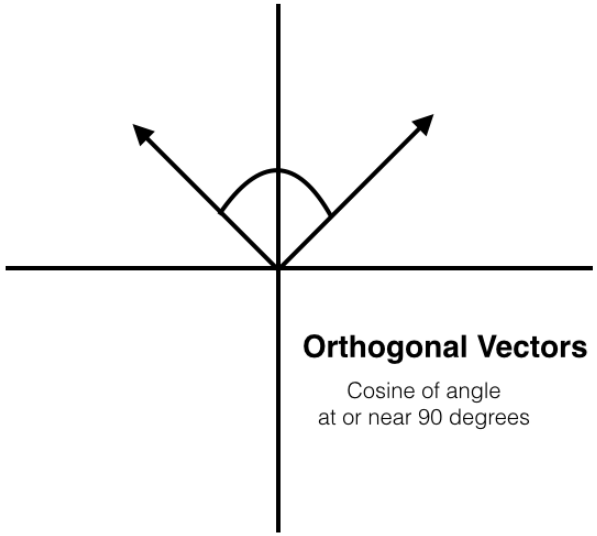
Cosine Similarity

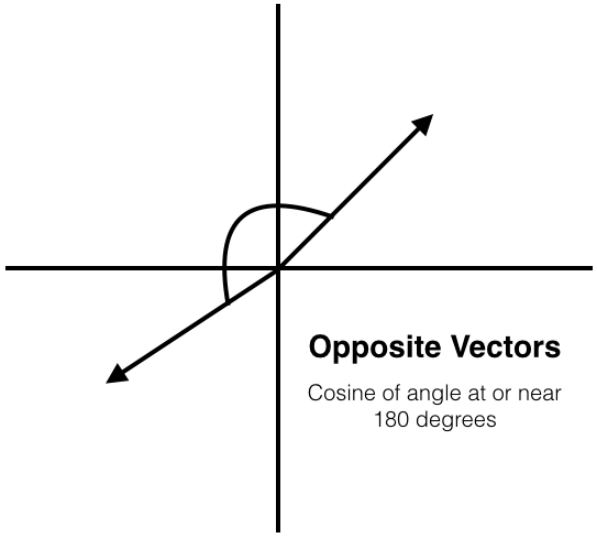
$$Sim(d_1, d_2) = \frac{\sum_{i=1}^{n} w_{d_1 i} \times w_{d_2 i}}{\sqrt{\sum_{i=1}^{n} (w_{d_1 i})^2} \sqrt{\sum_{i=1}^{n} (w_{d_2 i})^2}}$$

Cosine Similarity

- Measures the angle between two vectors







Text Classification in scikit-learn

- Tokenising

```
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
count_vect.get_params()      #shows the default parameters
```

```
{'analyzer': 'word',
 'binary': False,
 'decode_error': 'strict',
 'dtype': numpy.int64,
 'encoding': 'utf-8',
 'input': 'content',
 'lowercase': True,
 'max_df': 1.0,
 'max_features': None,
 'min_df': 1,
 'ngram_range': (1, 1),
 'preprocessor': None,
 'stop_words': None,
 'strip_accents': None,
 'token_pattern': '(?u)\\b\\w\\w+\\b',
 'tokenizer': None,
 'vocabulary': None}
```

Different kinds of tokenising

Code Notebook:
10 Text Classification

Text Classification in scikit-learn

- Dimension Reduction

```
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
count_vect.get_params()      #shows the default parameters
```

```
{'analyzer': 'word',
 'binary': False,
 'decode_error': 'strict',
 'dtype': numpy.int64,
 'encoding': 'utf-8',
 'input': 'content',
 'lowercase': True,
 'max_df': 1.0,
 'max_features': None,
 'min_df': 1,
 'ngram_range': (1, 1),
 'preprocessor': None,
 'stop_words': None,
 'strip_accents': None,
 'token_pattern': '(?u)\\b\\w\\w+\\b',
 'tokenizer': None,
 'vocabulary': None}
```

Document Frequency Reduction

Stopwords

Text Classification in scikit-learn

- Term-Doc Matrix

```
tdm = count_vect.fit_transform(twentyTrain.data)

#tdm is a matrix - 2-d array
tdm.shape
```

```
Out[11]: (2321, 31164)
```

```
#count_vect is a dictionary
#show the freq of word 'and'
count_vect.vocabulary_.get('and')
```

```
Out[12]: 4204
```

Document Frequency Reduction

Text Classification in scikit-learn

- Transform the raw frequencies

```
In [13]: from sklearn.feature_extraction.text import TfidfTransformer
         transformer = TfidfTransformer()
         transformer.get_params()     #show default parameters

Out[13]: {'norm': 'l2', 'smooth_idf': True, 'sublinear_tf': False, 'use_idf': True}

In [14]: tdm_tfidf = transformer.fit_transform(tdm)  #transform the TDM
         tdm_tfidf.shape

Out[14]: (2321, 31164)
```

```
TfidfVectoriser =
        CountVectoriser
        + TfidfTransformer
```

- Build the classifier

```
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(tdm_tfidf, twentyTrain.target)
```

- Transform the test data

```
test_counts = count_vect.transform(docs_test)
test_tfidf = transformer.transform(test_counts)
```

Word Context

"You shall know a word by the company it keeps"

(Firth 1957)

A bottle of *tesgüino* is on the table

Everybody likes *tesgüino*

tesgüino makes you drunk

We make *tesgüino* out of corn

(Nida 1975)

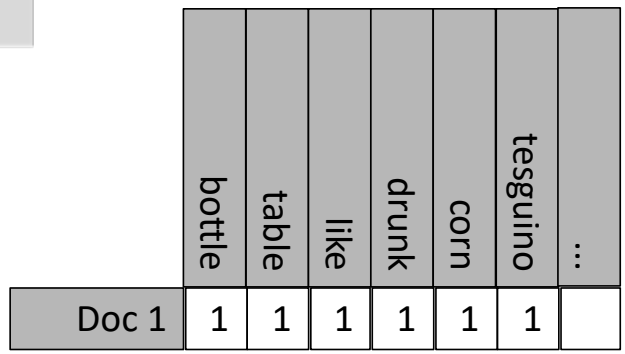
Word Vectors

A bottle of *tesgüino* is on the table

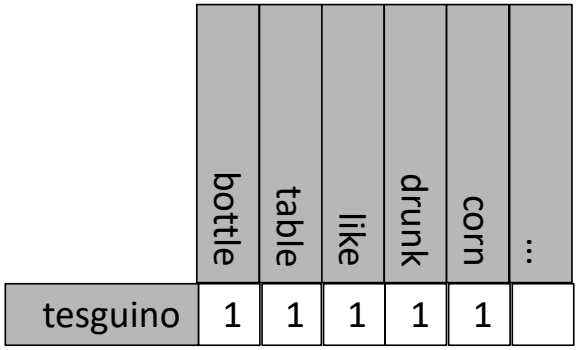
Everybody likes *tesgüino*

tesgüino makes you drunk

We make *tesgüino* out of corn



document focus



word focus

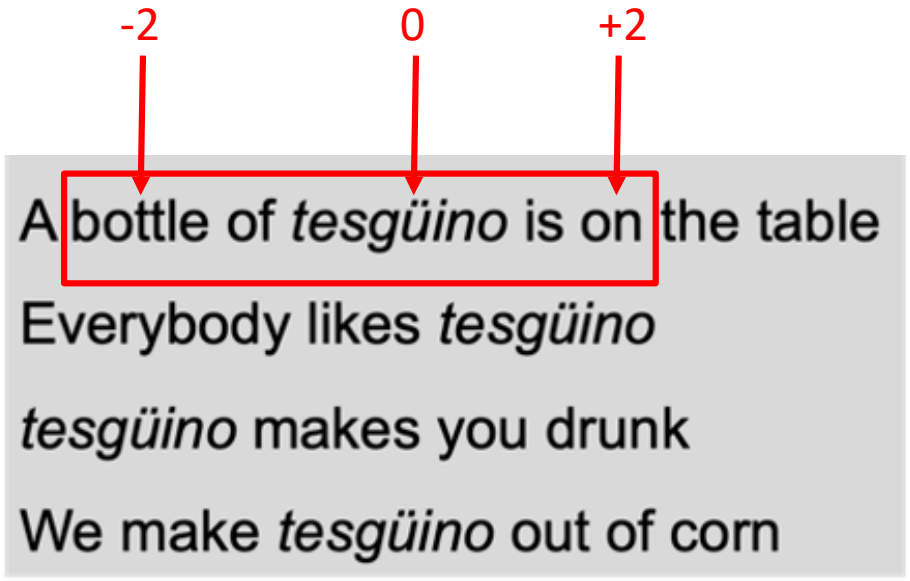
Word Vectors

- Distributed Representation models the meaning of a word by *embedding* in a vector space
- Different approaches:
  - Co-occurence
  - SVD and LSA
  - ANN inspired models

Co-occurrence

- Create a co-occurrence matrix X which will be a $d \times d$ matrix where d is the vocabulary / number of features in the term document matrix

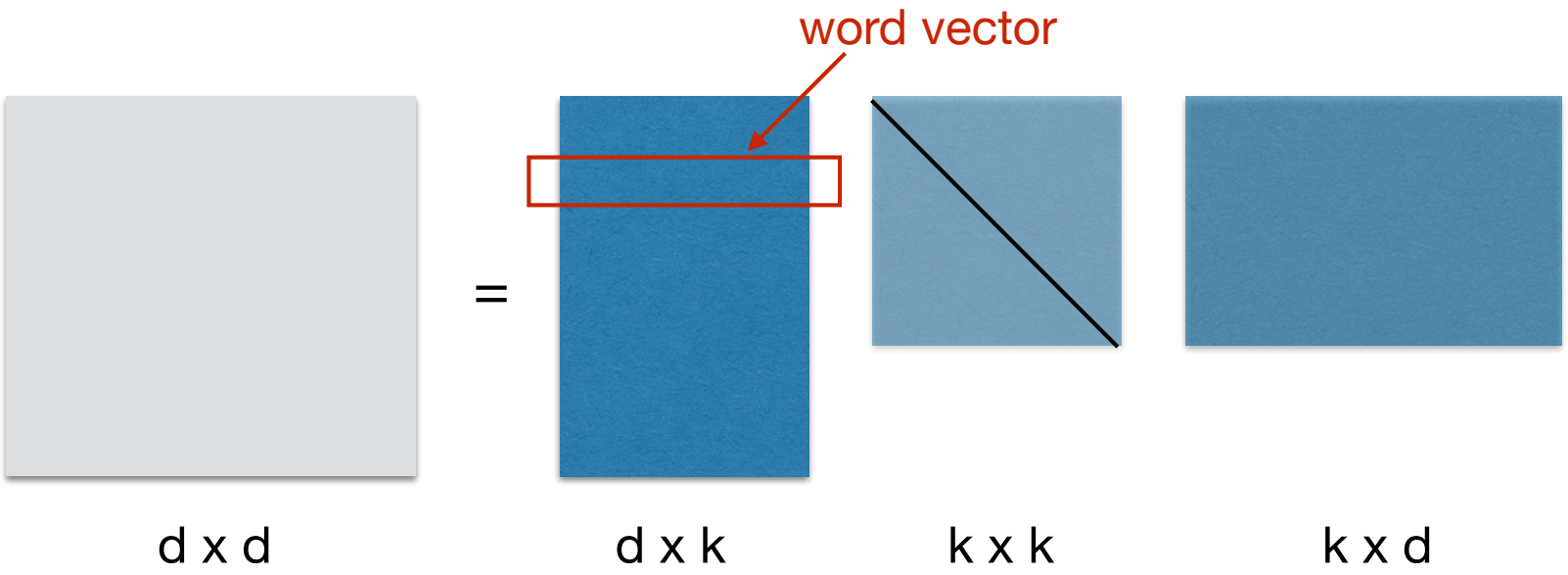
	bottle	table	like	drunk	corn	…
tesguino	1	1	1	1	1	



- Row i represents the word context vector (representation) for word i
- Result is a sparse vector with high dimensionality

Singular Value Decomposition

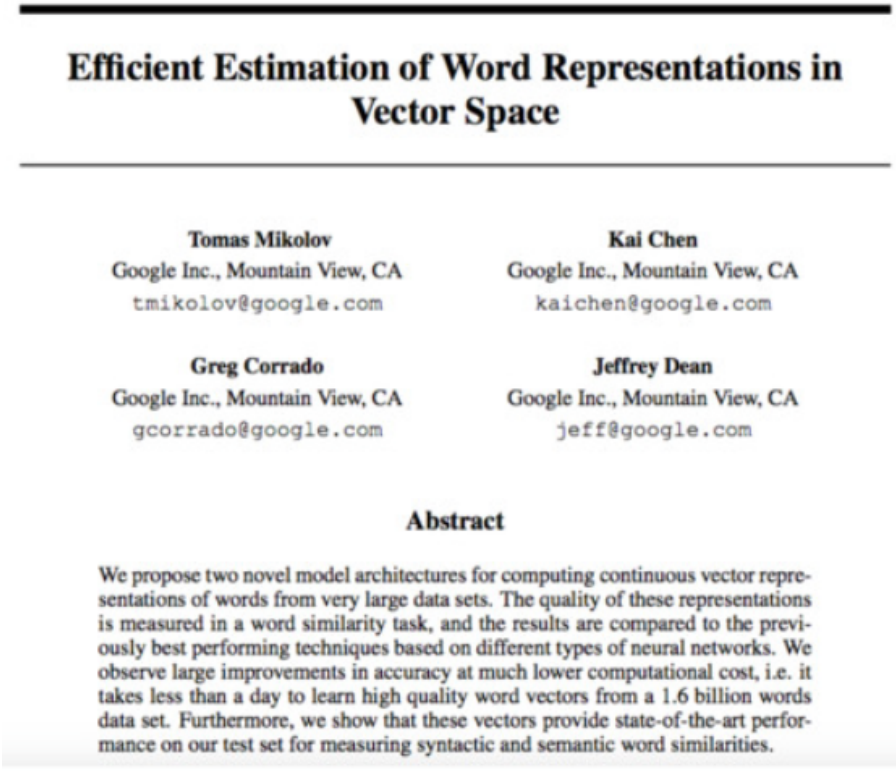
- Use SVD to factorise the co-occurence matrix
- Word vector is reduced to k dimensions



- Global Vectors (GloVe) embeddings use this approach
  https://nlp.stanford.edu/projects/glove/

Word Embeddings

Using Neural Network Prediction Models, e.g. word2Vec [Mikolov 2013]



Efficient Estimation of Word Representations in Vector Space

Tomas Mikolov
Google Inc., Mountain View, CA
tmikolov@google.com

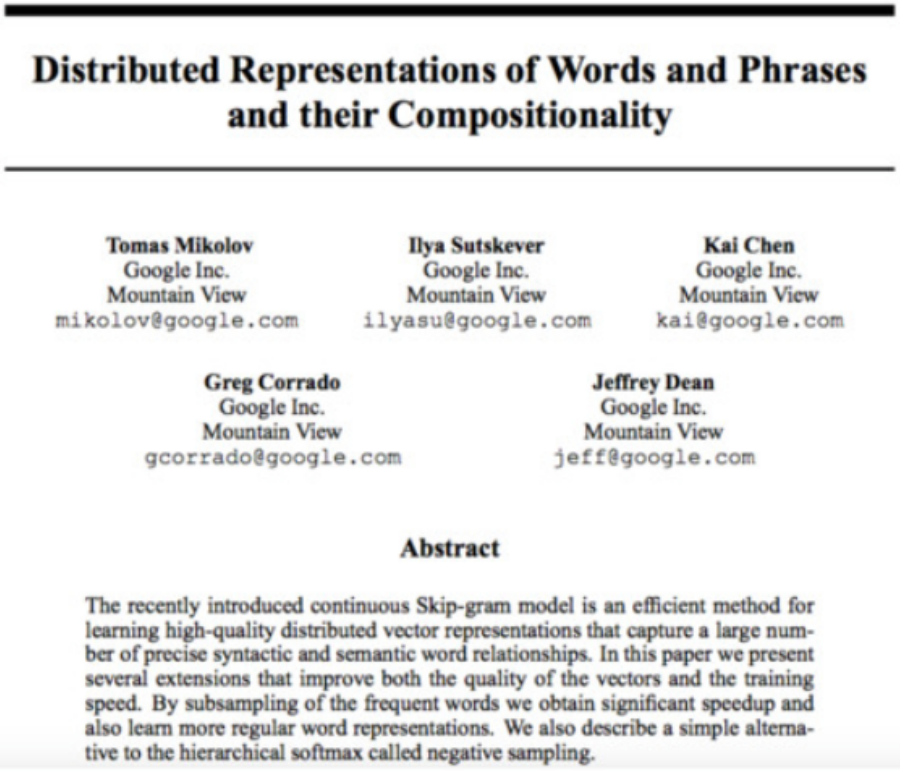
Kai Chen
Google Inc., Mountain View, CA
kaichen@google.com

Greg Corrado
Google Inc., Mountain View, CA
gcorrado@google.com

Jeffrey Dean
Google Inc., Mountain View, CA
jeff@google.com

Abstract

We propose two novel model architectures for computing continuous vector representations of words from very large data sets. The quality of these representations is measured in a word similarity task, and the results are compared to the previously best performing techniques based on different types of neural networks. We observe large improvements in accuracy at much lower computational cost, i.e. it takes less than a day to learn high quality word vectors from a 1.6 billion words data set. Furthermore, we show that these vectors provide state-of-the-art performance on our test set for measuring syntactic and semantic word similarities.



Distributed Representations of Words and Phrases and their Compositionality

Tomas Mikolov
Google Inc.
Mountain View
mikolov@google.com

Ilya Sutskever
Google Inc.
Mountain View
ilyasu@google.com

Kai Chen
Google Inc.
Mountain View
kai@google.com

Greg Corrado
Google Inc.
Mountain View
gcorrado@google.com

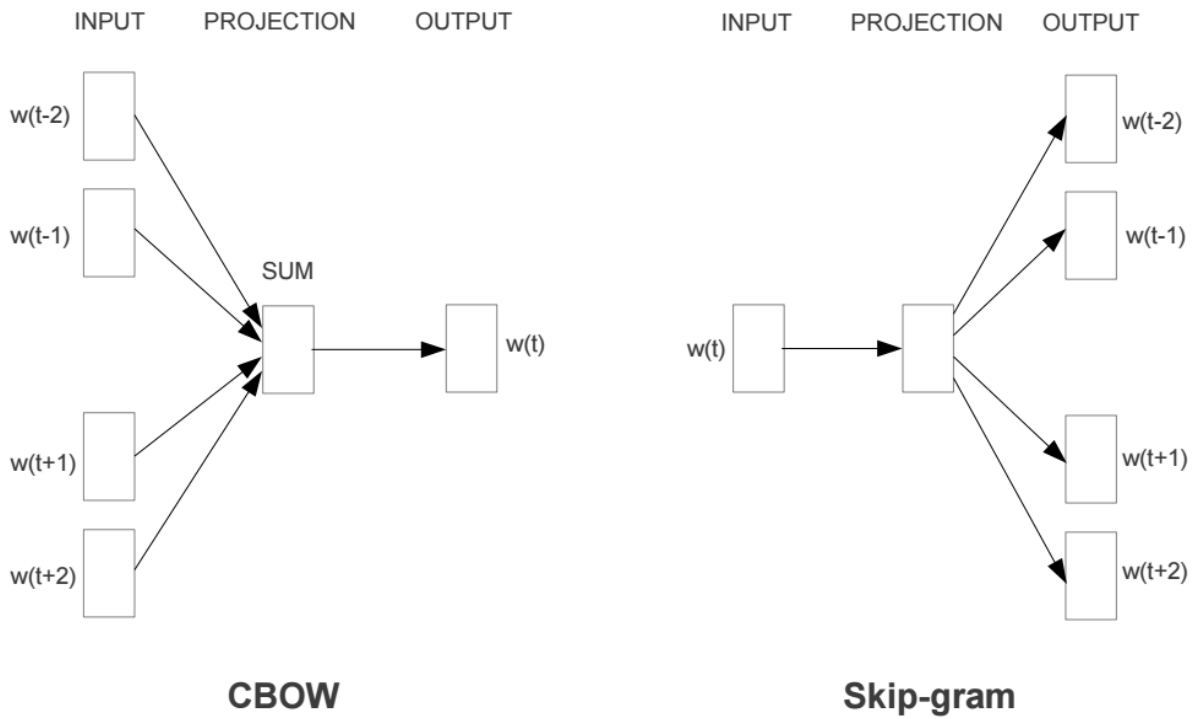
Jeffrey Dean
Google Inc.
Mountain View
jeff@google.com

Abstract

The recently introduced continuous Skip-gram model is an efficient method for learning high-quality distributed vector representations that capture a large number of precise syntactic and semantic word relationships. In this paper we present several extensions that improve both the quality of the vectors and the training speed. By subsampling of the frequent words we obtain significant speedup and also learn more regular word representations. We also describe a simple alternative to the hierarchical softmax called negative sampling.

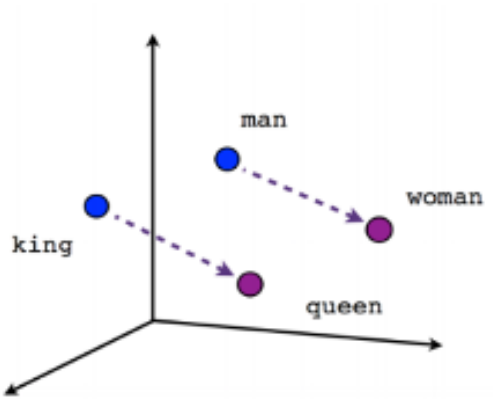
word2Vec

- Two approaches
  - Continuous Bag of Words - predict the probability of a target word give context words
  - Skip-gram - given a target word, predict the probability of the context words

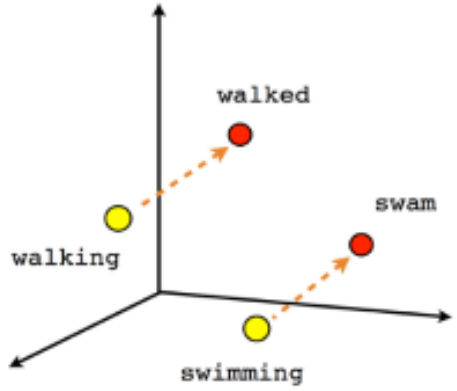


Pre-trained Embeddings

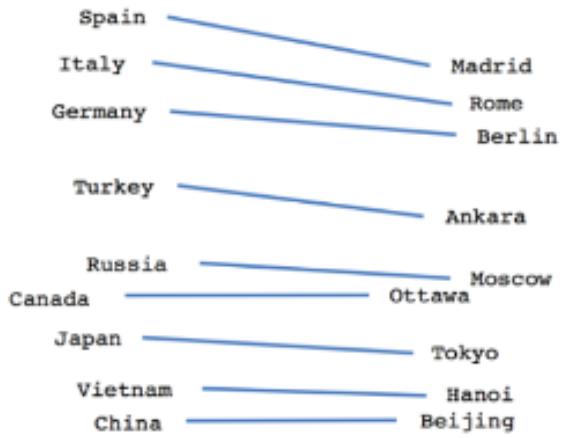
- word2Vec trained on Google News
- Relationships are learned from the data
- Algebraic computations are possible on the vectors



Male-Female



Verb tense



Country-Capital

Word2Vec

King – man + woman = ?

('queen', 0.7118192911148071),
('monarch', 0.6189674139022827),
('princess', 0.5902431607246399)...

Paris – France + Ireland = ?

('Dublin', 0.7617993354797363),
('Cork', 0.6133344769477844),
('Limerick', 0.5828378200531006),
('Galway', 0.582069456577301)…

Pretrained Vectors

King – man + woman = ?

('queen', 0.7118192911148071),
('monarch', 0.6189674139022827),
('princess', 0.5902431607246399)...

Paris – France + Ireland = ?

('Dublin', 0.7617993354797363),
('Cork', 0.6133344769477844),
('Limerick', 0.5828378200531006),
('Galway', 0.582069456577301)…

Computer Programmer – man + woman = ?

Word2Vec

Code Notebook:
10 Embeddings

King – man + woman = ?

('queen', 0.7118192911148071),
('monarch', 0.6189674139022827),
('princess', 0.5902431607246399)...

Paris – France + Ireland = ?

('Dublin', 0.7617993354797363),
('Cork', 0.6133344769477844),
('Limerick', 0.5828378200531006),
('Galway', 0.582069456577301)…

Computer Programmer – man + woman = ?

('homemaker', 0.5627118945121765),
('housewife', 0.5105046033859253),
('graphic_designer', 0.5051802396774292),
('schoolteacher', 0.497949481010437),
('businesswoman', 0.4934891760349273 7)…

Word Embeddings

- Pretrained embeddings can have out-of-vocabulary (OOV) words

- fastText (Facebook Research) uses character embeddings, can handle out-of-vocabulary words, good for mis-spellings and is fast [Joulin et al. 2016]

Contextual Embeddings

"I left my pen on the left hand side of the desk"

- *Contextualised Word Embeddings* consider context
- Use language models that take word order into account
- e.g. Embeddings from Language Models (ELMo)
  [Peters et al. 2018]

Sentence Embeddings

- Provide a vector representation for a sentence, rather than an individual word
- No look-up capability
- Averaging the word vectors is a simple but effective approach
- Other solutions require pre-trained models that accept sentences and provide embeddings
  - Unsupervised approach: e.g. Skip-thought (similar to skip-gram) [Kiros et al. 2015]
  - Supervised approach: e.g. InferSent [Conneau et al. 2017]
  - Multitask Learning: a generalisation, trying to combine several training objectives in one training scheme e.g. Universal Sentence Encoder [Cer et al. 2018]

Transfer Learning for NLP

- Transfer Learning is a machine learning method where a model trained for one task in one domain is used for a different task in a different domain
- E.g. Know how to ride a motorcycle —> Learn how to drive a car
- The knowledge from the first task is transferred and leverages the second task
- Transfer learning in NLP
  - Step 1: Train a Deep Learning model on a large unlabelled text corpus (can be supervised or unsupervised training)
  - Step 2: Fine tune the model on specific tasks
  - e.g. Bi-directional Encoder Representational from Transformers

BERT

Summary

- Text data is unstructured and requires pre-processing to convert to a structured representation that algorithm can work on
- Traditional content based representations are typically frequency-based, e.g. tf, tf-idf
- Linguistic approaches offer use of richer, more structured representations which are less robust and computationally more difficult
- Distributed representations take into account context
- Text data has very high dimensionality so dimensionality reduction techniques are needed
  - Simplest are approaches such as stop word removal and document frequency reduction
  - Filter feature selection techniques can often help