Stockholm University

Basic Natural Language Processing I

Martin Duneld

Department of Computer and Systems Sciences (DSV)

xmartin@dsv.su.se

Purpose of this Lecture

Stockholm University

- Learn the *whats*, *whys* and *hows* of basic NLP

- **What**
  - Common NLP tasks, with a focus on pre-processing text
- **Why**
  - To the computer a “string is a string is a string”
- **How**
  - Examples given in Python using **NLTK** and **spaCy**

Statistics in Corpus Linguistics

Stockholm University

- Today, handling natural language in AI applications usually entails some form of statistical language modelling

- Important terms in linguistic statistics
  - Corpus
  - Types
  - Tokens
  - *n*-grams

- In NLP, a **corpus** is a body of natural language text, often consisting of many documents/texts

Types, Tokens and *n*-grams

Stockholm University

- A **type** is a linguistic unit, this unit can be a:
  - Letter (b, ä, ł, etc.)
  - Word (dog) or punctuation (?)
  - Part-of-Speech (noun, verb, etc.)
  - Some other annotation class, e.g. named entity class (country, body_part, etc.)

- A **token** is an **occurrence** of a type
- Example: "rose is a rose is a rose is a rose"
  - 3 *word* types, but 10 *word* tokens

- The set of *types* is often called our **lexicon**

Types and Tokens in Python

Stockholm University

- Counting tokens:

```
sentence = ["rose","is","a","rose","is","a","rose","is","a","rose"]
print(str(len(sentence)) + " tokens")
```

```
10 tokens
```

- Counting types:

```
types = set(sentence)
print(str(len(types)) + " types")
```

```
3 types
```

***n*-grams**

- ***n*-grams** are continuous sequences *tokens*

- In technical terms, they can be defined as the neighbouring sequences of linguistic items in a text

- Why are they called ***n***-grams?
  - ***n*** is a variable that can have positive integer values [1, 2, 3, …, *n*], and basically refers to any multiple

Types of *n*-grams

Stockholm University

n	Term
1	Unigram
2	Bigram
3	Trigram
n	*n*-gram

- Different *n*-grams are suitable for different applications
- Try different *n*-grams in your models to determine which works best for your specific text analysis
- For instance, research has shown that *trigrams* and *4-grams* work best in the case of *spam filtering*

Example: "I live in Sweden"

Stockholm University

Type of *n*-gram	Generated *n*-grams
Unigram	["I", "live", "in", "Sweden"]
Bigram	["I live", "live in", "in Sweden"]
Trigram	["I live in", "live in Sweden"]

- For some applications we might want to:
  - Include punctuation (., !, ?, etc.)
  - Include input *beginning* and *end symbols*, e.g. marking sentence borders
- Also note that the higher *n* we choose, the fewer observations we will make in our data
  - May lead to data sparseness!

Basic NLP processing

- Tokenisation
- Term normalisation
  - Case, orthography, morphology, etc
- Morphology
  - Stemming
  - Lemmatisation
- Tagging
  - Specifically part-of-speech tagging
- Stop word removal

The NLP Pipeline

Stockholm University

- The first thing you need to do in (almost) any NLP project is text preprocessing
- Preprocessing input text simply means putting the data into a predictable and analysable form
- This preprocessing is often carried out in discrete steps in a pipeline architecture
  - The output of previous steps act as input to later steps

Tokenisation

Stockholm University

- Tokenisation is often the first step in any NLP pipeline
- A tokeniser breaks unstructured data such as natural language text into chunks of information (called tokens) that can be considered as discrete linguistic elements
- Tokenisation can split an input string into e.g. sentences, words, subwords or characters

Why Tokenisation?

Stockholm University

- Tokenisation turns an unstructured string of characters (text document) into a numerical data structure suitable for machine learning
- The token occurrences in a document can be used directly as a vector representing that document
- Different types of tokens can be used as features in machine learning

Equal for all Languages?

- Tokenisation is often non-trivial
- What is a word
  - in the English language?
  - in Chinese?
- However, for many Germanic languages **a small set of regular expressions** and a list of the most **common abbreviations** usually suffices

White Space Tokenisation

Stockholm University

- The simplest way to tokenise text is to use whitespace within a string as the "delimiter" of words
- This can be accomplished with Python's **split** function, which is available on all string object instances

```
sentence = "I am working at DSV. Working at DSV is fun."
sentence.split()
```

```
['I', 'am', 'working', 'at', 'DSV.', 'Working', 'at', 'DSV', 'is', 'fun.']
```

- Notice something in the tokenised string?

Different Tools for Tokenisation

Stockholm University

- **NLTK** is the grand old dad of Natural Language Toolkits for Python and provides several different tokenisers
- We start by importing a few:

```
import nltk
nltk.download('punkt')
from nltk.tokenize import (sent_tokenize,
                           word_tokenize,
                           wordpunct_tokenize,
                           TreebankWordTokenizer,
                           TweetTokenizer,
                           MWETokenizer)
```

- Note: In order to use these we must also download the **Punkt** model for handling punctuation

Basic Tokenisation with NLTK

Stockholm University

- Example of sentence tokenisation:

```
sent_tokenize('My name is Martin. I teach NLP.')
```

```
['My name is Martin.', 'I teach NLP.']
```

- Example of word tokenisation:

```
word_tokenize('My name is Martin. I teach NLP.')
```

```
['My', 'name', 'is', 'Martin', '.', 'I', 'teach', 'NLP', '.']
```

NLTK Punctuation-based Tokeniser

Stockholm University

- This tokeniser splits the input string into words based on whitespaces and punctuations

- A string tokenised with *word_tokenize*:

```
word_tokenize('Learning #NLP with M. Duneld.')
```

```
['Learning', '#', 'NLP', 'with', 'M.', 'Duneld', '.']
```

- Same string tokenised with *wordpunct_tokenize*:

```
wordpunct_tokenize('Learning #NLP with M. Duneld.')
```

```
['Learning', '#', 'NLP', 'with', 'M', '.', 'Duneld', '.']
```

NLTK Treebank Word Tokeniser

Stockholm University

- This tokeniser incorporates a variety of common rules for English word tokenisation
  - It separates phrase-terminating punctuation (?!.;,) from adjacent tokens and retains decimal numbers as a single token
  - It also contains rules for English contractions

```
tbwt = TreebankWordTokenizer()
print(tbwt.tokenize("Don't learn #NLP with M. Duneld."))
```

```
['Do', "n't", 'learn', '#', 'NLP', 'with', 'M.', 'Duneld', '.']
```

- Same sentence tokenised with wordpunct_tokenize

```
print(wordpunct_tokenize("Don't learn #NLP with M. Duneld."))
```

```
['Don', "'", 't', 'learn', '#', 'NLP', 'with', 'M', '.', 'Duneld', '.']
```

- Spot the difference?

NLTK Tweet Tokeniser

Stockholm University

- NLTK also provides a rule-based tokeniser especially for tweets
  - This splits emojis into different tokens, so they can be used for tasks like sentiment analysis

```
tweet= "Don't get bitcoin advice on Twitter 😅👌"
tweet_tokenizer= TweetTokenizer()
print(tweet_tokenizer.tokenize(tweet))
```

```
["Don't", 'get', 'bitcoin', 'advice', 'on', 'Twitter', '😅', '👌']
```

- Same sentence tokenised with wordpunct_tokenize

```
print(tbwt.tokenize(tweet))
```

```
['Do', "n't", 'get', 'bitcoin', 'advice', 'on', 'Twitter', '😅👌']
```

- Spot the difference?

NLTK Multi-Word Expression Tokeniser

Stockholm University

- This tokeniser provides a function *add_mwe()* that allows the user to register multi-word expressions before tokenising the text
  - This then merges multi-word expressions into single tokens

```
sentence = "They didn't move to New York."
mwe_tokenizer = MWETokenizer()
print(mwe_tokenizer.tokenize(word_tokenize(sentence)))
mwe_tokenizer.add_mwe(('New','York'))
print(mwe_tokenizer.tokenize(word_tokenize(sentence)))
```

```
['They', 'did', "n't", 'move', 'to', 'New', 'York', '.']
['They', 'did', "n't", 'move', 'to', 'New_York', '.']
```

Choosing Your Tokeniser

Stockholm University

- Always have *your* specific application in mind
- Since the tokeniser often is the first step in an NLP pipeline it will affect *everything* that comes after
- Make sure that your tokeniser gives you the output you need, otherwise:
  - If possible, find a better suited tokeniser
  - Otherwise, add an intermediary step that post-processes the tokeniser output before next step

Counting Token Frequencies (TF)

Stockholm University

```
sentence = "Rose is a rose is a rose is a rose"
wordlist = tbwt.tokenize(sentence)
wordfreq = []
for word in wordlist:
    wordfreq.append(wordlist.count(word))

print("String: " + "'" + sentence+ "'")
print("Tokens: " + str(wordlist))
print("Frequencies: " + str(wordfreq))
print("TF: " + str(sorted(set(zip(wordlist, wordfreq)),reverse=True)))
```

```
String: 'Rose is a rose is a rose is a rose'
Tokens: ['Rose', 'is', 'a', 'rose', 'is', 'a', 'rose', 'is', 'a', 'rose']
Frequencies: [1, 3, 3, 3, 3, 3, 3, 3, 3, 3]
TF: [('rose', 3), ('is', 3), ('a', 3), ('Rose', 1)]
```

- Note rose/Rose above

Matching Strings

Stockholm University

- To a computer program any chunk of natural language text is just a string of characters
- For the computer dog, dogs and digs are all just a character away from one another, but it knows nothing of how they are related in language
- Apart from when doing exact matching (e.g. phrase search) we most often are interested in concepts
  - So we need to impose some order on the text

Term Normalisation

Stockholm University

- Case
  - Dogs make good pets.
  - I love dogs!
- Orthography
  - Match U.S.A. with USA
- Inflection (morphological ”variants”)
  - run, runs, running ...
  - ran !
- Synonyms
  - Semantics will be discussed later in the course

Case Normalisation

Stockholm University

- Crude approach
  - Lower case all tokens
  - Works well in many cases
  - Language independent
  - But: "'Ruby" (person) ≠ "ruby" (gemstone)

- More surgical approach
  - First: tag the text (at least part-of-speech)
  - Then: Lower case all non-name tokens
  - Language dependent
  - Needed to preserve named entities

Orthographical Normalisation

Stockholm University

- Crude approach
  - Remove all punctuation
  - Works well in many cases
  - Language independent

- More surgical approach
  - Use a list of common abbreviations, contractions and acronyms to bring all variants to a common form (Dr.→Dr→doctor)
  - Language dependent

Morphological Normalisation

Stockholm University

- Crude approach
  - Truncation
  - Not technically NLP…
  - Language independent

  - comput* → computation
    → computations
    → computational
    → computer

- Almost exclusively provided in search engines as a user-initiated feature
- Far to crude for most NLP applications!

Morphological Normalisation

Stockholm University

- More surgical approaches
  - Stemming
  - Lemmatisation

- **Stemming**
  - One of the most widely used techniques for text normalisation
  - **Idea:** improve recall by merging words with basically the same meaning
  - Morphological conflation based on **rewrite rules**
  - We want to ignore superficial morphological features, thus merge semantically similar tokens
  - student, study, studying, studious → studi
  - Language dependent

Example of Stemming Rules

Notation taken from StemmingLab

Stockholm University

Rule format: ^ leftcontext | morpheme $ -> replacement ;

#	denotes a comment
^	denotes NOT
|	denotes end of left context
$	denotes end of word
->	denotes transformation
;	denotes end of rule

```
# dogs -> dog & runs -> run
s$->;
ran$->run;
# stems need not be valid words
cycle$->cycl;
# negation: running -> run &
# cycling -> cycl but keeps ping
^p|ing$->;
```

Challenges with Stemming

Stockholm University

- Affix stripping
  - Suffix: morphological changes at the end
  - Prefix: morphological changes at the beginning
  - Infix: morphological changes in the middle

- German example:
  - Frage (eng. *question*)
  - Abfragen (eng. *to query*)
  - Abgefragt (eng. *to have queried*)

Stemming with NLTK

Stockholm University

- NLTK provides several different stemmers
- Let's import three of them:

```
import nltk
from nltk.stem import (PorterStemmer,
                       SnowballStemmer,
                       LancasterStemmer)
```

- The PorterStemmer is one of the very first stemmers (written in 1979, published 1980)
- Snowball is based on Porter, but heavily improved
- Lancaster is, as we will see, one of the most aggressive stemmers out there

Stemming with PorterStemmer

Stockholm University

- We start by getting an instance of the PorterStemmer

```
porter = PorterStemmer()
```

- Now we can try the PorterStemmer

```
print('amaze -> ' + porter.stem('amaze'))
print('amazed -> ' + porter.stem('amazed'))
print('amazing -> ' + porter.stem('amazing'))
print('amazement -> ' + porter.stem('amazement'))
print('amazon -> ' + porter.stem('amazon'))
```

```
amaze -> amaz
amazed -> amaz
amazing -> amaz
amazement -> amaz
amazon -> amazon
```

NLTK Snowball Stemmer

Stockholm University

- Snowball is a more modern stemmer than the Porter stemmer, and also created by Martin Porter
  - Apart from being a better stemmer than the classic PorterStemmer, Snowball also supports other languages than English

- Let's check what languages Snowball currently supports:

```
print(" ".join(SnowballStemmer.languages))
```

```
arabic danish dutch english finnish french german hungarian italian
norwegian porter portuguese romanian russian spanish swedish
```

Stemming with Snowball

Stockholm University

- First we get an instance of the Snowball stemmer

```
snowball = SnowballStemmer('english')
```

- Comparing Porter and Snowball stemming:

```
print("Porter returns '"+porter.stem('fairly')+"'")
print("SnowBall returns '"+snowball.stem('fairly')+"'")
```

```
Porter returns 'fairli'
SnowBall returns 'fair'
```

- As we can see Snowball seems to produce the better stem for the word “fairly”

More Porter and Snowball Examples

Stockholm University

- NLTK provides several different stemmers

```
print("Porter returns '"+porter.stem('generically')+"'")
print("Porter returns '"+porter.stem('generous')+"'")
print("SnowBall returns '"+snowball.stem('generically')+"'")
print("SnowBall returns '"+snowball.stem('generous')+"'")
```

```
Porter returns 'gener'
Porter returns 'gener'
SnowBall returns 'generic'
SnowBall returns 'generous'
```

- Here we see that the Porter stemmer conflates two words we probably want to count as two separate tokens
- We call this over-stemming

NLTK Lancaster Stemmer

Stockholm University

- The Lancaster is one of the most aggressive stemmers out there
- We start by getting an instance of the stemmer

```
lancaster = LancasterStemmer()
```

- We can immediately see that the Lancaster stemmer massively over-stems

```
print("Lancaster returns '"+lancaster.stem('generically')+"'")
print("Lancaster returns '"+lancaster.stem('generous')+"'")
print("Lancaster returns '"+lancaster.stem('gene')+"'")
```

```
Lancaster returns 'gen'
Lancaster returns 'gen'
Lancaster returns 'gen'
```

Choosing Your Stemmer

Stockholm University

- As with tokenisation you must have *your* specific application in mind
  - Does the stemmer conflate the words you want to have conflated, or does it over- or under-stem *with regards to your application*
- However, in most cases the Snowball stemmer will be a good choice
  - Especially since it provides stemmers for more languages than most other stemmers
- Also consider if **lemmatisation** might be a better choice for you

To Stem or not to Stem

Stockholm University

- There is an ongoing discussion in the NLP field wether stemming words really is a good idea given
  - the computing power and more advanced models that we are able to apply in our NLP pipeline today. (e.g. Deep Neural Networks)
  - that all normalisations lead to some degree of information loss

- Stemming may be a good idea for
  - algorithms that may deal poorly with a high number of dimensions
  - for smaller data sets where we do not have enough observations for many tokens, leading to data sparseness

Lemmatisation

- The process of grouping together different **inflected forms** of a word so they can be analysed as a single item (its lemma)
  - Transforms morphological variants to their root form
  - Language dependent
- Requires knowledge of **part-of-speech** (i.e. is it a *noun* or a *verb*), which in turn requires knowledge of grammar (*morphology* and *syntax*)
  - Non-trivial to implement for a new language
- So, first we dive into part-of-speech tagging

Tagging

- Tagging in NLP is the process of annotating tokens with some form of linguistic analysis
  - Part-of-speech: noun, verb etc.
  - Morphology: plural form, past tense etc.
  - Named entities: person, country, medication etc.
  - Antecedent: who/what is a pronoun referring to
  - and much, much more

- In this lecture we will focus on part-of-speech tagging
  - Often shortened as **PoS-tagging**

Part-of-Speech Tagging

Stockholm University

- Back in elementary school you learnt the difference between nouns, verbs, adjectives, and adverbs
- These "word classes" are not just the idle invention of grammarians, but are useful categories for many language processing tasks
- A PoS-tagger processes a sequence of words and attaches a part-of-speech tag to each word
- As with tokenisers and stemmers, there are several choices for PoS-taggers
  - At least for English

Example Tag Set

Stockholm University

Tag	Part of Speech	Examples
CC	Coordinating Conjunction	and
CD	Cardinal Digit	99, 20.4
DT	Determiner	the
JJ	Adjective	ill
NN	Noun, singular	patient
NNP	Proper Noun, singular	Martin, Prednisolon
RB	Adverb	occasionally, silently
VB	Verb, base form	take
VBD	Verb, past tense	took
VBD	Verb, present tense	continue

+ more

PoS-tagging with NLTK

Stockholm University

- First download a tagger model trained for English

```
nltk.download('averaged_perceptron_tagger')
from nltk import pos_tag
```

- Let's try it out!

- We start by tokenising the text

```
text = "The patient took 20.4 mg Prednisolon. Will continue to take for 4 weeks."
tagged_text = word_tokenize(text)
```

- Now we can PoS-tag the text

PoS-tagging with NLTK

Stockholm University

```
text = "The patient took 20.4 mg Prednisolon. Will continue to take for 4 weeks."
tagged_text = word_tokenize(text)
pos_tag(tagged_text)
```

```
[('The', 'DT'),
 ('patient', 'NN'),
 ('took', 'VBD'),
 ('20.4', 'CD'),
 ('mg', 'NN'),
 ('Prednisolon', 'NNP'),
 ('.', '.'),
 ('Will', 'NNP'),
 ('continue', 'VBP'),
 ('to', 'TO'),
 ('take', 'VB'),
 ('for', 'IN'),
 ('4', 'CD'),
 ('weeks', 'NNS'),
 ('.', '.')]
```

- Notice something?

Lemmatisation with NLTK

Stockholm University

- One NLTK lemmatiser is the WordNetLemmatizer
  - We start by downloading the English pre-trained model used by the lemmatiser, importing the lemmatiser and finally instantiating it

```
nltk.download('wordnet')
from nltk import WordNetLemmatizer
wnl = WordNetLemmatizer()
```

- Now we can lemmatise some words

```
print(wnl.lemmatize("cats"))
print(wnl.lemmatize("cacti"))
print(wnl.lemmatize("geese"))
```

```
cat
cactus
goose
```

Part of Speech and Lemmatisation

Stockholm University

- If no PoS is given to WordNetLemmatizer *noun* is assumed by default
  - This will in many instances give us incorrect lemma

- But we can provide the lemmatiser with the correct PoS tag for the word

```
print(wnl.lemmatize("better"))
print(wnl.lemmatize("better", pos="a"))
print(wnl.lemmatize("running"))
print(wnl.lemmatize("running", pos="a"))
print(wnl.lemmatize("running", pos="v"))
```

```
better
good
running
running
run
```

A Simple NLP Pipeline

Stockholm University

- Let's say we want to lemmatise a sentence
- Our pipeline would look like this

  Tokenise → PoS-tag → Lemmatise

- However, the PoS-tagger and the lemmatiser we have use different tag sets
  - NLTK *averaged_perceptron_tagger* is trained on the Wall Street Journal part of Penn Treebank
  - While *WordNetLemmatizer* is trained on WordNet

- These have different tag sets for PoS tags
  - Thus we need to convert between these

Converting Between Tag Sets

Stockholm University

- First we need to download and import WordNet

```
nltk.download('wordnet')
from nltk.corpus import wordnet
```

- Now we can define a function for converting between the two tag sets

```
def get_wordnet_pos(tag):
    if tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
        return wordnet.NOUN
    elif tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN
```

A Simple NLP Pipeline, contd.

Stockholm University

- Back to our pipeline

  Tokenise → PoS-tag → Lemmatise

- As we can see we have three steps
  - Below, Python code and output is shown for each step in the pipeline

```
sentence = "The striped bats are hanging on their feet for best"
words = word_tokenize(sentence)
print("1. " + str(words))
tags = pos_tag(words)
print("2. " + str(tags))
lemmas = " ".join([wnl.lemmatize(word,get_wordnet_pos(tag)) for (word,tag) in tags])
print('3. ' + lemmas)
```

```
1. ['The', 'striped', 'bats', 'are', 'hanging', 'on', 'their', 'feet', 'for', 'best']
2. [('The', 'DT'), ('striped', 'JJ'), ('bats', 'NNS'), ('are', 'VBP'), ('hanging', 'VBG'),
('on', 'IN'), ('their', 'PRP$'), ('feet', 'NNS'), ('for', 'IN'), ('best', 'JJS')]
3. The striped bat be hang on their foot for best
```

Alternatives to NLTK

Stockholm University

- NLTK is not the only NLP toolkit for Python

- Some alternatives are
  - spaCy: industrial strength NLP engine
  - TextBlob: powerful and fast NLP package
  - Gensim: library for topic modelling, document indexing and similarity retrieval with large corpora
  - Pattern by CLiPs: web mining library with many NLP capabilities

- We'll have quick look at **spaCy**

Introducing spaCy

Stockholm University

- spaCy
  - is one of the best text analysis libraries for Python
  - excels at large-scale information extraction tasks
  - is much faster and accurate than NLTK and TextBlob

- Top Features of spaCy
  - Support for **64+ languages**
  - 64 pre-trained pipelines for **19 languages**
  - Linguistically-motivated tokenisation
  - Syntax-driven sentence segmentation
  - Part-of-speech tagging
  - Named entity recognition
  - Labeled dependency parsing
  - Pre-trained word vectors

Analysing Text with spaCy

Stockholm University

- We start with importing spaCy

```
import spacy
```

- Then, we initialise spaCy with the English model
  - keeping only the PoS-tagger needed for lemmatisation

```
nlp = spacy.load('en_core_web_sm', disable=['parser', 'ner'])
```

- spaCy analyses whole texts in one go and returns a document object

```
doc = nlp(sentence)
```

- Let's have look at this document object

Tokenisation, PoS-Tagging and Lemmatisation with spaCy

Stockholm University

- As we have the text analysis collected in the document object, we can just iterate over it and access the analysis - token by token

```
for token in doc:
  print(token.text + "\t" + token.pos_ + "\t" + token.lemma_)
```

```
The     DET     the
striped VERB    stripe
bats    NOUN    bat
are     AUX     be
hanging VERB    hang
on      ADP     on
their   PRON    their
feet    NOUN    foot
for     ADP     for
best    ADJ     good
```

Stemming vs Lemmatisation

Stockholm University

- The word walk is the base form for the word walking, and thus should be matched both with stemming and lemmatisation
- The word better has good as its lemma, but this might be missed by stemming (if we lack rewrite rules for irregular word forms)
- "cycle" and "cycling" can be conflated with stemming, but **may not** with lemmatisation
  - Note: Lemmatisation always stays within the Part of Speech!

Are All Words Equal?

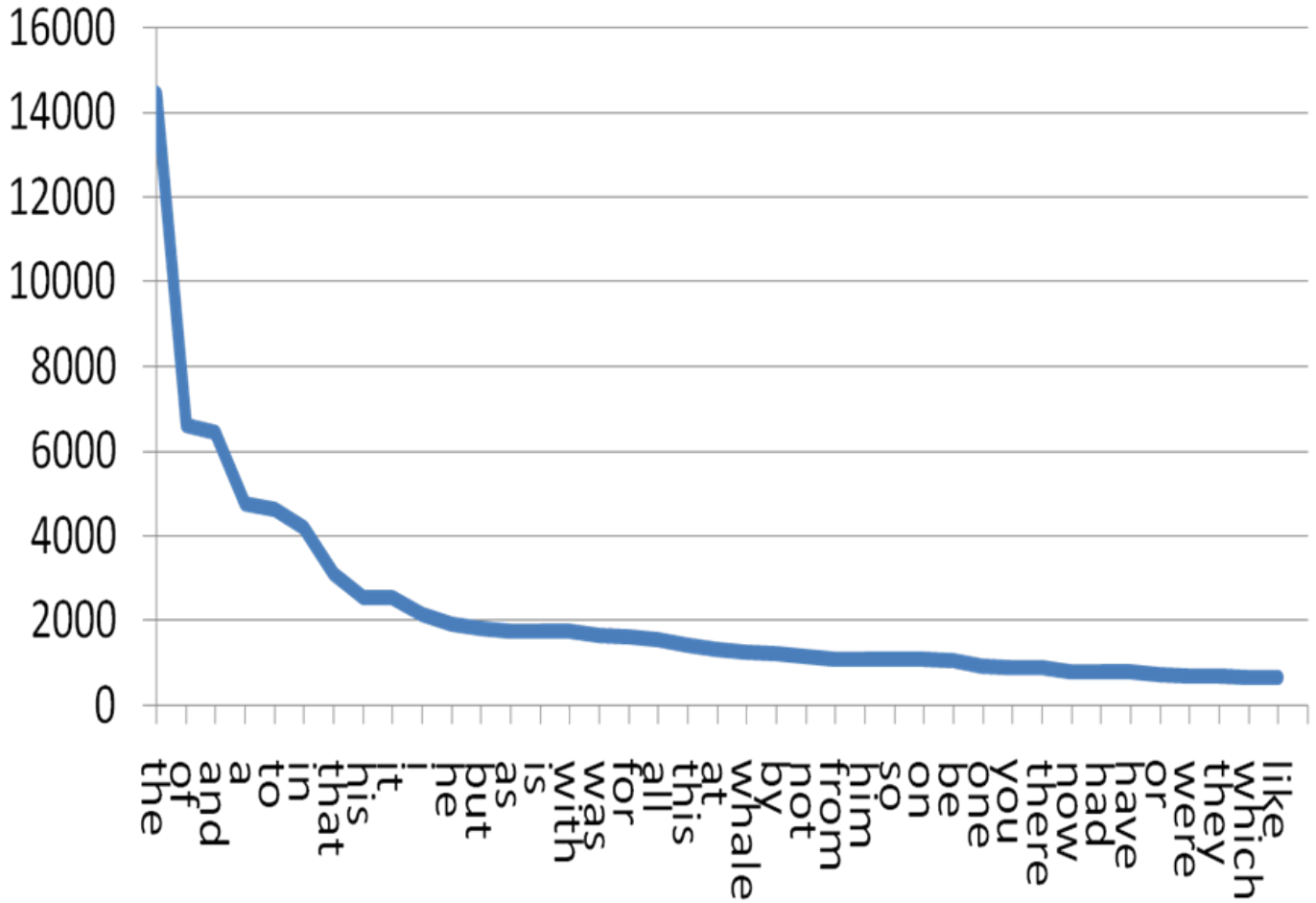
Stockholm University

Table 3
The Bank of English Tagged Word List

Word	Word class tag	Frequency position	Number of occurrences
the	AT	1	11,610,921
be	V	2	6,861,894
of	IN	3	5,359,123
and	CC	4	4,941,292
a	AT	5	4,537,315
in	ININ	6	3,777,696
to	TO	7	3,306,951
abashing	VBG	158,614	4
abandoned	VBD	158,615	4
aban	NN	158,616	4
abalones	NNS	158,617	4
abah	NN	158,618	4

The Long Tail

word frequencies from Moby Dick



Stop Words

Stockholm University

- Words that do not carry much meaning (contribute to the topic of a text) can be removed (or stopped) before indexing a web page or building a language model

- In English: and, the, a, at, or, on, for, etc
  - Typically syntactic markers
  - Typically the most common terms
  - Typically kept in a negative dictionary
  - Often 10–1,000 elements
  - Example:http://ir.dcs.gla.ac.uk/resources/linguistic_utils/stop_words

Inverse Document Frequencies

- **IDF** measures the discriminating power of a term **t**

  $idf_t = N/df_t$

  - **N** is the number of documents in the corpus
  - **df_t** is the number of documents **t** occurs in

- IDF will be low for the most common words
  - Stop words such as "and" are present in almost all documents in any corpus
  - **N/df_t** will give a very low value to such words

Inverse Document Frequencies

- In case of a large corpus, say 100,000,000 documents, the IDF value explodes

- To avoid this we take the *log* of *idf*

  $idf_t = \log(N/(df_t + 1))$

  - **+1** is added to avoid division by zero when looking up words not in the corpus (but in a lexicon or query)

tf-idf

Stockholm University

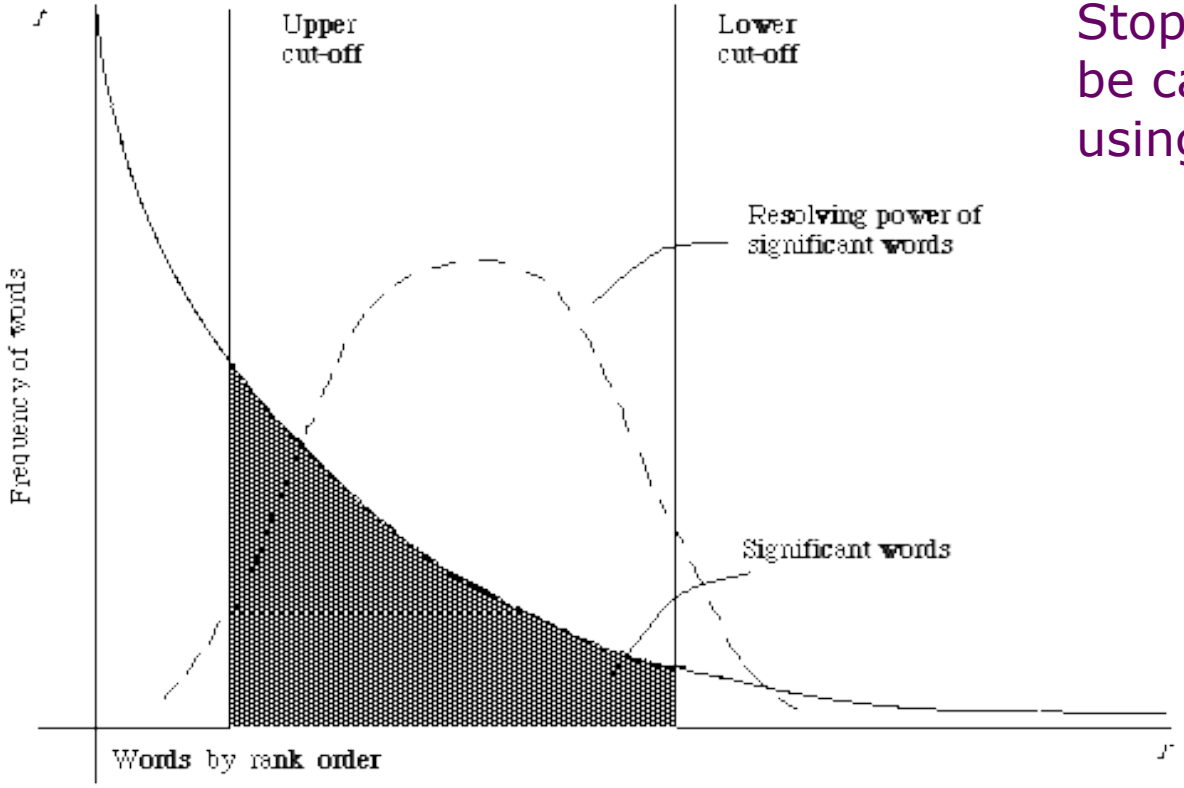
- Measures the informativeness of a term **t**

$$\text{tf-idf}(t,d) = \text{tf}(t,d) * \log(N/(df_t + 1))$$

- Assigns to term **t** a weight in document **d** that is:
  - *highest* when the term **t** occurs many times within a small number of documents (thus lending high discriminating power to those documents)
  - *lower* when the term occurs fewer times in a document, or occurs in many documents (thus offering a less pronounced relevance signal)
  - *lowest* when the term occurs in virtually all documents (such as in the case of stop words)

- Can be used to create domain specific stop word lists

Informativeness Visualised



Stop words can be captured using *tf-idf*

A word's ability to discriminate content is directly related to its frequency

Filtering out Stop Words with spaCy

Stockholm University

- First we get a copy of spaCy's default stop word list
  - And analyse a test sentence to work with

```
stopwords = nlp.Defaults.stop_words
doc = nlp("To be or not to be, that is the question.")
```

- Now we can filter out stop words

```
words = []
for token in doc:
    if token.text.lower() not in stopwords:
        words.append(token.text)
print("Text after filtering out stop words: ", ' '.join(words))
```

```
Text after filtering out stop words:  , question .
```

- We can easily add and remove stop words from the list

```
stopwords.add("question")
stopwords.remove("to")
```

To Stop or not to Stop Words

Stockholm University

- As with stemming, there is an ongoing discussion in the NLP field wether removing stop words really is a good idea (for all the same reasons)

- December 2021 Google subsidiary **DeepMind** announced **Gopher**, a 280-billion-parameter AI NLP model
  - Trained on a 10.5TB corpus called MassiveText, Gopher outperformed the current state-of-the-art on 100 of 124 evaluation tasks

- With such a model and corpus there is not much need for stemming and stop word removal

Take-Home Message:
- Get to Know Your Tools!

Stockholm University

- In this lecture we have seen several ways to accomplish the same task
  - Whether it be tokenisation, stemming, lemmatisation or part-of-speech tagging
- However, different approaches to the same task differ in output
  - Depending on intended purpose
  - Different implementations also make different errors
- Never just grab the first tool you happen to come by
  - Always, always, *always* read up on your tools and play around with them so you know what they produce
  - Unexpected output in one step can lead to grave errors in the input to the next

Links to Libraries in this Lecture

Stockholm University

- NLTK
  - https://www.nltk.org/
  - https://www.nltk.org/book/
- NLTK Corpora
  - http://www.nltk.org/nltk_data/
- spaCy
  - https://spacy.io/
  - https://spacy.io/usage/spacy-101
- spaCy *en_core_web_sm* English model
  - https://spacy.io/models/en

Additional Link Tips

Stockholm University

- TextBlob NLP library
  - https://textblob.readthedocs.io/en/dev/
- Gensim semantic modelling library
  - https://radimrehurek.com/gensim/
- Pattern (CLiPs) web mining library
  - https://www.clips.uantwerpen.be/clips.bak/pages/pattern
- Swedish NLTK Corpora
  - https://github.com/spraakbanken/sb-nltk-tools
- Free dictionaries in Swedish and Danish
  - http://runeberg.org/words/
- PyEnchant spell checking library
  - https://pyenchant.github.io/pyenchant/

Stockholm University

Questions?