

Natural Language Processing

Unit 1 – Words

Anantharaman Narayana Iyer

narayana dot anantharaman at gmail dot com

7th and 8th Aug 2015

References

Efficient Estimation of Word Representations in Vector Space

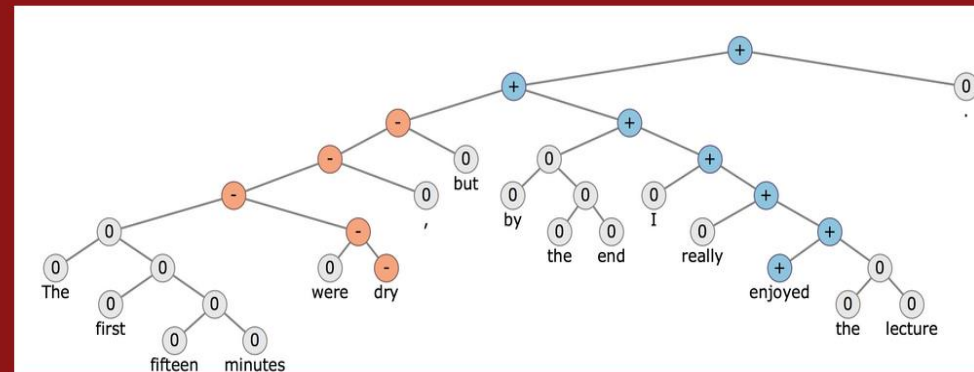
Google Inc., Mountain View, CA
gcorrado@google.com

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

Distributed Representations of Words and Phrases and their Compositionality

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

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




Distributed Representations of Sentences and Documents

QVL@GOOGLE.COM
TMIKOLV@GOOGLE.COM

Google Inc, 1600 Amphitheatre Parkway, Mountain View, CA 94043

Tools pertaining to this lecture

 **ananthpn / nlp**

 Watch 

1 commit


1 branch

0 releases

1 contributor

Branch: **master** **nlp** / +

Cosine class version 0.1

 **ananthpn** authored on 1 Sep 2013 latest commit 3326b62def

 .gitattributes	Cosine class version 0.1	2 years ago
 .gitignore	Cosine class version 0.1	2 years ago
 cosine.py	Cosine class version 0.1	2 years ago
 readme.md	Cosine class version 0.1	2 years ago

 **readme.md**

Cosine - The purpose of this module is to make it easy to evaluate cosine similarity for a set of text sentences. This is a helper that invokes NLTK library for cosine similarity but makes it simple by:

Branch: **master** **sentence2vec** / +

some fix. add test files for demo

 **kib3713** authored on 23 Sep 2014 latest commit e38a4f5ad4

 README.md	some fix. add test files for demo	11 months ago
 demo.py	init project	a year ago
 matutils.py	init project	a year ago
 sent.txt	some fix. add test files for demo	11 months ago
 test.txt	some fix. add test files for demo	11 months ago
 utils.py	init project	a year ago
 voidptr.h	init project	a year ago
 word2vec.py	some fix. add test files for demo	11 months ago
 word2vec_inner.pyx	init project	a year ago

 **README.md**

sentence2vec

Branch: **master** **word2vec** / +

Merge branch 'master' of github.com:danielfrg/word2vec

 **danielfrg** authored 16 days ago latest commit b97bb84a3f

 examples	experimental doc2vec	4 months ago
 word2vec-c	experimental doc2vec	4 months ago
 word2vec	Merge branch 'master' of github.com:danielfrg/word2vec	16 days ago
 .env	Bump to 0.8 and add some dev files	16 days ago
 .gitignore	ignore data	a year ago
 .travis.yml	fix pushd	4 months ago
 LICENSE.txt	Initial commit	2 years ago
 MANIFEST.in	remove make file	4 months ago
 README.md	os support note	4 months ago
 requirements.txt	Bump to 0.8 and add some dev files	16 days ago
 setup.py	Bump to 0.8 and add some dev files	16 days ago

 **README.md**

word2vec

Natural Language Toolkit

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, and an active discussion forum.

Foundation

Words and Sentences

- Text documents are constituted by words and sentences
- The words and their composition provide us the meaning
- Each class of documents (for example: technology, sports, political, films etc) have some common words and also some distinguishing words. For example, the term LBW is more likely to occur in a document pertaining to cricket. Likewise terms like: comedy, hero, climax etc occur often in film reviews.
- By analysing the occurrence and frequency of words we can predict which document the words came from.
- Notion of: term frequency (tf), inverse document frequency (idf) and tf-idf

Some basic concepts to get started

- We will now introduce a few concepts most of which will be covered in detail in later classes
 - Word and Sentence segmentation
 - Pre-processing the text and Normalization: stop words removal, stemming, lemmatization
 - Term Frequency (tf)
 - Inverse document Frequency (idf)
 - Tfidf
 - Bag of words (BOW)
 - Vector Space models
 - Cosine Similarity
- Language Models

Vocabulary – ref: Dan Jurafsky and Christopher Manning

N = number of tokens

V = vocabulary = set of types

$|V|$ is the size of the vocabulary

Church and Gale (1990): $|V| > O(N^{\frac{1}{2}})$

	Tokens = N	Types = $ V $
Switchboard phone conversations	2.4 million	20 thousand
Shakespeare	884,000	31 thousand
Google N-grams	1 trillion	13 million

Issues in tokenization - Dan Jurafsky and Christopher Manning

- Finland's capital → Finland Finlands Finland's ?
- what're, I'm, isn't → What are, I am, is not
- Hewlett-Packard → Hewlett Packard ?
- state-of-the-art → state of the art ?
- Lowercase → lower-case lowercase lower case ?
- San Francisco → **one token or two?**
- m.p.h., PhD. → ??

A motivating example: Pre-processing and Text Normalization challenges from tweet data

- Consider the following tweets:
 - GOD's been very kind as HE didn't create FRIENDS with price tags.If He did, I really couldn't afford U [#HappyFriendshipDay](#) [@KimdeLeon](#)
 - Why are wishing friendship day today? Aren't we spps to be friends for all the 365 days?[#HappyFriendshipDay](#)
 - eVrYthin ChnGZ & nthin StaYs D SMe bt aS v grOw uP 1 thnG vl reMain i ws wT u b4& vL b tiLl D end [#HappyFriendshipDay](#) [pic.twitter.com/Ym3ZAnFiFn](#)
- How do we tokenize, normalize the above text?
 - Often the answer to the above is application dependent

Words

- How do we identify word boundaries and tokenize the words?
 - Don't, would've, b4 (if we have a rule/logic where we consider valid English words to consist only of alphabets [a-zA-Z], then the word boundary would break at the letter 4)
- Normalizing lowercase, uppercase
 - Some news headlines may be capitalized: "PM Modi Announces US \$1 Billion Concessional Line of Credit to Nepal"
 - Some tweets may be in all capital letters
- Normalization to same word expressed with differing syntax
 - This is needed for applications like information retrieval so that the query string can be more appropriately matched with the content being searched - E.g, U.K and UK may be resolved to one term, similarly Mr and Mr., Ph.D, PhD etc
- How do we handle words that are distorted versions of a valid English word?
 - eVrYthin, ChnGZ, D, etc – if we have a vocabulary of a given corpus that has the terms everything, changes, the would we count the abbreviated terms same as the valid ones or would we consider them different?
 - "I am tooooooooooo happpppppppppppy" – If we correct these in to: "I am too happy" we may lose the implied and hidden emotion of the strength of happiness expressed in the phrase.
- How do we handle new words not part of the language?
 - Tweepie, selfie, bestie etc – consider a spell checker application that we would like to build. Would we correct these? Are these candidate correction words of some one's misspells, eg, Selfy instead of selfie?

Sentence Segmentation

- Sentence segmentation is concerned about splitting the given input text into sentences.
- Characters like !, ?, . may indicate sentence boundaries
- However, there could be ambiguity, for instance:
 - The quarterly results of Yahoo! showed a promising trend
 - Microsoft announced updates to .NET framework
 - Indian rupee appreciated by 0.5% against USD during today's trade
 - Who moved my cheese? is a great read.
- The ambiguity may also arise due to spelling mistakes
- Possible ways to resolve:
 - Traditional rule based regular expressions
 - Decision trees encoding some rules
 - Classifiers

Challenges with social data

1	i	17,071,657
2	-i	4,254
3	ijust	1,446
4	dontcha	872
5	ireally	705
6	d'you	527
7	whaddya	511
8	istill	477
9	//i	438
10	ijus	424
11	i-i	393
12	inever	362
13	#uever	347

1	haven't	170,431
2	havent	38,303
3	shoulda	14,114
4	would've	13,043
5	should've	12,997
6	hadn't	11,899
7	woulda	10,061
8	could've	7,435
9	coulda	6,009
10	havnt	5,227
11	shouldve	3,972
12	wouldve	3,752
13	must've	3,164
14	musta	2,159
15	couldve	2,142
16	haven't	...

1	love	1,908,093
2	luv	58,486
3	lovee	9,927
4	envy	9,668
5	salute	5,629
6	loveee	5,107
7	lov	3,625
8	dread	3,188
9	looove	3,109
10	cba	2,975
11	loveeee	2,932
12	loooove	2,740
13	loove	1,664
14	loooooove	1,584
15	loveeeee	1,435
16	lurve	868
17	looooooove	849
18	l0ve	712
19	loveeeeeee	667
20	luff	626
21	l.o.v.e	585
22	lovelovelove	583
23	luvv	558

Some key challenges are:

- How do we tokenize words like: ireally, l0ve, #unever etc.
- How to handle tagging problems like: POS tagging, NE tagging and so on
- More broadly: How to address the traditional NLP core tasks and applications?

Tokenization - Tools

Tweet NLP



Carnegie Mellon

We provide a tokenizer, a part-of-speech tagger, hierarchical word clusters, and a dependency parser for tweets, along with annotated corpora and web-based annotation tools.

Contributors: Archana Bhatia, Dipanjan Das, Chris Dyer, Jacob Eisenstein, Jeffrey Flanigan, Kevin Gimpel, Michael Heilman, Lingpeng Kong, Daniel Mills, Brendan O'Connor, Olutobi Owoputi, Nathan Schneider, Noah Smith, Swabha Swayamdipta and Dani Yogatama.

Quick Links

- [Part-of-speech Tagger and POS annotated data](#) - also [Ttokenizer](#): tokenizer software (part of tagger package) and [Tagging Models](#) -- [Download Link](#)
- [Tweeboparser and Twebank](#): Dependency parser software and dependency annotated data -- [Download Link](#)
- [Documentation, annotation guidelines, and papers](#) describing this work
- [Hierarchical Twitter Word Clusters](#)



The Stanford Natural Language Processing Group

[home](#) · [people](#) · [teaching](#) · [research](#) · [publications](#) · [software](#) · [events](#) · [local](#)

```
>>> zen = TextBlob("Beautiful is better than ugly. "  
...           "Explicit is better than implicit. "  
...           "Simple is better than complex.")  
>>> zen.words  
WordList(['Beautiful', 'is', 'better', 'than', 'ugly', 'Explicit', 'is', 'better',  
>>> zen.sentences  
[Sentence("Beautiful is better than ugly."), Sentence("Explicit is better than imp
```

NLTK 3.0 documentation

[PREVIOUS](#) | [MODULES](#) | [INDEX](#)

nlk.tokenize package

Stanford Tokenizer

[About](#) | [Obtaining](#) | [Usage](#) | [Questions](#) | [Mailing Lists](#)

About

A tokenizer divides text into a sequence of tokens, which roughly correspond to "words". We provide a class suitable for tokenization of English, called PTBTokenizer. It was initially designed to largely mimic [Penn Treebank 3](#) (PTB) tokenization hence its name, though over time the tokenizer has added quite a few options and a fair amount of Unicode compatibility, so in general it will work well over text encoded in the Unicode Basic Multilingual Plane that does not require word segmentation (such as writing systems that do not put spaces between words) or more exotic language-particular rules (such as writing systems that use : or ? as a character inside words, etc.). An ancillary tool uses this tokenization to provide the ability to split text into sentences. PTBTokenizer mainly targets formal English writing rather than SMS-speak.

PTBTokenizer is an efficient, fast, deterministic tokenizer. (For the more technically inclined, it is implemented as a finite automaton, produced by [JFlex](#).) On a 2015 laptop computer, it will tokenize text at a rate of about 1,000,000 tokens per

tfidf

- Term frequency denotes the count of occurrences of a given term in the given document:
 - Term frequency of a term t in a document d : $tf(t, d)$
 - Term Frequency signifies the importance of the given term t in a given document d
- Suppose we have N documents and suppose we are counting the occurrence of a term t across all the N documents. The Inverse Document Frequency (idf) is the log of ratio of number of documents to those that contain the term t .
 - IDF of a term t across N documents: $idf(t, D) = \log(N / D)$ where D = set of all documents containing the term t
 - IDF indicates whether the given term t is a common term that may be found in a large number of documents in the corpus of N documents or it is a rare term found only in a small subset of documents in the corpus
 - Thus, IDF is a measure of how much information the given term provides. A common term provides very less information while terms that are unique to a given document help distinguish the documents.
- Tfidf is the product of tf and idf
 - $tfidf(t, d, D) = tf(t, d) * idf(t, D)$
 - Tfidf assigns a weight to term t in a document d , where d belongs to a corpus of N documents
 - Terms with relatively high tfidf help discriminate one document from the other
 - Given a query term and document d , we can compute a score:

$$\text{Score}(q, d) = \sum_{t \in q} \text{tf-idf}_{t,d}.$$

Tfidf Illustration

Anantharaman Narayana Iyer

Narayana dot Anantharaman at gmail dot com

13 Aug 2015

Algorithm: Computing document ranks with respect to a query

Input:

- Query String
- Corpus of N documents

Output:

- Rank list : $[(d_i, s_i), (d_j, s_j), \dots]$ # d_i is the top ranked document, d_j has rank 2 etc

1. Word tokenize the query to a set of words: q_1, q_2, \dots, q_k
2. Set ranks to an empty list # $\text{ranks} = []$: output will be produced here
3. Compute for each document d_i in the document corpus D:
 1. Initialize score of $d_i = 0$
 2. Compute for each query term t in q :
 1. $n = \text{Tf}(t, d_i)$ # compute term frequency
 2. $m = \text{idf}(t, D)$ # compute IDF for t across documents: **NOTE: If needed add 1 smoothing**
 3. $\text{tfidf} = n * m$
 4. Add tfidf to score of d_i
 3. Add the tuple (index of d_i , score for d_i) to ranks # (d_i, s_i)
4. Sort the ranks list
5. Return ranks

Rank ordering the 2 documents

- Consider the two documents shown in the next slide
- Let the query string be: $q = \text{"Python Web Server"}$
- The above query is made up of 3 words: (Python, Web, Server)
- We are required to rank the documents in terms of match for this query
 - `Compute_ranks(q, documents)` : returns the rank of the documents

Illustration of tfidf computation

HOWTO Use Python in the web

Author: Marek Kubica

Document 1: From Python official web site

Abstract

This document shows how Python fits into the web. It presents some ways to integrate Python with a web server, and general practices useful for developing web sites.

Programming for the Web has become a hot topic since the rise of "Web 2.0", which focuses on user-generated content on web sites. It has always been possible to use Python for creating web sites, but it was a rather tedious task. Therefore, many frameworks and helper tools have been created to assist developers in creating faster and more robust sites. This HOWTO describes some of the methods used to combine Python with a web server to create dynamic content. It is not meant as a complete introduction, as this topic is far too broad to be covered in one single document. However, a short overview of the most popular libraries is provided.

Web server

From Wikipedia, the free encyclopedia

Document 2: From Wikipedia



This article **needs additional citations for verification**. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed. (March 2009)

The term **web server**, also written as **Web server**, can refer to either the **hardware** (the computer) or the **software** (the computer application) that helps to deliver **web content** that can be accessed through the **Internet**.^[1]



Tfidf computation and rank ordering

Query term	Tf(t, d1)	Tf(t, d2)	Tfidf(t, D)	Tfidf(t, D)
Python	4	0	$4 * \log 2 = 1.2$	0
Web	9	4	0	0
Server	2	3	0	0

Note:

- As this is just an illustration, the tf values above are just counts and not normalized. Since the document sizes can vary, we may need to normalize in some applications
- This illustration uses log with base 10
- We have not used smoothing approach. If a query term is not found in a document, it is excluded from computing tfidf score.
- Tfidf = 0 for the terms web and server because both documents contain these terms and hence $\text{idf} = \log 1$, which is zero. Hence $\text{tf} * \text{idf} = 0$

As per our algorithm, $\text{score}(q, d1) = 1.2$ and $\text{score}(q, d2) = 0$

Hence the output is: $\text{ranks} = [(d1, 1.2), (d2, 0)]$

Text Normalization

Text Normalization – Wiki Definition

Text normalization

Text normalization is the process of transforming **text** into a single canonical form that it might not have had before. **Normalizing text** before storing or processing it allows for separation of concerns, since input is guaranteed to be consistent before operations are performed on it.

Text normalization - Wikipedia, the free encyclopedia

en.wikipedia.org/wiki/Text_normalization ▼

Text Normalization

- The goal of text normalization is to transform the text in to a form that makes it suitable for further processing.
- The kind of processing that we might do on the normalized text depends on the goals of the application.
- Hence it is imperative that there may not be one single best way to normalize the text.
- Natural language text generated in social media adds to the challenges of NLP. Social media texts may contain:
 - Misspelt words
 - Peter Norvig reports an accuracy of 98.9% for his spell correction program that computes edit distance of 2 on a standard text corpus (<http://norvig.com/spell-correct.html>). But some of the tweets in social media may have words that can not be corrected with an edit distance of 2. eg, “vL” corresponds to “we will” in some tweets!
 - New words and terminologies
 - URLs, emoticons, hashtags, @names etc as part of the text
 - Named entities that contain non alphabet symbols as a part of the name: eg, Yahoo!
 - Sentence and word boundaries not well marked
 - For example, tokenizing words for “vL b” yields 2 word tokens while “we will be” yields 3 tokens

Text tokenization and normalization: Motivating example

Consider 2 problems:

- Suppose we need to pick the tweets from the list below that have distinct information (as against having same information expressed in different forms), which ones we would pick?
- Suppose we need to draw a tag cloud of terms from the tweets, which words are to be shown?
- Key questions:
 - What is the contribution of twitter ids, hash tags, URLs etc?
 - Which words contribute most? Least?
 - What kind of text processing may bring best results?

Sample Tweets (time stamp: 8 Aug 2014, 5:20 pm IST):

1. RT @IndianCricNews: Pujara was dismissed for a duck for the first time in his Test career. 1st duck in FC cricket since November 2008 <http://t.co/4FFMDOcJBV>
2. 4th Test Match' England Vs India' Eng 160-5 over 48 Root 9* M Ali 10* Lead by 8 runs' *GEO PAKISTAN*
3. RT @CricketAus: If you're playing at home, #EngvInd day two is starting. Can India fight back? Follow LIVE <http://t.co/4FFMDOcJBV>
4. Virat Kohli going through worst patch of his career - Zee News <http://t.co/MHUII6zi6a>
5. India vs England Live Score: 4th Test, Day 2 - IBNLive - IBNLiveIndia vs England Live Score: 4th Test, Day 2IBNLiv... <http://t.co/ZIUTD5Y94j>
6. England v India: Fourth Test, Old Trafford, day twoLive - BBC Sport <http://t.co/OHOEiRj9QX>
7. RT @cricbuzz: .@ECB_cricket move into the lead now! Follow here: <http://t.co/GdIOLvNgmb> #EngvInd

Text normalization

- Text normalization is required for most NLP applications
- Consider the problem of generating the index of a corpus of text, where the index contains the word and the locations in the text where the word is found.
- This involves:
 - Splitting the text in to sentences
 - Word tokenization that yields (word, location of the word)
- Word types and tokens
 - Similar to the token types and token values that we encounter during the lexical analysis of programming languages

Lemmatisation and Stemming (ref: Wikipedia definition)

- **Lemmatisation** is the process of grouping together the different inflected forms of a word so they can be analysed as a single item
 - Lemmatization is sensitive to POS: E.g: Meeting (noun), meeting (verb)
- **Stemming:** In [linguistic morphology](#) and [information retrieval](#), **stemming** is the process for reducing inflected (or sometimes derived) words to their [stem](#), base or [root](#) form—generally a written word form.

Note: We will revisit this topic and discuss Porter's algorithm for stemming in a later class

```
>>> p.stem('happier')
'happier'
>>> p.stem('happiest')
'happiest'
>>> p.stem('better')
'better'
>>> p.stem('going')
'go'
>>> p.stem('walking')
'walk'
>>> p.stem('loci')
'loci'
>>> p.stem('foci')
'foci'
>>> print(lemmatizer.lemmatize('running', pos['noun']))
running
>>> print(lemmatizer.lemmatize('running', pos['verb']))
Run
>>> print(lemmatizer.lemmatize('holding', pos['verb']))
hold
```

Stemming and Lemmatization Differences

- Both lemmatization and stemming attempt to bring a canonical form for a set of related word forms.
- Lemmatization takes the part of speech in to consideration. For example, the term 'meeting' may either be returned as 'meeting' or as 'meet' depending on the part of speech.
- Lemmatization often uses a tagged vocabulary (such as Wordnet) and can perform more sophisticated normalization. E.g. transforming mice to mouse or foci to focus.
- Stemming implementations, such as the Porter's stemmer, use heuristics that truncates or transforms the end letters of the words with the goal of producing a normalized form. Since this is algorithm based, there is no requirement of a vocabulary.
- Some stemming implementations may combine a vocabulary along with the algorithm. Such an approach for example convert 'cars' to 'automobile' or even 'Honda City', 'Mercedes Benz' to a common word 'automobile'
- A stem produced by typical stemmers may not be a word that is part of a language vocabulary but lemmatizer transform the given word forms to a valid lemma.

Vector Space Representation of Sentences

Bag of words and Vector Space Model

9.1 Broad to Rahane, no run, touch short, outside off and Rahane fiddles at it, stuck in the crease

9.2 Broad to Rahane, no run, snorting bouncer from Broad, flying through at throat height, Rahane limbos for his life underneath it

- Bag of words is a representation of a document as an unordered set of words with the respective frequency
- This approach of modelling a document by the list of words it contains, without regard to the order, is adequate for several applications (such as those that use Naïve Bayes Classifier or query matching) but is a severe limitation for certain other applications.
- Let us model the above 2 statements in the example as a bag of words

Vocabulary size: $|V| = |V1 \cup V2|$

We have 29 unique words – hence 29 dimensional vector space

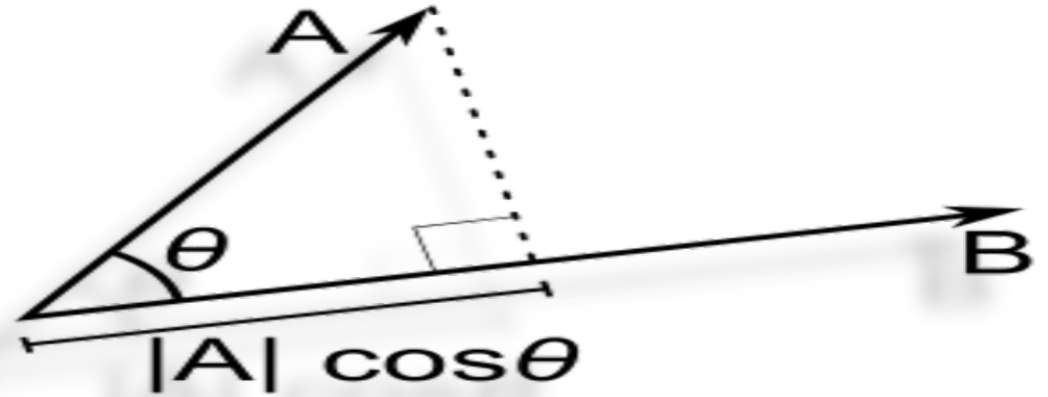
a = (1,1,2,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0)

b = (2,1,2,1,1,0,0,0,0,0,0,1,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1)

Word	Count 1	Count 2	Word	Count 1	Count 2
Broad	1	2	snorting	0	1
to	1	1	bouncer	0	1
Rahane	2	2	from	0	1
no	1	1	flying	0	1
run	1	1	through	0	1
touch	1	0	throat	0	1
Short	1	0	height	0	1
Outside	1	0	limbos	0	1
Off	1	0	for	0	1
And	1	0	his	0	1
fiddles	1	0	life	0	1
At	1	1	undernea	0	1
It	1	0	th		
Stuck	1	0			
In	1	0			
The	1	0			
crease	1	0			

Cosine similarity

- We can map each document in a corpus to a n-dimensional vector, where n is the size of the vocabulary.
 - Here, we represent each unique word as a dimension and the magnitude along this dimension is the count of that word in the document.
- Given such vectors a, b, ..., we can compute the vector dot product and cosine of the angle between them.
- The angle is a measure of alignment between 2 vectors and hence similarity.
- An example of its use in information retrieval is to: Vectorize both the query string and the documents and find similarity(q, di) for all i from 1 to n.



Cosine Similarity – handwritten notes

Vector dot product:

$$\vec{a} \cdot \vec{b} = |\vec{a}| |\vec{b}| \cos \theta$$

$$\cos \theta = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| |\vec{b}|}$$

$$\text{Let: } \vec{a} = a_1 \vec{i} + a_2 \vec{j} + a_3 \vec{k}$$

$$\vec{b} = b_1 \vec{i} + b_2 \vec{j} + b_3 \vec{k}$$

$$\begin{aligned} \vec{a} \cdot \vec{b} &= a_1 b_1 + a_2 b_2 + a_3 b_3 \\ &= \sum_{i=1}^3 a_i b_i \end{aligned}$$

If the 2 vectors are **orthogonal**, the dot product is 0 and hence they are **dissimilar**.

If the 2 vectors are fully **aligned** (theta = 0), cosine is 1 and hence the documents are **similar** (not necessarily identical)

Linear Algebra helps us to perform computations on vectors.

$$a = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \quad b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$a \cdot b = a^T b$$

$$= [a_1 \ a_2 \ a_3] \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$= a_1 b_1 + a_2 b_2 + a_3 b_3 = \sum_{i=1}^3 a_i b_i$$

In general, for n dimensions:

$$\vec{a} \cdot \vec{b} = \sum_{i=1}^n a_i b_i$$

$$\begin{aligned} |\vec{a}| &= \sqrt{a_1^2 + a_2^2 + \dots + a_n^2} \\ |\vec{b}| &= \sqrt{b_1^2 + b_2^2 + \dots + b_n^2} \end{aligned}$$

Emerging

Word Representation