3 - Introduction to Natural Language Processing

Giovanni Della Lunga
giovanni.dellalunga@unibo.it

Advanced Machine Learning for Finance

Bologna - April-May, 2022

What is NPL (Natural Language Processing)?

Introduction

- Natural language is what people use to communicate with each other. Unlike formal languages (e.g. programming languages), which are defined by strict rules, natural language is flexible, contextual, and evolving.
- As a result, natural language is not as straightforward for a computer program to process as a script written in a language like Java, Python, or SQL.
- When we talk about Natural Language Processing (or **NLP** for short), we are talking about the ways in which we can use computers to process and interact with human language.

Introduction: What is NLP ?

- Having an insight into what people are talking about can be very valuable to financial traders.
- NLP is being used to track news, reports, comments about possible mergers between companies, everything can then be incorporated into a trading algorithm;
- Banks can determine what customers are saying about a service or product by identifying and extracting information in sources like social media.
- This sentiment analysis can provide a lot of information about customers choices and their decision drivers.

Introduction: What is NLP ?

Text Mining and Text Analysis

- At some level, text analysis is the act of breaking up larger bodies of work into their constituent components - unique vocabulary words, common phrases, syntactical patterns - then applying statistical analysis to them;
- We will soon see that there are many levels to which we can apply our analysis, all of which revolve around a central text dataset: the **corpus**.

Introduction: What is NLP ?

What is a Corpus

- Corpora are collections of related documents that contain natural language.
- A corpus can be large or small, though generally they consist of dozens or even hundreds of gigabytes of data inside of thousands of documents.
- A corpus can be broken down into categories of documents or individual documents.

Introduction: What is NLP ?

What is a Corpus

- Corpora can be **annotated** *meaning that the text or documents are labeled with the correct responses for supervised learning algorithms* , or **unannotated**, making them candidates for topic modeling and document clustering.
- For example, one common type of annotation is the addition of tags, or labels, indicating the word class to which words in a text belong

```
sentence = "My name is Giovanni"
token = nltk.word_tokenize(sentence)
token
```

```
['My', 'name', 'is', 'Giovanni']
```

```
nltk.pos_tag(token)
```

```
[('My', 'PRP$'), ('name', 'NN'), ('is', 'VBZ'), ('Giovanni', 'NNP')]
```

Introduction: What is NLP ?

Domain Specific Corpora

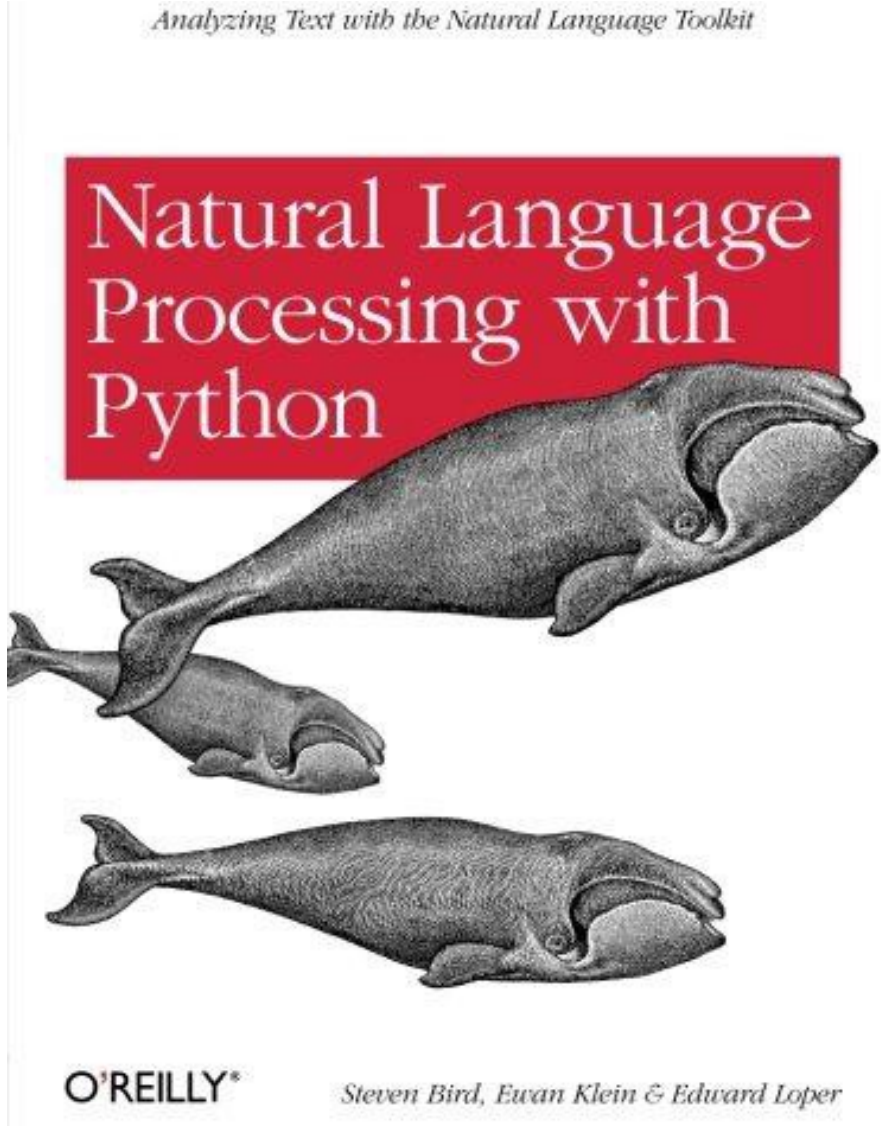
- The best applications tend to use language models trained on domain-specific corpora (collections of related documents containing natural language).
- The reason for this is that language is highly contextual, and words can mean different things in different contexts.
- For example, depending on context, the word **bank** can refer a place where you put your money, the side of a river, the surface of a mine shaft, or the cushion of a pool table!
- With domain-specific corpora, we can reduce ambiguity and prediction space to make results more intelligible.

Subsection 1

The NLTK Package

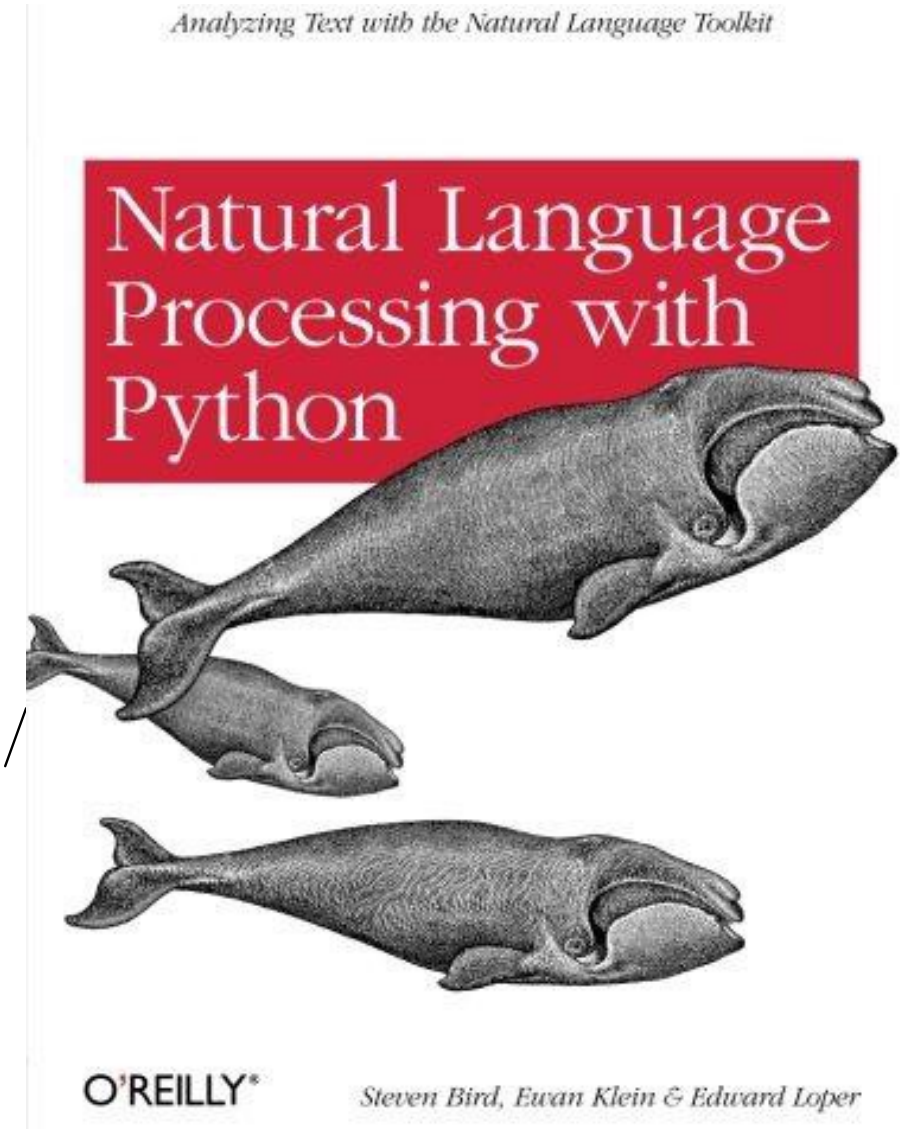
NLTK Package

- The [Natural Language Toolkit](https://www.nltk.org/), or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing (NLP) for English written in the Python programming language.
- It was developed by Steven Bird and Edward Loper in the Department of Computer and Information Science at the University of Pennsylvania.



NLTK Package

- NLTK includes graphical demonstrations and sample data.
- It is accompanied by a book that explains the underlying concepts behind the language processing tasks supported by the toolkit, plus a text book available also on line [here](https://www.nltk.org/book/



NLTK Package

- The latest version is NLTK 3.3. It can be used by students, researchers, and industrialists. It is an Open Source and free library. It is available for Windows, Mac OS, and Linux.
- You can install nltk using *pip installer* if it is not installed in your Python installation. To test the installation:
- Open your Python IDE or the CLI interface (whichever you use normally)
- Type **import nltk** and press enter if no message of missing nltk is shown then nltk is installed on your computer.

NLTK Package

- A **CorpusReader** is a programmatic interface to read, seek, stream, and filter documents, and furthermore to expose data wrangling techniques like encoding and preprocessing for code that requires access to data within a corpus.
- A **CorpusReader** is instantiated by passing a root path to the directory that contains the corpus files, a signature for discovering document names, as well as a file encoding (by default, UTF-8).
- NLTK comes with a variety of corpus readers (66 at the time of this writing) that are specifically designed to access the text corpora and lexical resources that can be downloaded with NLTK.

Corpus Reader

PlaintextCorpusReader. *A reader for corpora that consist of plain-text documents, where paragraphs are assumed to be split using blank lines.*

```
import nltk
from nltk.corpus import PlaintextCorpusReader

corpus_root = '.\corpus\EBA'
corpus_list = PlaintextCorpusReader(corpus_root, '.*', encoding='latin-1')
corpus_list.fileids()
```

```
['Final Guidelines on Accounting for Expected Credit Losses (EBA-GL-2017-06).txt',
 'Final Guidelines on the management of interest rate risk arising from non-trading activities.txt',
 'Final Report on Guidelines on LGD estimates under downturn conditions.txt',
 'Final Report on Guidelines on default definition (EBA-GL-2016-07).txt',
 'Final Report on Guidelines on uniform disclosure of IFRS9 transitional arrangements (EBA-GL-2018-01).txt',
 'Final report on updated GL Funding Plans (EBA 9.12.2019).txt',
 'Final report on updated GL Funding Plans_(EBA-GL-2019-05)_09122019.txt']
```

Corpus Reader

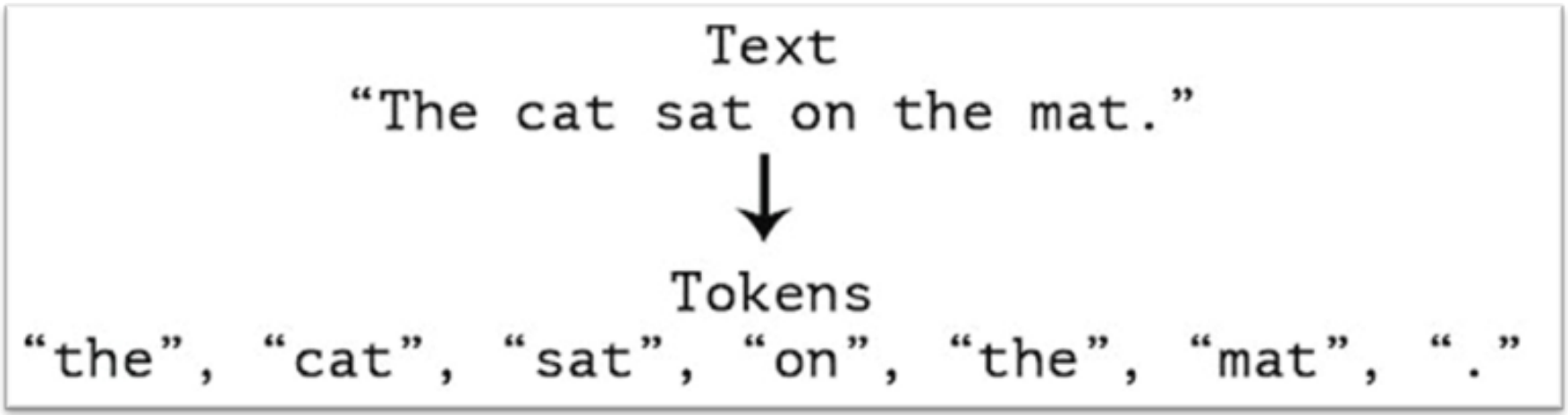
- **TaggedCorpusReader**. *A reader for simple part-of-speech tagged corpora, where sentences are on their own line and tokens are delimited with their tag.*
- **BracketParseCorpusReader**. *A reader for corpora that consist of parenthesis-delineated parse trees.*
- **ChunkedCorpusReader**. *A reader for chunked (and optionally tagged) corpora formatted with parentheses.*
- **TwitterCorpusReader**. *A reader for corpora that consist of tweets that have been serialized into line-delimited JSON.*
- **WordListCorpusReader**. *List of words, one per line. Blank lines are ignored.*
- **XMLCorpusReader**. *A reader for corpora whose documents are XML files.*
- **CategorizedCorpusReader**. *A corpus readers whose documents are organized by category.*

Subsection 2

Text Preprocessing

Tokenization

- Tokenization Is the process of segmenting text into sentences and words.
- In essence, it is the task of cutting a text into pieces called tokens, and at the same time throwing away certain characters, such as punctuation.



Stop Words Removal

- In this process some very common words that appear to provide little or no value to the NLP objective are filtered and excluded from the text to be processed, hence removing widespread and frequent terms that are not informative about the corresponding text.
- Ex.: common language articles, pronouns and prepositions such as **and**, **the** or **to** in English.
- In many situations, stop words can be safely ignored by carrying out a lookup in a pre-defined list of keywords, freeing up database space and improving processing time.
- But **this is not always the case**.

Stop Words Removal

Be aware that:

- There is no universal list of stop words!
- stop words removal can wipe out relevant information and modify the context in a given sentence.
- For example, if we are performing a **sentiment analysis** we might throw our algorithm off track if we remove a stop word like **not**.
- Under these conditions, you might select a minimal stop word list and add additional terms **depending on your specific objective**.

Stemming and Lemmatization

Stemming

- Stemming refers to the process of slicing the end or the beginning of words with the intention of removing affixes (lexical additions to the root of the word).
- Affixes that are attached at the beginning of the word are called **prefixes** (e.g. **astro** in the word **astrobiology**) and the ones attached at the end of the word are called **suffixes** (e.g. **ful** in the word **helpful**).

Stemming and Lemmatization

Lemmatization

- Has the objective of reducing a word to its base form and grouping together different forms of the same word.
- For example, verbs in past tense are changed into present (e.g. **went** is changed to **go**) and synonyms are unified (e.g. **best** is changed to **good**), hence standardizing words with similar meaning to their root.
- Although it seems closely related to the stemming process, lemmatization uses a different approach to reach the root forms of words.

Stemming and Lemmatization

Typically a large corpus will contain many words that have a common root – for example: offer, offered and offering. Lemmatisation and stemming both refer to a process of reducing a word to its root. The difference is that stem might not be an actual word whereas, a lemma is an actual word. It's a handy tool if you want to avoid treating different forms of the same word as different words. Let's consider the following example:

- **Stemming**: considered, considering, consider $\rightarrow$ "consid"
- **Lemmatising**: considered, considering, consider $\rightarrow$ "consider"

Stemming and Lemmatization

- Lemmatization resolves words to their dictionary form (known as lemma) for which it requires detailed dictionaries in which the algorithm can look into and link words to their corresponding lemmas.
- For example, the words **running** , **runs** and **ran** are all forms of the word **run** , so **run** is the lemma of all the previous words.

Stemming and Lemmatization

- lemmatization is a much more resource-intensive task than performing a stemming process.
- At the same time, since it requires more knowledge about the language structure than a stemming approach, it **demands more computational power** than setting up or adapting a stemming algorithm.

Example

- Using **chapter-4-1** Notebook
- Par. 4.1-4.4

Subsection 3

Information Extraction

Part-of-Speech (POS) Tagging

- The process of classifying words into their parts of speech and labeling them accordingly is known as **part-of-speech tagging** or **POS-tagging**, or simply **tagging**.
- The part of speech explains how a word is used in a sentence

```
sentence = "My name is Giovanni"
token = nltk.word_tokenize(sentence)
token
```

```
['My', 'name', 'is', 'Giovanni']
```

```
nltk.pos_tag(token)
```

```
[('My', 'PRP$'), ('name', 'NN'), ('is', 'VBZ'), ('Giovanni', 'NNP')]
```

POS Tagging

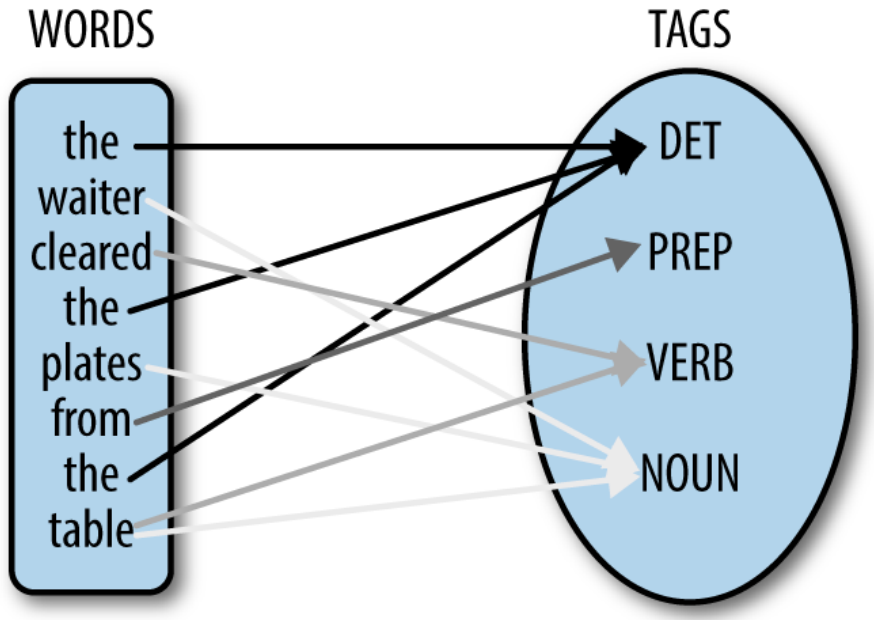
- There are eight main parts of speech:
- nouns,
- pronouns,
- adjectives,
- verbs,
- adverbs,
- prepositions,
- conjunctions and
- interjections.

POS Tagging

- **Noun (N)** : Daniel, London, table, dog, teacher, pen, city
- **Verb (V)** : go, speak, run, eat, play, live, walk, have, like, are, is
- **Adjective (ADJ)** : big, happy, green, young, fun, crazy, three
- **Adverb (ADV)** : slowly, quietly, very, always, never, too, well, tomorrow
- **Preposition (P)** : at, on, in, from, with, near, between, about, under
- **Conjunction (CON)** : and, or, but, because, so, yet, unless, since, if
- **Pronoun (PRO)** : I, you, we, they, he, she, it, me, us, them, him, her, this
- **Interjection (INT)** : Ouch! Wow! Great! Help! Oh! Hey! Hi!

POS Tagging

- The collection of tags used for a particular task is known as a Tagset.
- A part-of-speech tagger, or POS-tagger, processes a sequence of words, and attaches a part of speech tag to each word.



POS Tagging

AT	article
BEZ	the word *is*
IN	preposition
JJ	adjective
JJR	comparative adjective
MD	modal (*may, can, ...*)
MN	singular or mass noun
NNP	singular proper noun
NNS	plural noun
PERIOD	. : ? !
PN	personal pronoun
RB	adverb
RBR	comparative adverb
TO	the word *to*
VB	verb, base form
VBD	verb, past tense
VBG	verb, present participle
VBN	verb, past participle
VBP	verb, non 3d person singular present
VBZ	verb, 3d person singular present
WDT	wh-determiner (*what, which ...*)

Some POS Tag Common Use

- **Partial parsing**: syntactic analysis
- **Information Extraction**: tagging and partial parsing help identify useful terms and relationships between them.
- **Question Answering**: analyzing a query to understand what type of entity the user is looking for and how it is related to other noun phrases mentioned in the question.

Example

- Using **chapter-4-1** Notebook
- Par. 4.5

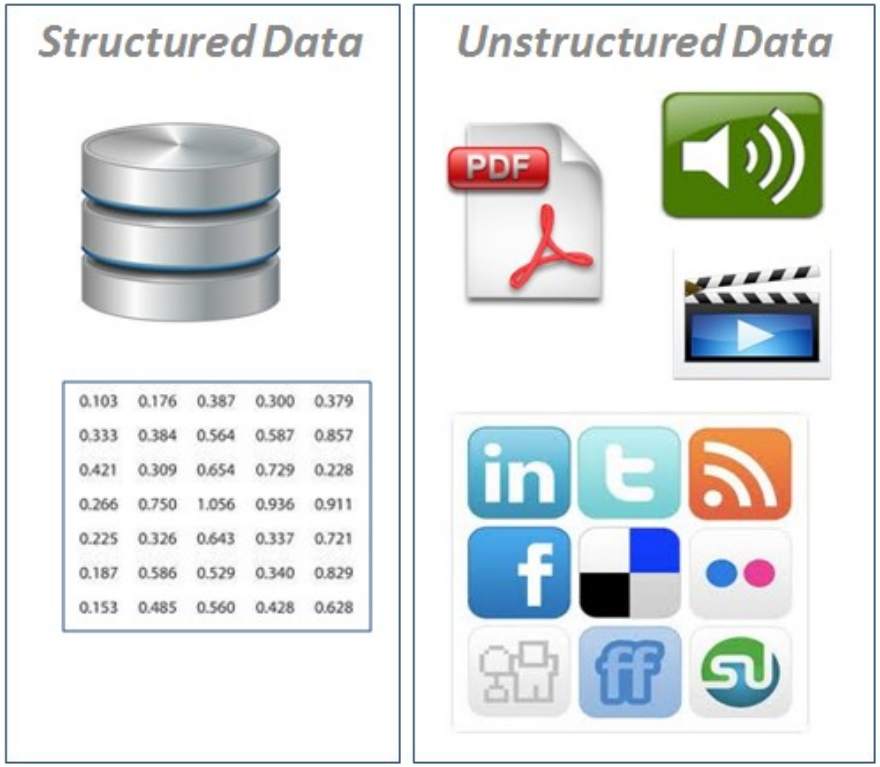
Information Extraction

Unstructured Data Analysis

- Data generated from conversations, declarations or even tweets are examples of unstructured data.
- Unstructured data doesn't fit neatly into the traditional row and column structure of relational databases, and represent the vast majority of data available in the actual world.
- It is messy and hard to manipulate.
- Nevertheless, thanks to the advances in disciplines like machine learning a big revolution is going on regarding this topic.

Information Extraction

- In order to act on unstructured form of information (data), the ML models have to perform one of the crucial processes called Information Extraction(IE).
- Information Extraction is the process of retrieving key information intertwined within the unstructured data.
- In other words, extracting structured data from the unstructured data.



Information Extraction

The goal of this section is to answer the following questions:

- How can we build a system that extracts structured data, such as tables, from unstructured text?
- What are some robust methods for identifying the entities and relationships described in a text?
- Which corpora are appropriate for this work, and how do we use them for training and evaluating our models?

Along the way, we'll apply techniques from the previous sections to the problems of chunking and named-entity recognition.

Information Extraction

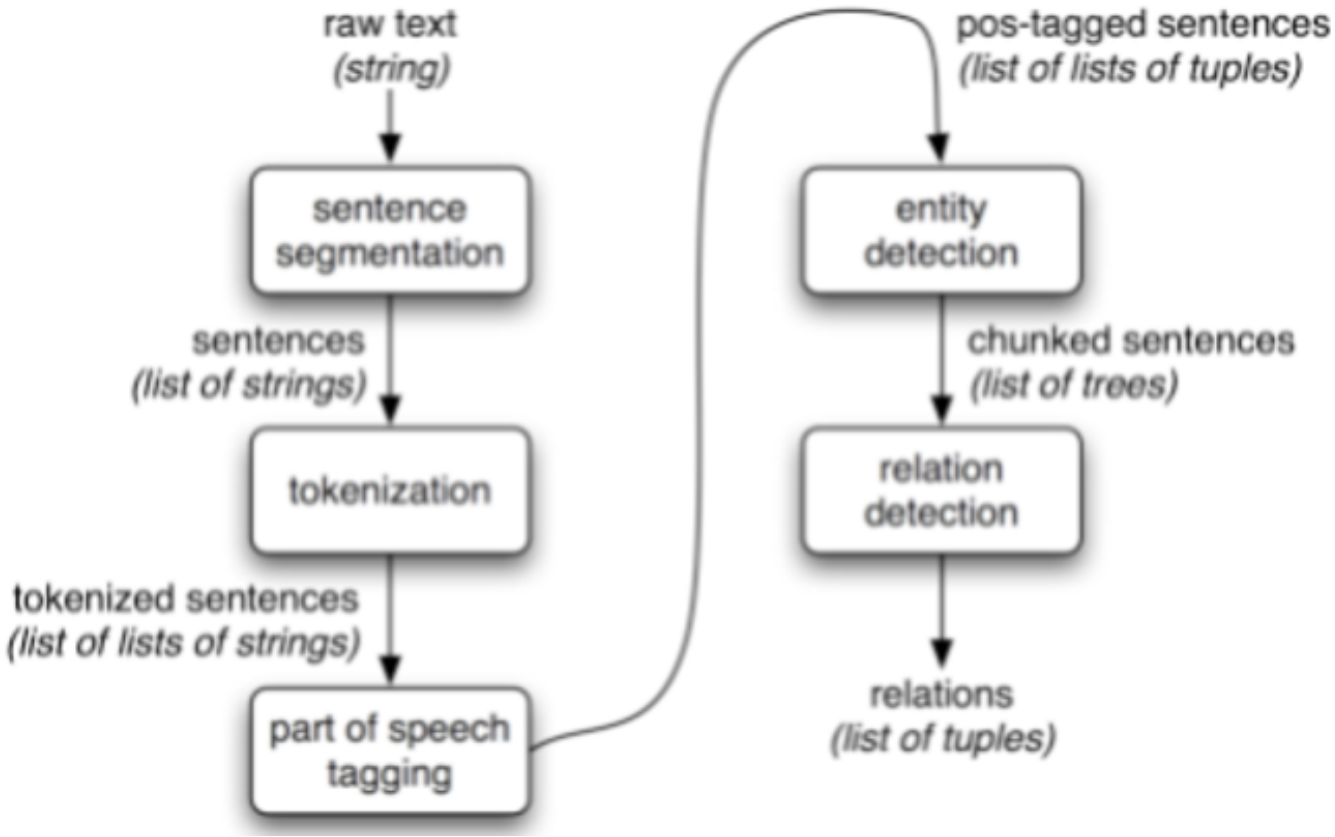
- One approach to this problem involves building a very general representation of meaning.
- In order to simplify the problem at hand, we will take a different approach, deciding in advance that we will only look for very specific kinds of information in text, such as the relation between organizations and locations;
- Rather than trying to use text like to answer a question directly, we first **convert the unstructured data** of natural language sentences into a **structured data**.
- Then we apply the benefits of powerful query tools such as SQL.
- This method of getting meaning from text is called **Information Extraction**.

Information Extraction

- In the following slide we shows the architecture for a simple information extraction system based on the NLTK project (see references).
- It begins by processing a document using several of the procedures already discussed: first, the raw text of the document is split into sentences using a sentence segmenter, and each sentence is further subdivided into words using a tokenizer.
- Next, each sentence is tagged with part-of-speech tags, which will prove very helpful in the next step, named entity detection.
- In this step, we search for mentions of potentially interesting entities in each sentence.
- Finally, we use relation detection to search for likely relations between different entities in the text.

Information Extraction Architecture

source: *Bird S. et al. Natural Language Processing with Python, Chapter 7*



Named Entity Recognition

- A crucial component in IE systems is Named Entity Recognition (NER).
- Named-entity recognition is the problem of identifying and classifying entities into categories such as the names of people, locations, organizations, the expressions of quantities, times, measurements, monetary values, and so on.
- In general terms, entities refer to names of people, organizations (e.g. United Nations, American Airlines), places/cities (Rome, Boston), etc.

Named Entity Recognition

- *The fourth Wells account moving to another agency is the packaged paper-products division of Georgia-Pacific Corp., which arrived at Wells only last fall. Like Hertz and the History Channel, it is also leaving for an Omnicom-owned agency, the BBDO South unit of BBDO Worldwide. BBDO South in Atlanta, which handles corporate advertising for Georgia-Pacific, will assume additional duties for brands like Angel Soft toilet tissue and Sparkle paper towels, said Ken Haldin, a spokesman for Georgia-Pacific in Atlanta.*
- The question is: **which companies are based in Atlanta?**

Taken from Bird S. et al. *Natural Language Processing with Python* O'Reilly (2009)

Extracting Information from Text

ENTITIES: ADVERTISING AGENCY

The fourth Wells account moving to another agency is the packaged paper-products division of Georgia-Pacific Corp., which arrived at Wells only last fall. Like Hertz and the History Channel, it is also leaving for an Omnicom-owned agency, the BBDO South unit of BBDO Worldwide. BBDO South in Atlanta, which handles corporate advertising for Georgia-Pacific, will assume additional duties for brands like Angel Soft toilet tissue and Sparkle paper towels, said Ken Haldin, a spokesman for Georgia-Pacific in Atlanta.

Wells Rich Greene BDDP

Extracting Information from Text

ENTITIES: MANUFACTORYING COMPANIES

ENTITIES: TELEVISION NETWORK

HISTORY

The fourth Wells account moving to another agency is the packaged paper-products division of Georgia-Pacific Corp., which arrived at Wells only last fall. Like Hertz and the History Channel, it is also leaving for an Omnicom-owned agency, the BBDO South unit of BBDO Worldwide. BBDO South in Atlanta, which handles corporate advertising for Georgia-Pacific, will assume additional duties for brands like Angel Soft toilet tissue and Sparkle paper towels, said Ken Haldin, a spokesman for Georgia-Pacific in Atlanta.

ENTITIES: CAR RENTAL COMPANY

Hertz

Extracting Information from Text

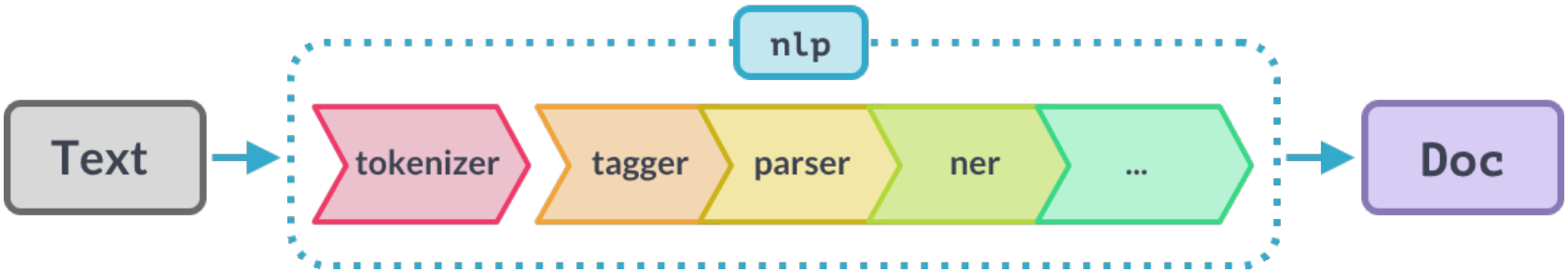
ENTITIES: GEOGRAPHICAL ENTITY

The fourth Wells account moving to another agency is the packaged paper-products division of Georgia-Pacific Corp., which arrived at Wells only last fall. Like Hertz and the History Channel, it is also leaving for an Omnicom-owned agency, the BBDO South unit of BBDO Worldwide. BBDO South in Atlanta, which handles corporate advertising for Georgia-Pacific, will assume additional duties for brands like Angel Soft toilet tissue and Sparkle paper towels, said Ken Haldin, a spokesman for Georgia-Pacific in Atlanta.

RELATION: IN

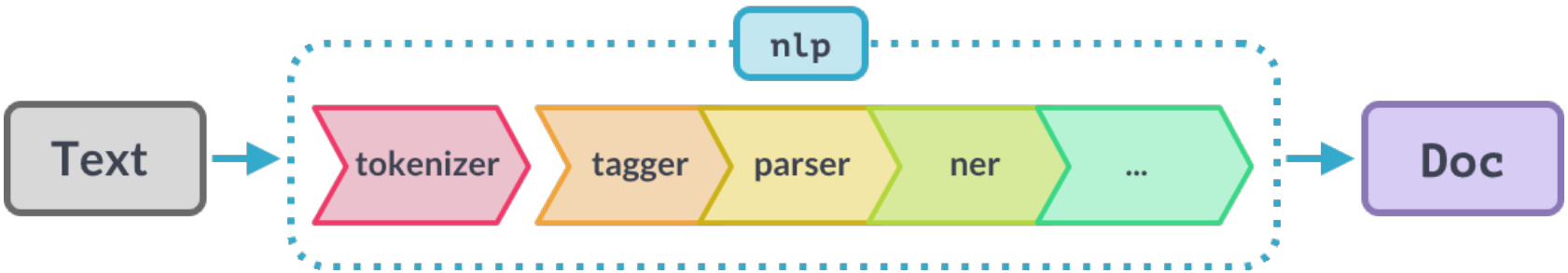
Extracting Information from Text: Using spaCy

- Download and install a trained pipeline (in this case 'en_core_web_sm'), you can load it via spacy.load.
- This will return a Language object containing all components and data needed to process text. We call it 'spacy_nlp'. Calling the nlp object on a string of text will return a processed Doc.
- In particular, When you call 'spacy_nlp' on a text, spaCy first tokenizes the text to produce a Doc object.



Extracting Information from Text: Using spaCy

- The Doc is then processed in several different steps – this is also referred to as the processing pipeline.
- The pipeline used by the trained pipelines typically include a tagger, a lemmatizer, a parser and an entity recognizer. Each pipeline component returns the processed Doc, which is then passed on to the next component.



Extracting Information from Text: Using spaCy

```
spacy_nlp = spacy.load("en_core_web_sm")
document  = spacy_nlp(article)
print('Original article: %s \n' % (article))
for element in document.ents:
    print('Type : %s, Value : %s' % (element.label_, element))

spacy.displacy.render(spacy_nlp(article), style='ent', jupyter=True)
```

The fourth ORDINAL Wells ORG account moving to another agency is the packaged paper-products division of Georgia-Pacific Corp. ORG , which arrived at Wells ORG only last fall. Like Hertz ORG and the History Channel ORG , it is also leaving for an Omnicom ORG -owned agency, the BBDO South ORG unit of BBDO Worldwide ORG . BBDO South ORG in Atlanta GPE , which handles corporate advertising for Georgia-Pacific ORG , will assume additional duties for brands like Angel Soft PERSON toilet tissue and Sparkle paper towels, said Ken Haldin PERSON , a spokesman for Georgia-Pacific ORG in Atlanta GPE .

Extracting Information from Text: Using NLTK with Stanford NER

The Stanford Natural Language Processing Group

people publications research blog software teaching join local

Software > Stanford Named Entity Recognizer (NER)

About | Citation | Getting started | Questions | Mailing lists | Download | Extensions | Models | Online demo | Release history | FAQ

About

Stanford NER is a Java implementation of a Named Entity Recognizer. Named Entity Recognition (NER) labels sequences of words in a text which are the names of things, such as person and company names, or gene and protein names. It comes with well-engineered feature extractors for Named Entity Recognition, and many options for defining feature extractors. Included with the download are good named entity recognizers for English, particularly for the 3 classes (PERSON, ORGANIZATION, LOCATION), and we also make available on this page various other models for different languages and circumstances, including models trained on just the CoNLL 2003 English training data.

Stanford NER is also known as CRFClassifier. The software provides a general implementation of (arbitrary order) linear chain Conditional Random Field (CRF) sequence models. That is, by training your own models on labeled data, you can actually use this code to build sequence models for NER or any other task. (CRF models were pioneered by Lafferty, McCallum, and Pereira (2001); see Sutton and McCallum (2006) or Sutton and McCallum (2010) for more comprehensible introductions.)

The original CRF code is by Jenny Finkel. The feature extractors are by Dan Klein, Christopher Manning, and Jenny Finkel. Much of the documentation and usability is due to Anna Rafferty. More recent code development has been done by various Stanford NLP Group members.

Stanford NER is available for download, **licensed under the GNU General Public License** (v2 or later). Source is included. The package includes components for command-line invocation (look at the shell scripts and batch files included in the download), running as a server (look at `NERServer` in the sources jar file), and a Java API (look at the simple examples in the `NERDemo.java` file included in the download, and then at the javadocs). Stanford NER code is dual licensed (in a similar manner to MySQL, etc.). Open source licensing is under the *full* GPL, which allows many free uses. For distributors of proprietary software, commercial licensing is available. If you don't need a commercial license, but would like to support maintenance of these tools, we welcome gifts.

Example : Reading the Newspaper

- Notebook: **chapter-4-1**
- Libraries: NLTK, Stanford Group NER
- https://nlp.stanford.edu/software/

Comparing Results

- Natural language processing applications are characterized by complex interdependent decisions that require large amounts of prior knowledge.
- In this case, as you can see, the system designed by Stanford did not achieve the same result as spaCy, but it is pure accident; in fact, a lot depends on how well the models have been trained and with how much data.
- For this reason, in case there is a need to perform a task like this, the best thing to do is to use multiple tools and compare the results, in order to find the best one in terms of performance and response.

Text Vectorization

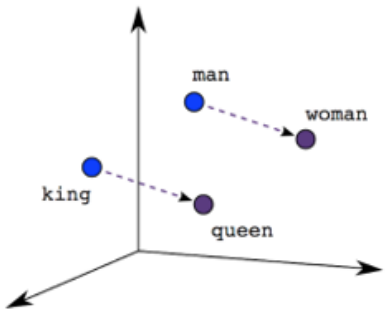
Turning Words into Numbers

Introduction

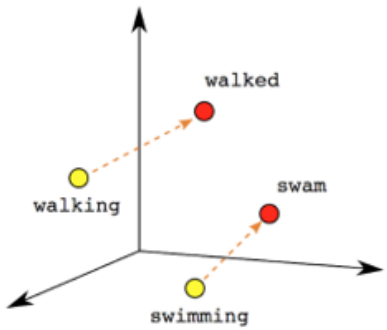
- Machine learning algorithms operate on a numeric feature space, expecting input as a two-dimensional array where rows are instances and columns are features.
- What are the appropriate features for applying a ML model to a text?
- In order to perform machine learning on text, we need to transform our documents into vector representations such that we can apply numeric machine learning.
- This process is called **feature extraction** or more simply, **vectorization**, and is an essential first step toward language-aware analysis.

Introduction

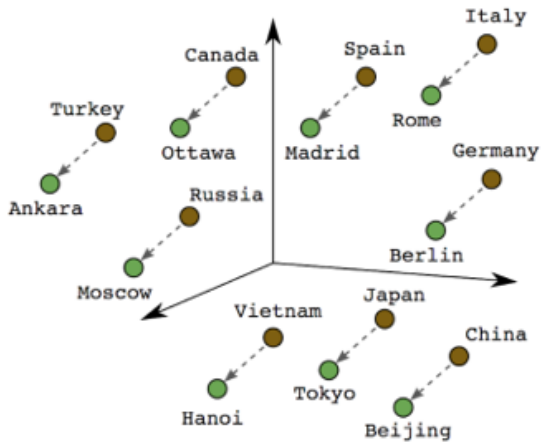
- For this reason, we must now make a critical shift in how we think about language - from a sequence of words to points that occupy a high-dimensional semantic space



Male-Female



Verb Tense



Country-Capital

Introduction

- Points in space can be close together or far apart, tightly clustered or evenly distributed.
- Semantic space is therefore mapped in such a way where documents with similar meanings are closer together and those that are different are farther apart.
- By encoding **similarity** as distance, we can begin to derive the primary components of documents and draw decision boundaries in our semantic space.
- The simplest encoding of semantic space is the bag-of-words model, whose primary insight is that meaning and similarity are encoded in **vocabulary**.

Subsection 1

Bag-of-Words (BOW)

Bag-of-Words

- To vectorize a corpus with a bag-of-words (BOW) approach, **we represent every document from the corpus as a vector whose length is equal to the vocabulary of the corpus**.
- The vocabulary is the set of words without repetition present in the whole corpus

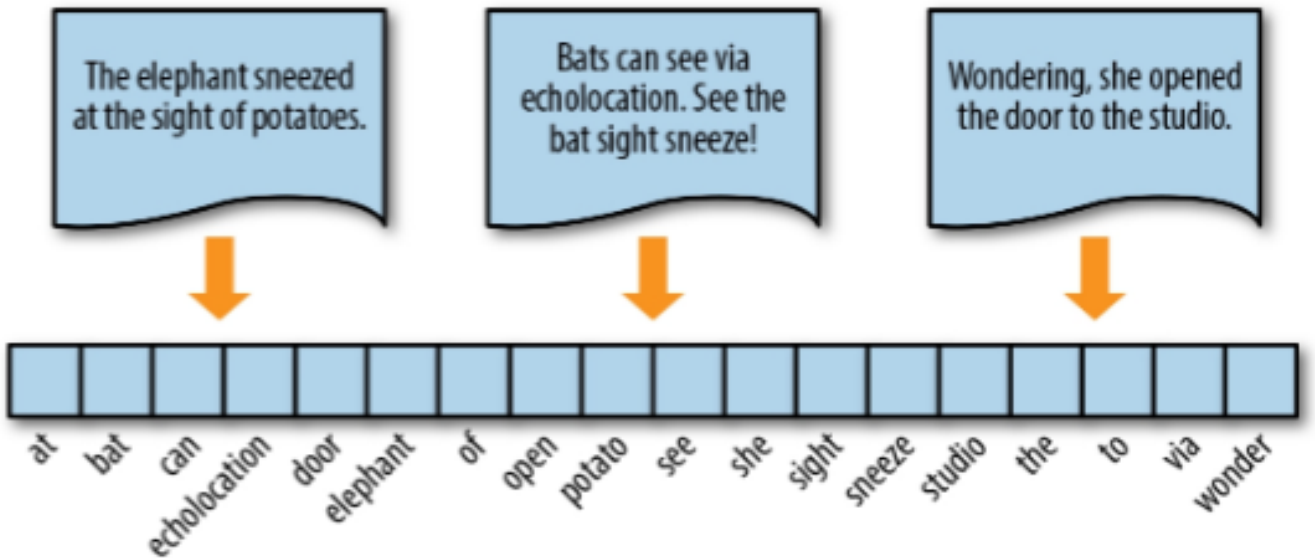


image source: *Bengfort B. et al. Text Analysis with Python*

Bag-of-Words

What should each element in the document vector be?

- We will explore several choices, each of which extends or modifies the base bag-of-words model to describe semantic space.
- We will look at three types of vector encoding - frequency, one-hot, TF-IDF, - and discuss their implementations in Scikit-Learn and NLTK.

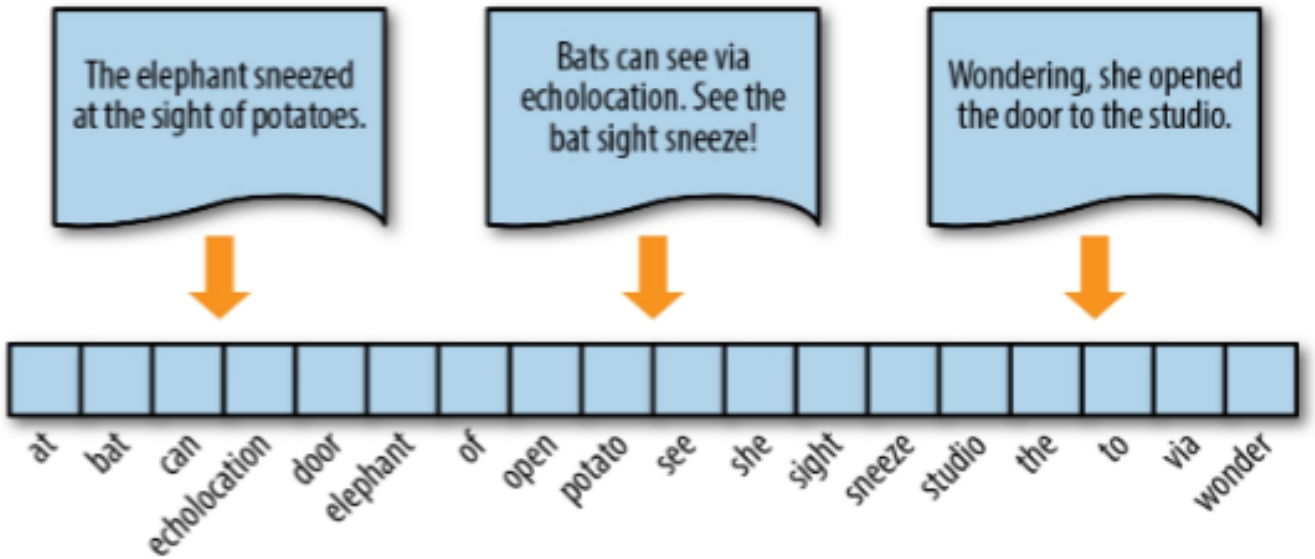


image source: *Bengfort B. et al. Text Analysis with Python*

Bag-of-Words

- **Frequency Vectors**
- The simplest vector encoding model is to simply fill in the vector with **the frequency of each word as it appears in the document**;
- In this encoding scheme each document is represented as the multiset of the tokens that compose it and the value for each word position in the vectr is its count;
- This representation can either be a straight count encoding or a normalized encoding where each word is weighted by the total number of words in the document;

Bag-of-Words

Token Frequency as Vector Encoding

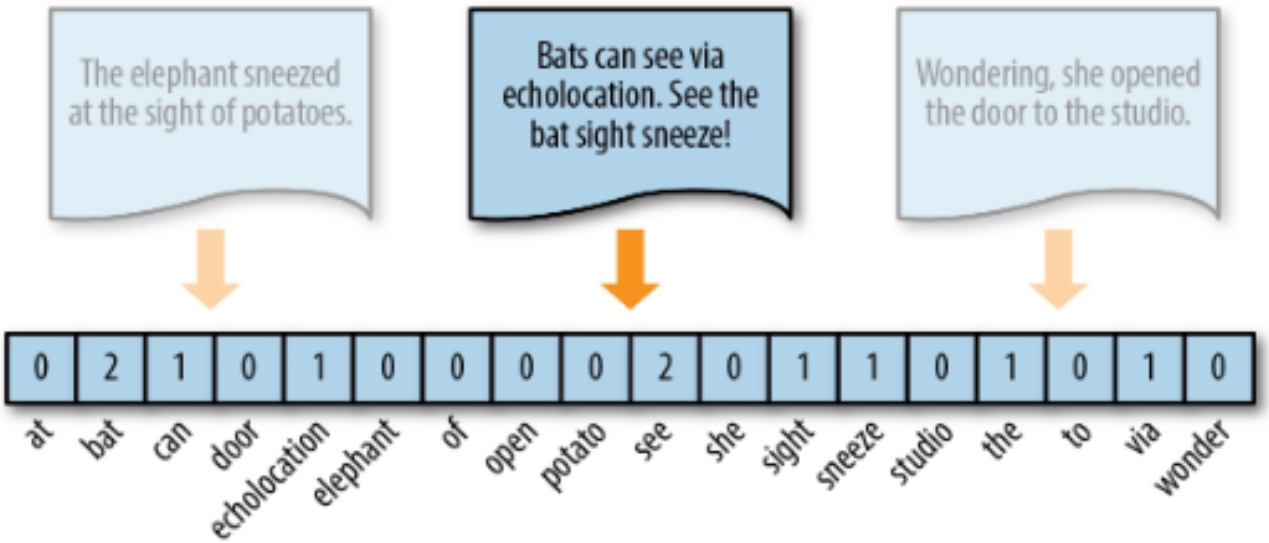


image source: *Bengfort B. et al. Text Analysis with Python*

Text Feature Extraction

- CountVectorizer: turns a collection of text documents into numerical feature vectors.

```
>>> from sklearn.feature_extraction.text import
CountVectorizer
>>> c = CountVectorizer()
```

- **TF**: Just counting the number of words in each document has 1 issue: it will give **more weightage to longer documents than shorter documents**. To avoid this, we can use frequency (**TF - Term Frequencies**) i.e. #count(word) / #Total words, in each document.

Bag-of-Words

Variants of term frequency (tf) weight

weighting scheme	tf weight
binary	$0, 1$
raw count	$f_{t,d}$
term frequency	$f_{t,d} \Big/ \sum_{t' \in d} f_{t',d}$
log normalization	$\log(1 + f_{t,d})$
double normalization 0.5	$0.5 + 0.5 \cdot \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$
double normalization K	$K + (1 - K)\frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$

One-Hot Encoding

One-Hot encoding is a method that produces a boolean vector. A particular vector index is marked as TRUE (1) if the token exist in the document and FALSE (0) if it does not.

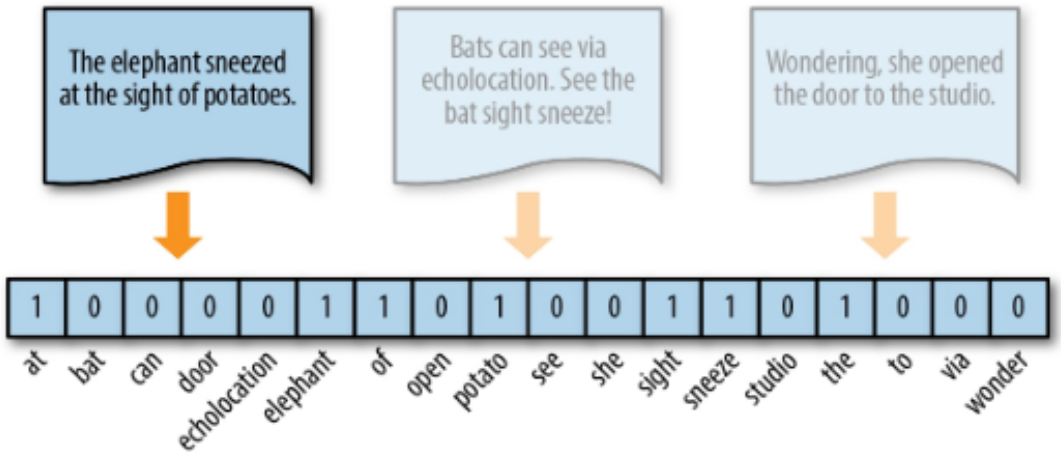


image source: *Bengfort B. et al. Text Analysis with Python*

Term Frequency-Inverse Document Frequency

- The bag-of-words representations that we have explored so far only describe a document in a standalone fashion, **not taking into account the context of the corpus**.
- A better approach would be to consider the **relative frequency or rareness of tokens in the document against their frequency in other documents**.
- The central insight is that **meaning is most likely encoded in the more rare terms from a document**.

Compute Inverse Document Frequency

- The inverse document frequency is a measure of how much information the word provides, that is, whether the term is common or rare across all documents.
- It is the logarithmically scaled inverse fraction of the documents that contain the word, obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient:

$$idf(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|} \tag{1}$$

where **the numerator (N) is the total number of documents** in the corpus and **the denominator is the number of documents where the term t appears**.

Compute Inverse Document Frequency

Variants of inverse document frequency (idf) weight

weighting scheme	idf weight ($n_t = \left\|\{d \in D : t \in d\}\right\|$)
unary	1
inverse document frequency	$\log \frac{N}{n_t} = -\log \frac{n_t}{N}$
inverse document frequency smooth	$\log\left(1 + \frac{N}{n_t}\right)$
inverse document frequency max	$\log\left(\frac{\max_{\{t' \in d\}} n_{t'}}{1 + n_t}\right)$
probabilistic inverse document frequency	$\log \frac{N - n_t}{n_t}$

Term Frequency-Inverse Document Frequency

- TF-IDF, term frequency-inverse document frequency, encoding normalizes the frequency of tokens in a document with respect to the rest of the corpus.
- This encoding approach accentuates terms that are very relevant to a specific instance, as shown in Figure, where the token studio has a higher relevance to this document since it only appears there.

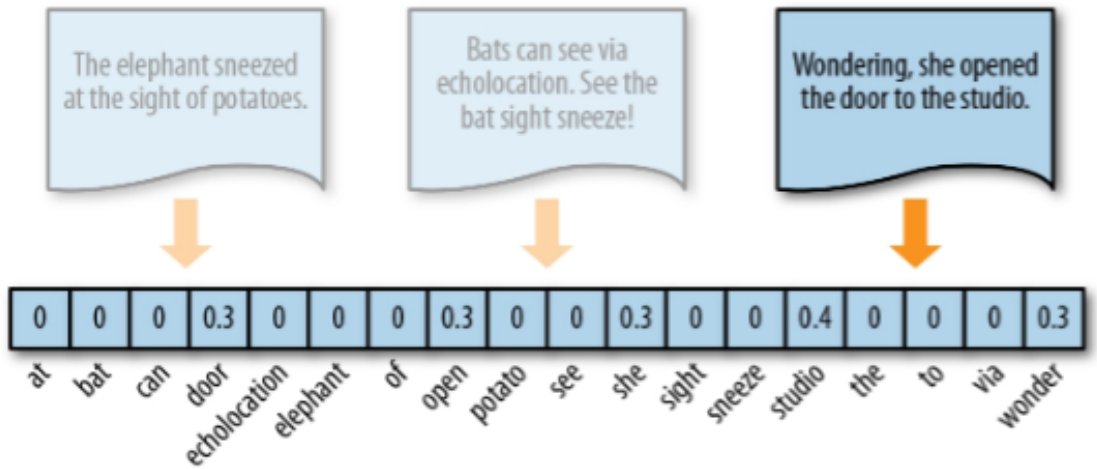


image source: *Bengfort B. et al. Text Analysis with Python*

Compute Term Frequency-Inverse Document Frequency

- Then tf-idf is calculated as:

$$tfidf(t, d, D) = tf(t, d) \cdot idf(t, D) \tag{2}$$

- A high weight in tf-idf is reached by a high term frequency (in the given document) and a low document frequency of the term in the whole collection of documents;
- The weights hence tend to filter out common terms.
- Since the ratio inside the idf's log function is always greater than or equal to 1, the value of idf (and tf-idf) is greater than or equal to 0. As a term appears in more documents, the ratio inside the logarithm approaches 1, bringing the idf and tf-idf closer to 0.

Compute Term Frequency-Inverse Document Frequency

Recommended tf–idf weighting schemes

weighting scheme	document term weight	query term weight
1	$f_{t,d} \cdot \log \frac{N}{n_t}$	$\left(0.5 + 0.5 \frac{f_{t,q}}{\max_t f_{t,q}}\right) \cdot \log \frac{N}{n_t}$
2	$1 + \log f_{t,d}$	$\log(1 + \frac{N}{n_t})$
3	$(1 + \log f_{t,d}) \cdot \log \frac{N}{n_t}$	$(1 + \log f_{t,q}) \cdot \log \frac{N}{n_t}$

Text Feature Extraction

CountVectorizer: turns a collection of text documents into numerical feature vectors.

```
>>> from sklearn.feature_extraction.text import
TfidfTransformer
>>> transformer = TfidfTransformer()
```

As tf-idf is very often used for text features, there is also another class called TfidfVectorizer that combines all the options of CountVectorizer and TfidfTransformer in a single model

```
>>> from sklearn.feature_extraction.text import
TfidfVectorizer
>>> vectorizer = TfidfVectorizer()
```

Document Similarity

- When you have vectorized your text, we can try to define a distance metric such that documents that are closer together in feature space are more similar.
- There are a number of different measures that can be used to determine document similarity;

String Matching	Distance Metrics	Relational Matching	Other Matching
Edit Distance - Levenstein - Smith-Waterman - Affine **Alignment** - Jaro-Winkler - Soft-TFIDF - Monge-Elkan **Phonetic** - Soundex - Translation	- Euclidean - Manhattan - Minkowski **Text Analytics** - Jaccard - TFIDF - Cosine similarity	**Set Based** - Dice - Tanimoto (Jaccard) - Common Neighbors - Adar Weighted **Aggregates** - Average values - Max/Min values - Medians - Frequency (Mode)	- Numeric distance - Boolean equality - Fuzzy matching - Domain specific **Gazettes** - Lexical matching - Named Entities (NER)

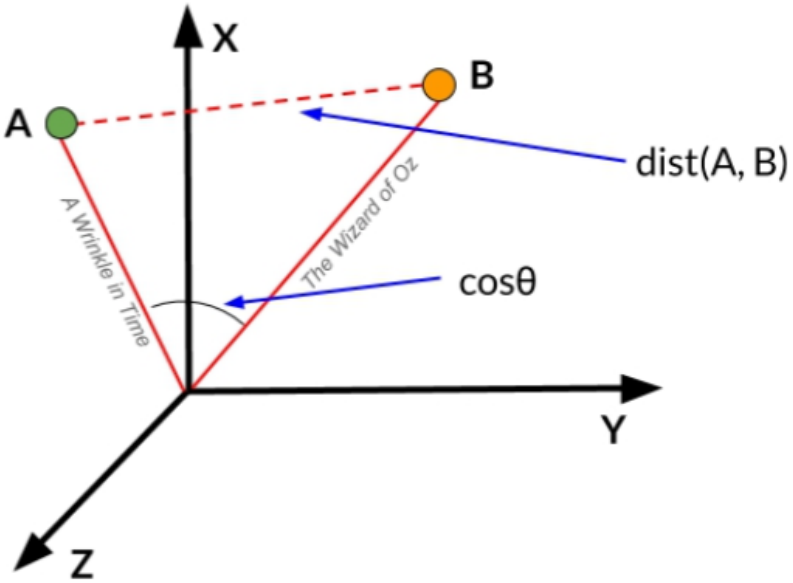
Document Similarity

- Fundamentally, each relies on our ability to imagine documents as points in space, where the relative closeness of any two documents is a measure of their similarity.

String Matching	Distance Metrics	Relational Matching	Other Matching
Edit Distance - Levenstein - Smith-Waterman - Affine **Alignment** - Jaro-Winkler - Soft-TFIDF - Monge-Elkan **Phonetic** - Soundex - Translation	- Euclidean - Manhattan - Minkowski **Text Analytics** - Jaccard - TFIDF - Cosine similarity	**Set Based** - Dice - Tanimoto (Jaccard) - Common Neighbors - Adar Weighted **Aggregates** - Average values - Max/Min values - Medians - Frequency (Mode)	- Numeric distance - Boolean equality - Fuzzy matching - Domain specific **Gazettes** - Lexical matching - Named Entities (NER)

Document Similarity

- We can measure vector similarity with cosine distance, using the cosine of the angle between the two vectors to assess the degree to which they share the same orientation.
- In effect, the more parallel any two vectors are, the more similar the documents will be (regardless of their magnitude).



Document Similarity

- Mathematically, Cosine similarity metric measures the cosine of the angle between two n-dimensional vectors projected in a multi-dimensional space. The Cosine similarity of two documents will range from 0 to 1. If the Cosine similarity score is 1, it means two vectors have the same orientation. The value closer to 0 indicates that the two documents have less similarity.
- The mathematical equation of Cosine similarity between two non-zero vectors is:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \ \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sum_{i=1}^{n} A_i^2 \sum_{i=1}^{n} B_i^2} \tag{3}$$

Example : Bag-of-Words

- Notebook: **chapter-4-2**
- Focus: Vectorization with sklearn, Document Similarity
- Libraries: NLTK, sklearn

Subsection 2

Word Embedding

What is Word Embedding

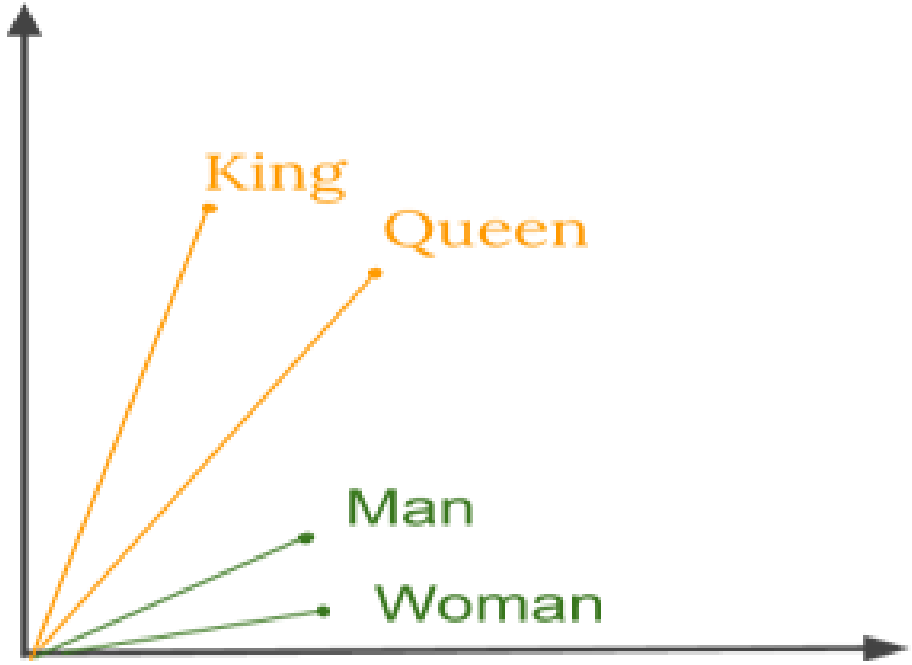
- The different encoding we have discussed so far is arbitrary as **it does not capture any relationship between words**.
- It can be challenging for a model to interpret, for example, a linear classifier learns a single weight for each feature.
- Because there is no relationship between the similarity of any two words and the similarity of their encodings, this feature-weight combination is not meaningful.
- Furthermore, **the dimension of the vector space**, being equal to the number of terms in the vocabulary, **tends to increase considerably** as the size of the corpus increases.

What is Word Embedding

- Relationship between words
- For example, we humans understand the words like king and queen, man and woman, tiger and tigress have a certain type of relation between them but how can a computer figure this out?

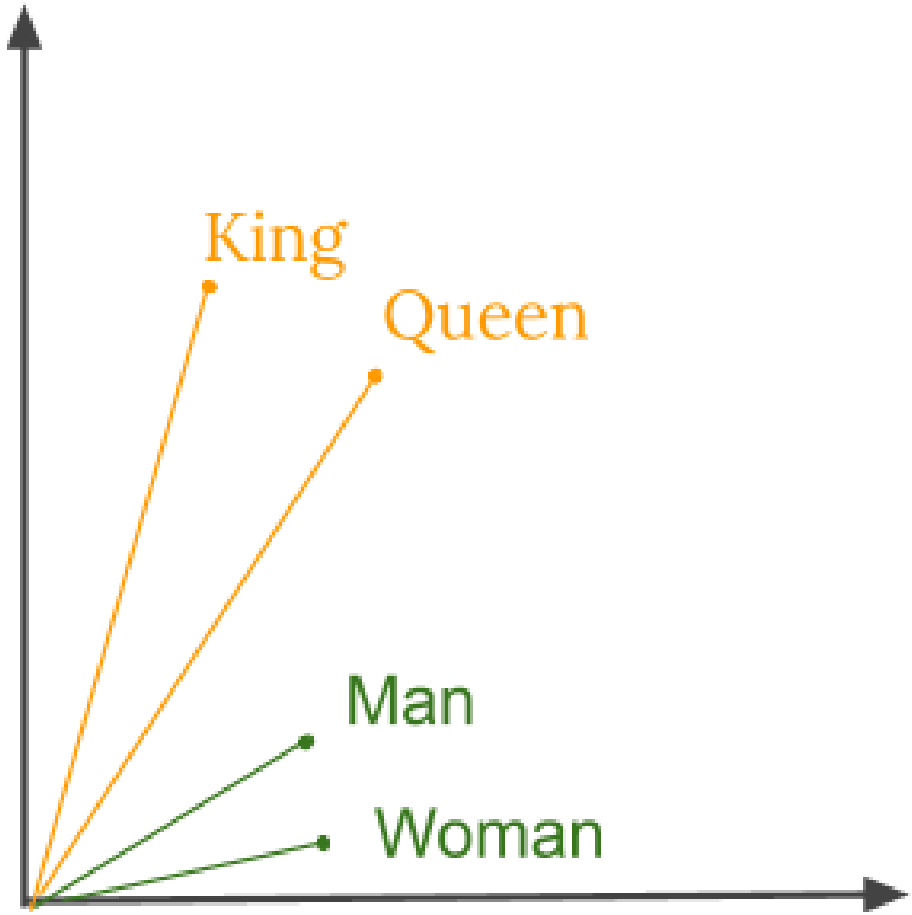
What is Word Embedding

- It should be nice to have representations of text in an n-dimensional space where words that have the same meaning have a similar representation .
- Meaning that two similar words are represented by almost similar vectors that are very closely placed in a vector space.



What is Word Embedding

- Thus when using word embeddings, all individual words are represented as real-valued vectors in a predefined vector space.
- Each word is mapped to one vector and the vector values are learned in a way that resembles a neural network.



The Importance of Context

- The concept of embeddings arises from a branch of Natural Language Processing called - *Distributional Semantics*. It is based on the simple intuition that:
- **Words that occur in similar contexts tend to have similar meanings.**
- In other words, a word meaning is given by the words that it appears frequently with.

Conceptually it involves the mathematical embedding **from space with many dimensions** per word to a **continuous vector space with a much lower dimension**.

What is Word2Vec?

- Word2vec is a method to efficiently create word embeddings by using a two-layer neural network.
- It was developed by Tomas Mikolov, et al. at Google in 2013 as a response to make the neural-network-based training of the embedding more efficient and since then has become the de facto standard for developing pre-trained word embedding.
- The input of word2vec is a text corpus and its output is a set of vectors known as feature vectors that represent words in that corpus.

Word2Vec

- The Word2Vec objective function causes the words that have a similar context to have similar embeddings.
- Thus in this vector space, these words are really close. Mathematically, the cosine of the angle (Q) between such vectors should be close to 1, i.e. angle close to 0.
- Word2vec is not a single algorithm but a combination of two techniques - CBOW(Continuous bag of words) and Skip-gram model.
- Both these are shallow neural networks which map word(s) to the target variable which is also a word(s).
- Both these techniques learn weights which act as word vector representations.

Word2Vec : CBOW approach

- CBOW predicts the probability of a word to occur **given the words surrounding it**. We can consider a single word or a group of words. But for simplicity, we will take a single context word and try to predict a single target word.
- The English language contains almost 1.2 million words, making it impossible to include so many words in our example. So I will consider a small example in which we have only four words i.e. **live**, **home**, **they** and **at**. For simplicity, we will consider that the corpus contains only one sentence, that being, **They live at home**.

Word2Vec : CBOW approach

First, we convert each word into a one-hot encoding form. Also, we will not consider all the words in the sentence but ll only take certain words that are in a window.

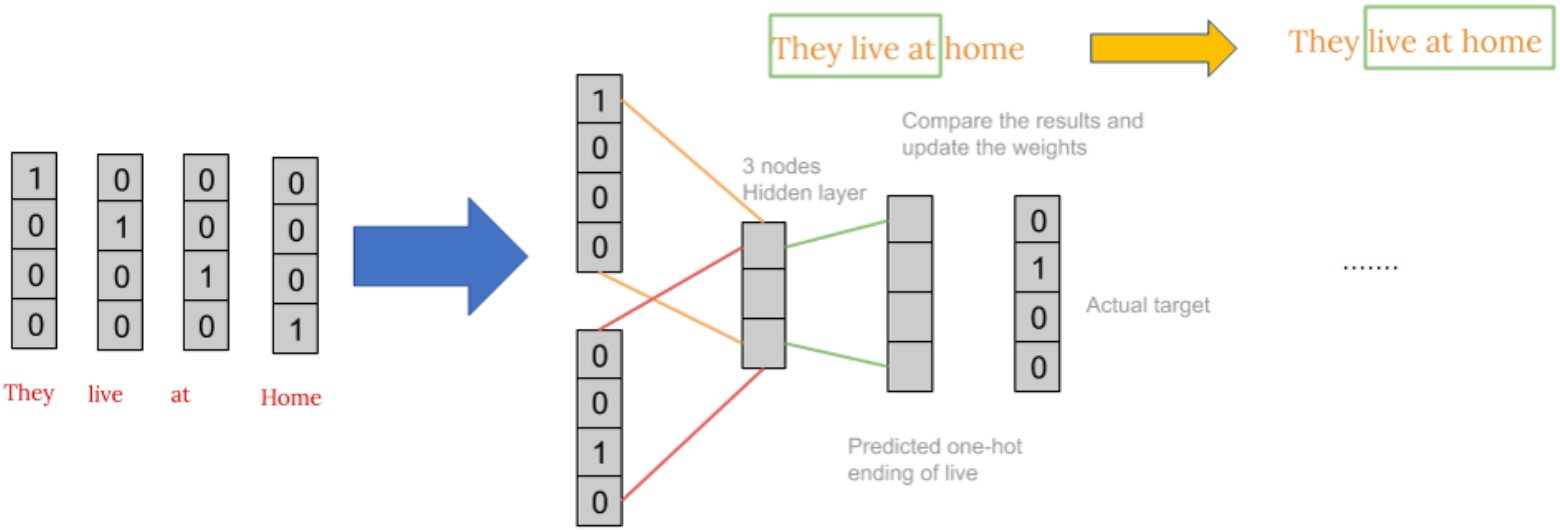


image source: *https://www.mygreatlearning.com/blog/word-embedding/*

Word2Vec : CBOW approach

For example, for a window size equal to three, we only consider three words in a sentence. The middle word is to be predicted and the surrounding two words are fed into the neural network as context. The window is then slide and the process is repeated again.

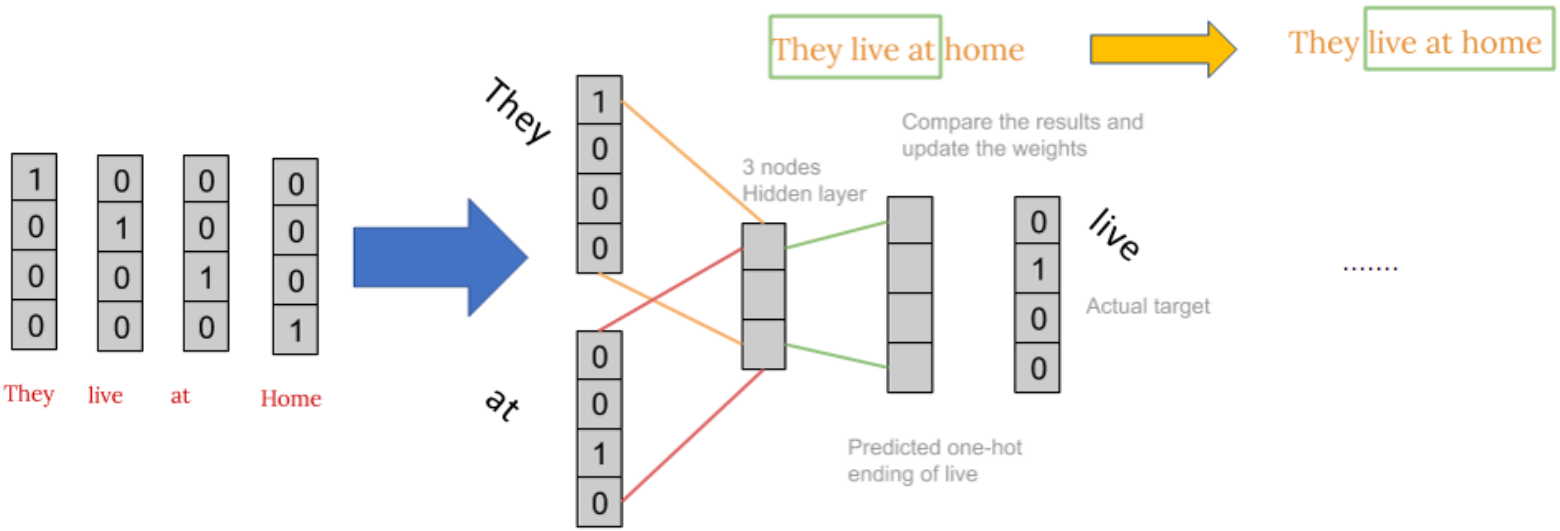


image source: *https://www.mygreatlearning.com/blog/word-embedding/*

Word2Vec : CBOW approach

- Finally, after training the network repeatedly by sliding the window a shown above, we get weights which we use to get the embeddings as shown below.
- Usually, we take a window size of around 8-10 words and have a vector size of 300.

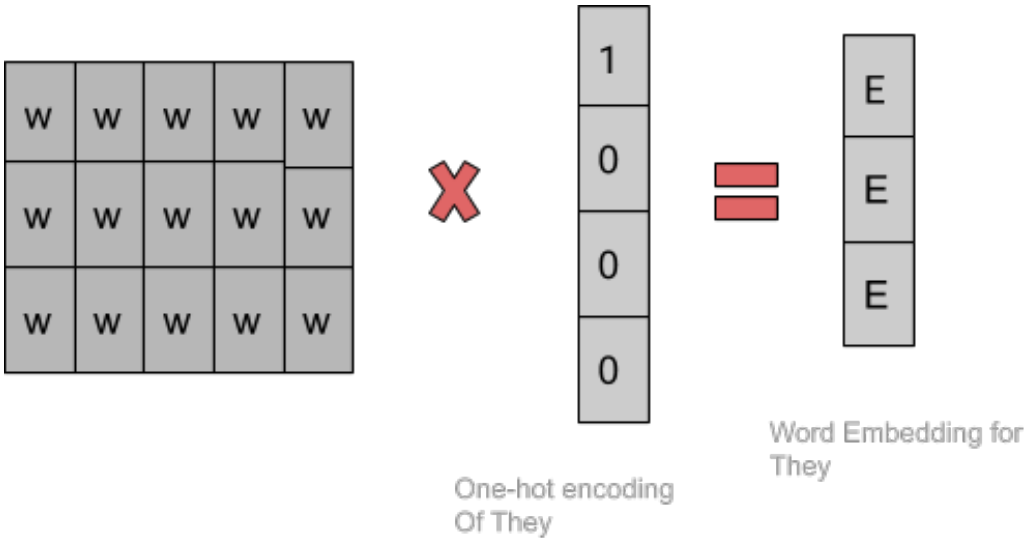


image source: *https://www.mygreatlearning.com/blog/word-embedding/*

Word2Vec : Skip-Gram Model

- The Skip-gram model tries to predict the source context words (surrounding words) given a target word (the centre word)
- The working is conceptually similar to the CBOW, there is just a difference in the architecture of its NN and in the way the weight matrix is generated:

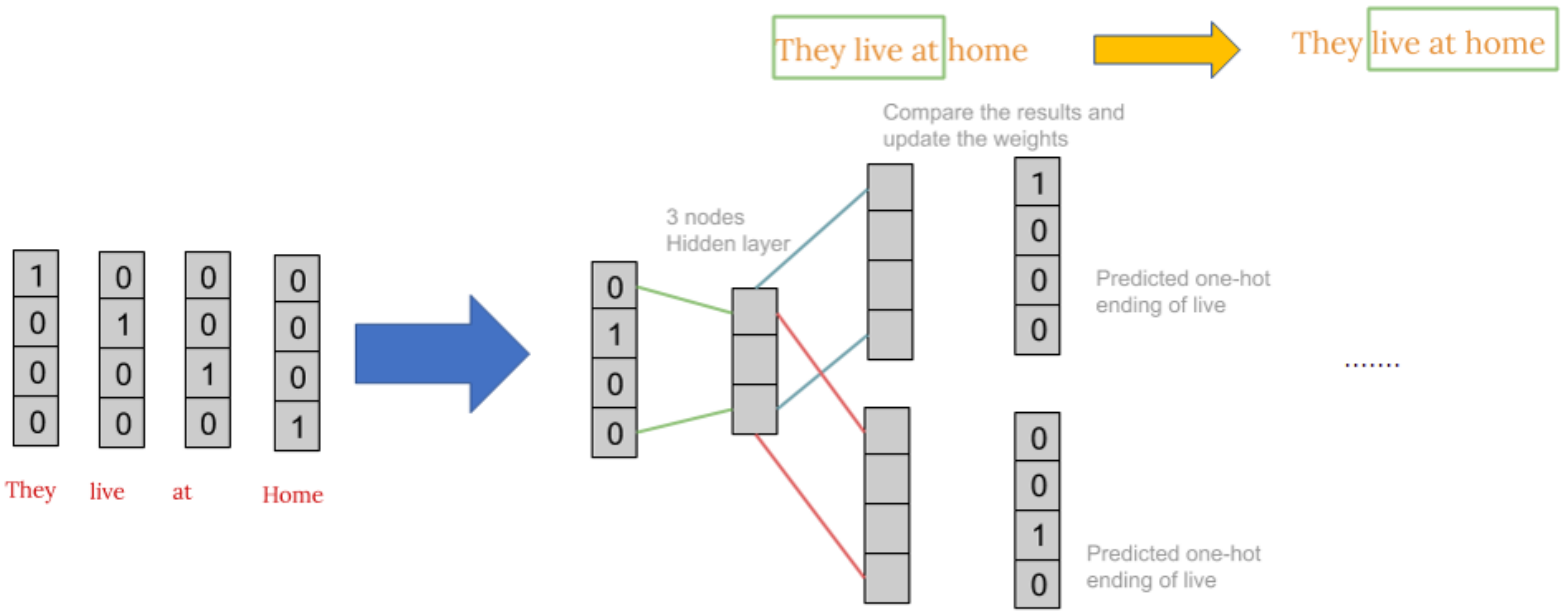


image source: *https://www.mygreatlearning.com/blog/word-embedding/*

Example : Word2Vec

- Notebook: **chapter-4-2**
- Focus: Implementing a word2vec model using a CBOW NN architecture
- Libraries: NLTK, keras

Text Classification

Introduction

- Text Classification is the process of labeling or organizing text data into groups - it forms a fundamental part of Natural Language Processing.
- In the digital age that we live in, we are surrounded by text on our social media accounts, commercials, websites, Ebooks, etc. The majority of this text data is **unstructured**, so classifying this data can be extremely useful.
- The premise of classification is simple: given a categorical target variable, learn patterns that exist between instances composed of independent variables and their relationship to the target.

Introduction

Text Classification: a formal definition

INPUT:

- a document d;
- a fixed set of classes $C = \{C_1, C_2, \ldots, C_n\}$

OUTPUT

- a predicted class $c \in C$

Introduction

Text Classification has a wide array of applications. Some popular uses are:

- Spam detection in emails
- Sentiment analysis of online reviews
- Topic labeling documents like research papers
- Language detection like in Google Translate
- Age/gender identification of anonymous users
- Tagging online content
- Speech recognition used in virtual assistants (like Siri and Alexa)

Introduction

Hand-coded rules

- Rules based on combinations of words or other features, for example: spam if black-list-address OR (“dollars” AND “you have been selected”)
- Accuracy can be high if rules carefully refined by expert
- But building and maintaining these rules is expensive

Introduction

- Because **the target is given ahead of time**, classification is a supervised machine learning model because a model can be trained to minimize error between predicted and actual categories in the training data.
- Once a classification model is fit, it assigns categorical labels to new instances based on the patterns detected during training.
- The application problem can be formulated to identify either a yes/no (binary classification) or discrete buckets (multiclass classification).

Introduction

Text Classification: Supervised Machine Learning

INPUT:

- a document d;
- a fixed set of classes $C = \{C_1, C_2, \ldots, C_n\}$
- a training set of m hand-labeled documents $(d_1, c_1), \ldots, (d_m, c_m)$

OUTPUT

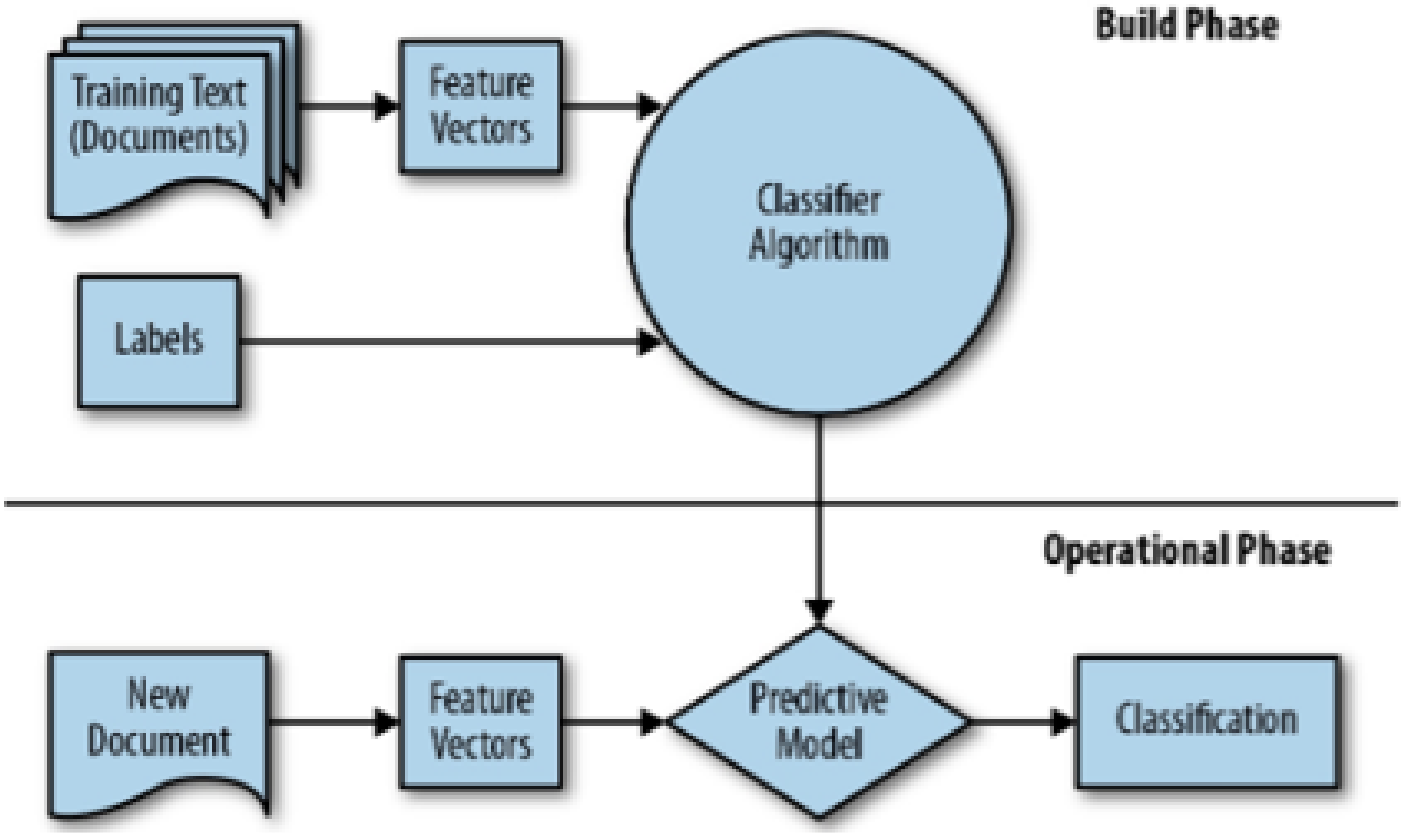
- a learned classifier $\gamma : d \rightarrow c$

Introduction

All classifier model families have the same basic workflow, and with Scikit-Learn Estimator objects, they can be employed in a procedural fashion and compared using cross-validation to select the best performing predictor. The classification workflow occurs in two phases: a build phase and an operational phase.

- In the build phase, a corpus of documents is transformed into feature vectors. The document features, along with their annotated labels (the category or class we want the model to learn), are then passed into a classification algorithm that defines its internal state along with the learned patterns.
- Once trained or fitted, a new document can be vectorized into the same space as the training data and passed to the predictive algorithm, which returns the assigned class label for the document.

Introduction



source: *Bengfort B. et al. Text Analysis with Python*

Example 1 - Gender Identification

In this first example we will try to build a simple algorithm to understand if a noun passed in input is of masculine or feminine gender. In this, and in the following examples, we will implicitly assume that the language used is English.

FOCUS

- How build a feature in text analysis
- Training Vs validation

Example 2 - The *20 Newsgroup* data set

- In this example we are going to work with the "20 Newsgroup Dataset".
- The 20 Newsgroups data set is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups. The 20 newsgroups collection has become a popular data set for experiments in text applications of machine learning techniques, such as text classification and text clustering.
- This data set is infact in-built in scikit, so we don't need to download it explicitly.

Example 2 - The *20 Newsgroup* data set

FOCUS

- using sklearn package
- numerical features and vectorization process
- building a ml pipeline with sklearn
- model optimization

Example : Text Classification

- Notebook: **chapter-4-3**
- Libraries: NLTK, keras

Subsection 1

Sentiment Analysis with Keras

From *Deep Learning with Python by F. Chollet*

Sentiment Analysis with Keras

- The IMDb Movie Reviews dataset is a binary sentiment analysis dataset consisting of 50,000 reviews from the Internet Movie Database (IMDb) labeled as positive or negative.
- The dataset contains an even number of positive and negative reviews.
- Only highly polarizing reviews are considered.
- A negative review has a score $= 4$ out of 10, and a positive review has a score $= 7$ out of 10.
- No more than 30 reviews are included per movie.
- The dataset contains additional unlabeled data.

The IMDb Movie Reviews Dataset

- Just like the MNIST dataset, the IMDB dataset comes packaged with Keras.
- It has already been preprocessed: the reviews (sequences of words) have been turned into sequences of integers, where each integer stands for a specific word in a dictionary.
- The following code will load the dataset (when you run it the first time, about 80 MB of data will be downloaded to your machine).

Listing 3.1 Loading the IMDB dataset

```
from keras.datasets import imdb

(train_data, train_labels), (test_data, test_labels) = imdb.load_data(
    num_words=10000)
```

The IMDb Movie Reviews Dataset

- **One-hot encode**
- All lists are transformed into vectors of 0s and 1s.
- This would mean, for instance, turning the sequence [3, 5] into a 10,000-dimensional vector that would be all 0s except for indices 3 and 5, which would be 1s.
- Then you could use as the first layer in your network a Dense layer, capable of handling floating-point vector data.

Listing 3.2 Encoding the integer sequences into a binary matrix

```
import numpy as np

def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results

x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
```

Creates an all-zero matrix of shape (len(sequences), dimension)

Sets specific indices of results[i] to 1s

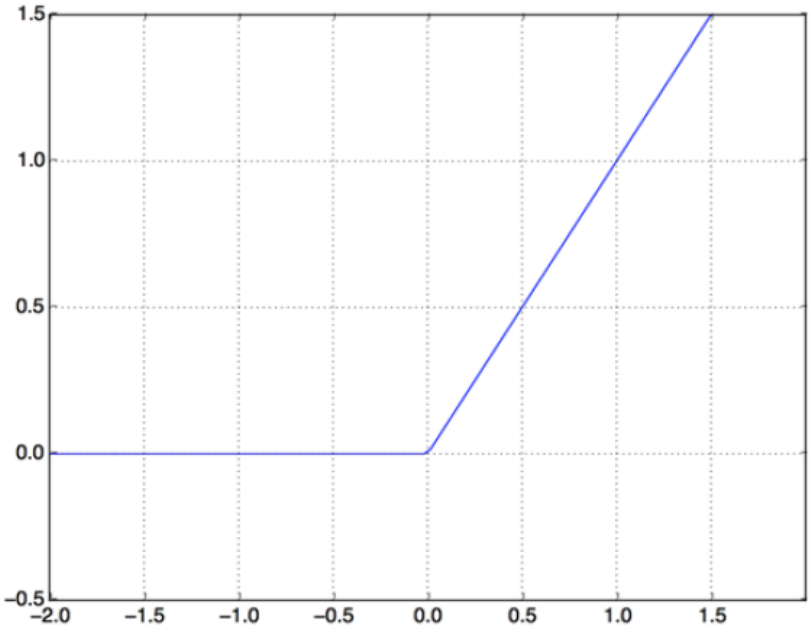
Vectorized training data

Vectorized test data

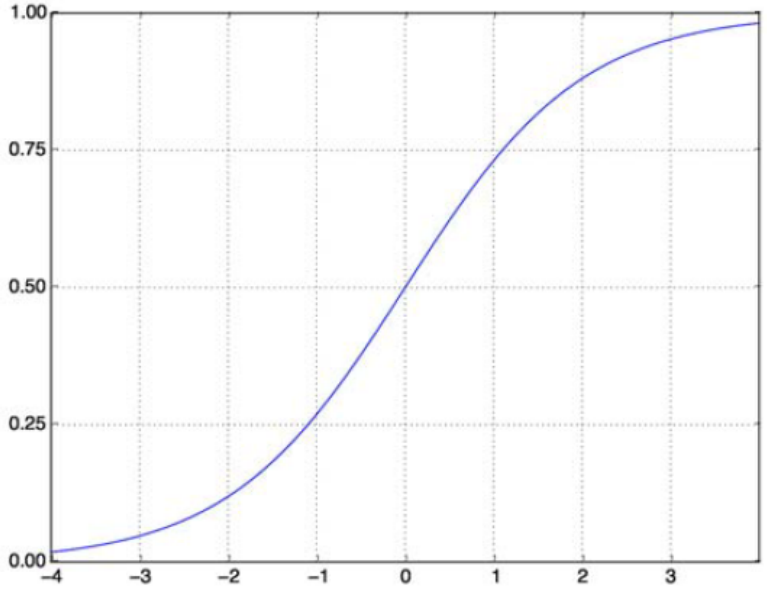
Preparing the data

- The input data is vectors, and the labels are scalars (1s and 0s).
- A type of network that performs well on such a problem is a simple stack of fully connected (Dense) layers with relu activations.
- The argument being passed to each Dense layer (16) is the number of hidden units of the layer.
- The intermediate layers will use relu as their activation function, and the final layer will use a sigmoid activation so as to output a probability (a score between 0 and 1, indicating how likely the review is to be positive).

Building our Network

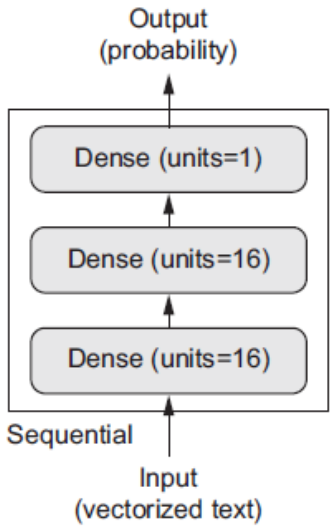


The rectified linear unit function



The sigmoid function

Building our Network



The three-layer network

Listing 3.3 The model definition

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

Building our Network

- Finally, we need to choose a loss function and an optimizer.
- Because we are facing a binary classification problem and the output of your network is a probability (we end our network with a single-unit layer with a sigmoid activation), it is best to use the binary_crossentropy loss.
- It is not the only viable choice: you could use, for instance, the well known mean squared error but crossentropy is usually the best choice when you're dealing with models that output probabilities.
- Crossentropy is a quantity from the field of Information Theory that measures the distance between probability distributions or, in this case, between the ground-truth distribution and your predictions.

Loss Function and Optimizer

- **RMSprop** is unpublished optimization algorithm designed for neural networks, first proposed by Geoff Hinton in lecture 6 of the online course **Neural Networks for Machine Learning**;
- **Adaptive learning rate** methods are an optimization of gradient descent methods with the goal of minimizing the objective function of a network by using the gradient of the function and the parameters of the network.

Listing 3.5 Configuring the optimizer

```
from keras import optimizers

model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

Example

- Now that we have described the construction of the neural network, let's see how it works in practice ...
- Using **chollet-deep-learning-with-python-sentiment** Notebook

Clustering for Text Similarity

Sorting Documents

Similarity

- At its core, any sorting task relies on our ability to compare two documents and determine their **SIMILARITY**.

Introduction

- Documents that are similar to each other are grouped together and the resulting groups broadly describe the overall themes, topics, and patterns inside the corpus.

Introduction

- While most document sorting is currently done manually, it is possible to achieve these tasks in a fraction of the time with the effective integration of unsupervised learning, as we will see in this lesson

Introduction

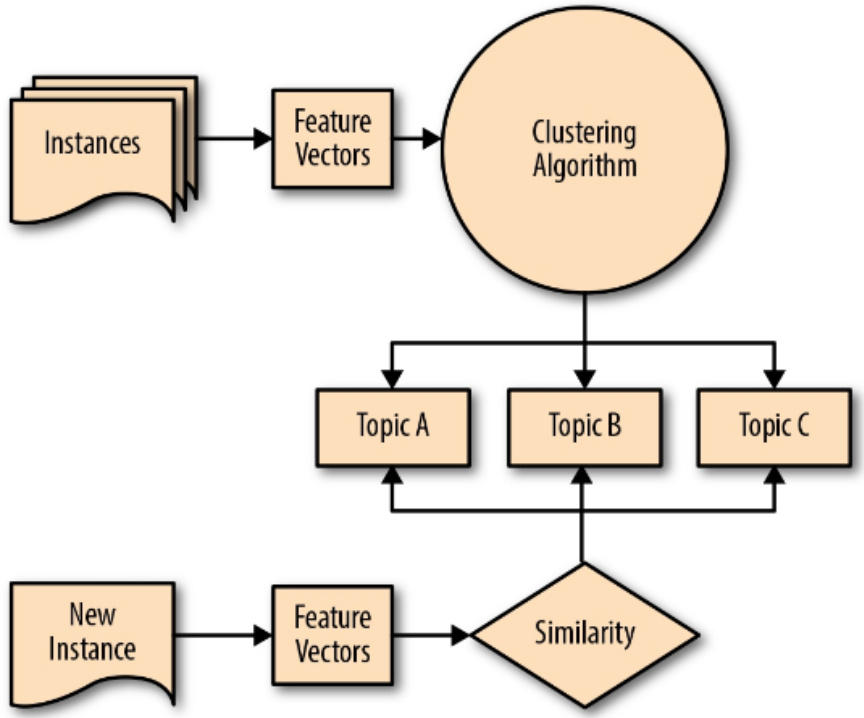
- In many situations, corpora do not arrive *pretagged* with labels ready for classification.
- In these cases, the only choice, or at least a necessary precursor for many natural language processing tasks, is an unsupervised approach.
- Clustering algorithms aim to discover **latent structure** or themes in unlabeled data using features to organize instances into meaningfully dissimilar groups.
- With text data, each **instance** is a single document or sentence, and the **features** are its tokens, vocabulary, structure, metadata, etc.

Unsupervised Learning on Text

- **Comparison between Clustering and Classification**
- The behavior of unsupervised learning methods is fundamentally different from that of supervised algorithm we have seen in the previous section;
- **Instead of learning a predefined pattern, the model attempts to find relevant patterns a priori**.
- A clustering algorithm is usually employed to create groups or topic clusters, using a distance metric such that documents that are closer together in feature space are more similar.
- New incoming documents can then be vectorized and assigned to the nearest cluster.

Unsupervised Learning on Text

- A corpus is transformed into feature vectors and a clustering algorithm is employed to create groups or topic clusters, using a distance metric such that documents that are closer together in feature space are more similar.
- New incoming documents can then be vectorized and assigned to the nearest cluster.



Unsupervised Learning on Text

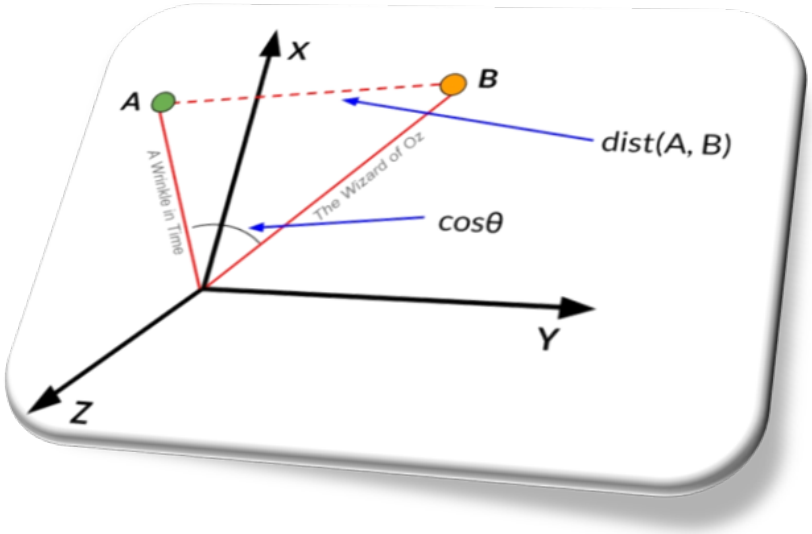
- As we have seen in section 3.1, there are a number of different measures that can be used to determine document similarity;
- Remember that, fundamentally, each relies on our ability to imagine documents as points in space, where the relative closeness of any two documents is a measure of their similarity.
- We have discussed the **cosine similarity** but there are others measure of distance we can use in clustering documents.

Unsupervised Learning on Text : Similarity

String Matching	Distance Metrics	Relational Matching	Other Matching
Edit Distance - Levenstein - Smith-Waterman - Affine **Alignment** - Jaro-Winkler - Soft-TFIDF - Monge-Elkan **Phonetic** - Soundex - Translation	- Euclidean - Manhattan - Minkowski **Text Analytics** - Jaccard - TFIDF - Cosine similarity	**Set Based** - Dice - Tanimoto (Jaccard) - Common Neighbors - Adar Weighted **Aggregates** - Average values - Max/Min values - Medians - Frequency (Mode)	- Numeric distance - Boolean equality - Fuzzy matching - Domain specific **Gazettes** - Lexical matching - Named Entities (NER)

Document Similarity

- We can measure vector similarity with cosine distance, using the cosine of the angle between the two vectors to assess the degree to which they share the same orientation.
- In effect, the more parallel any two vectors are, the more similar the documents will be (regardless of their magnitude).



Subsection 1

Clustering

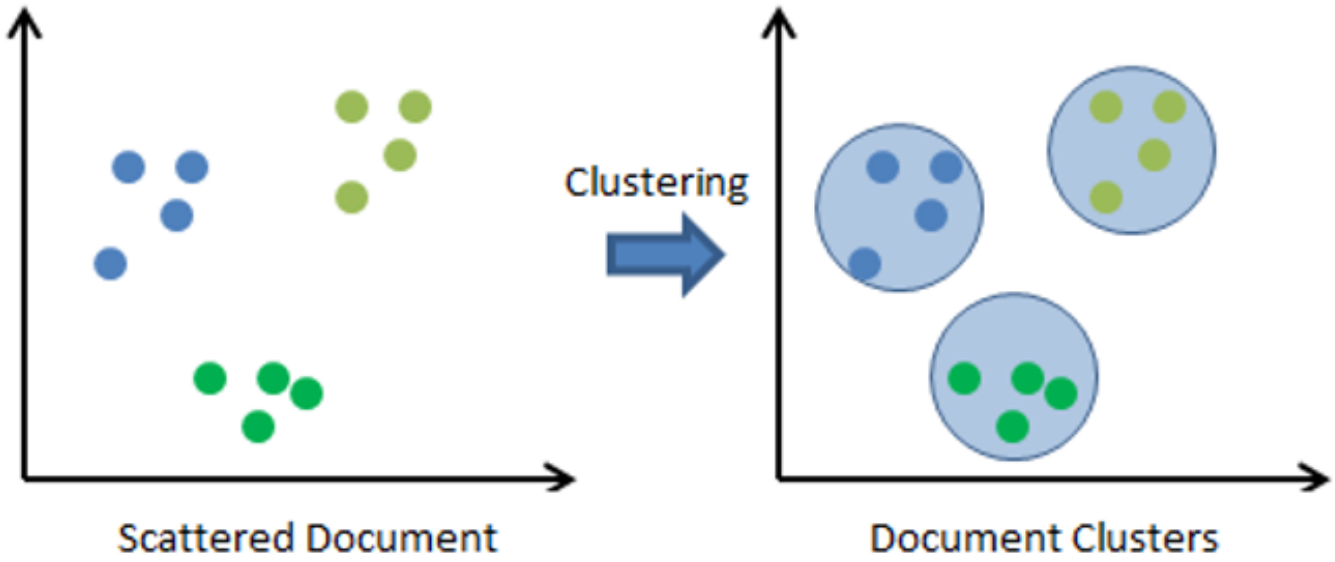
Clustering

- Now that we can quantify the similarity between any two documents, we can begin exploring unsupervised methods for finding similar groups of documents.
- **Partitive clustering** and **agglomerative clustering** are our two main approaches, and both separate documents into groups whose members share maximum similarity as defined by some distance metric.
- **We will focus on partitive methods, which partition instances into groups that are represented by a central vector (the centroid)** or described by a density of documents per cluster.
- Centroids represent an aggregated value (e.g., mean or median) of all member documents and are a convenient way to describe documents in that cluster.

Clustering

- As we have seen in the first part, clustering can be considered one of the most important unsupervised learning problem; so, as every other problem of this kind, it deals with finding a structure in a collection of unlabeled data.
- A loose definition of clustering could be *the process of organizing objects into groups whose members are similar in some way*.
- A cluster is therefore a collection of objects which are coherent internally, but clearly dissimilar to the objects belonging to other clusters.

Text Documents Clustering using K-Means Algorithm



source: *https://www.codeproject.com/Articles/439890/Text-Documents-Clustering-using-K-Means-Algorithm*

Text Documents Clustering using K-Means Algorithm

- **Document Representation**
- Each document is represented as a vector using the vector space model.
- The vector space model also called term vector model is an algebraic model for representing text document (or any object, in general) as vectors of identifiers. For example, TF-IDF weight.
- **Finding Similarity Score**
- We will use cosine similarity to identify the similarity score of a document.

Text Documents Clustering using K-Means Algorithm

- **A Practical Example: Clustering Movie Reviews**
- We are going to use data collected by Brandon Rose;
- here the link to the original post: http://brandonrose.org/top100
- And the GitHub Repository in which you can find all the data files necessary for this example: https://github.com/brandomr/document_cluster

Text Documents Clustering using K-Means Algorithm

5 Steps

- Prepare data
- Tokenizing and stemming each synopsis
- Transforming the corpus into vector space using tf-idf
- Calculating cosine distance between each document as a measure of similarity
- Clustering the documents using the k-means algorithm

Example : Clustering Document

- Notebook: **chapter-4-4**
- Libraries: NLTK, keras