Master Data Science and Innova

36118 Applied Natural Language Processing (ANLP)

Session 2

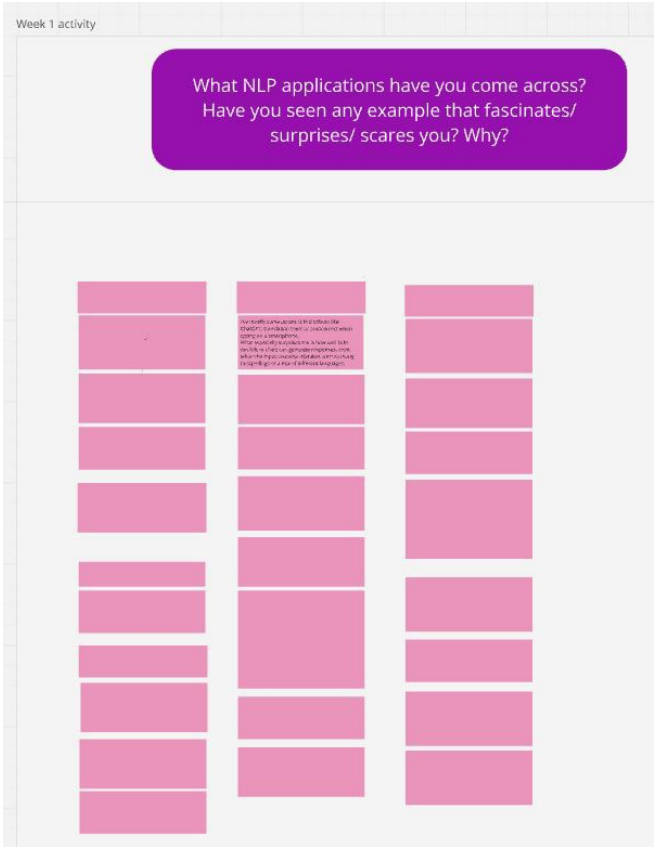
Dr. Antonette Shibani

Senior Lecturer, MDSI

Let’s start with our weekly check-in

Instructions to join Menti are provided in class

Miro Board



Class participation is counted towards AT3!

Complete this before Week 4!

Questions from Session 1

- Do we still need this level of manual work (e.g. creating custom stop words)?
- Fast-paced, some concepts are unclear
- Group work: 4-5 per team (max), 4 per team is recommended
- Miro board score tracking

Common approaches for NLP

1. **Rule-based heuristics** – Human-built heuristics provide a great start!

Using our implicit and explicit heuristics (E.g. Blacklist of words for spam, Regular expressions to extract email IDs)

Advantages:

- Possible with limited data
- More interpretable

2. **Machine Learning**
3. **Deep Learning**

} Covered in later sessions

In the last session

We started off by learning to

- Read textual data and display text
- Perform POS, NER tagging
- Build dependency trees
- Write regular expressions
- Clean punctuations and stop words
- Identify key words
- Calculate lengths and frequencies

At the individual word level!

Agenda for today's session

Analysing text

- nGrams (beyond single words)
- Relationships between words - Collocations, Associations, Concordances
- Text pre-processing pipeline
- Tf-idf

Agenda for today's session (2)

Analysis and Reporting

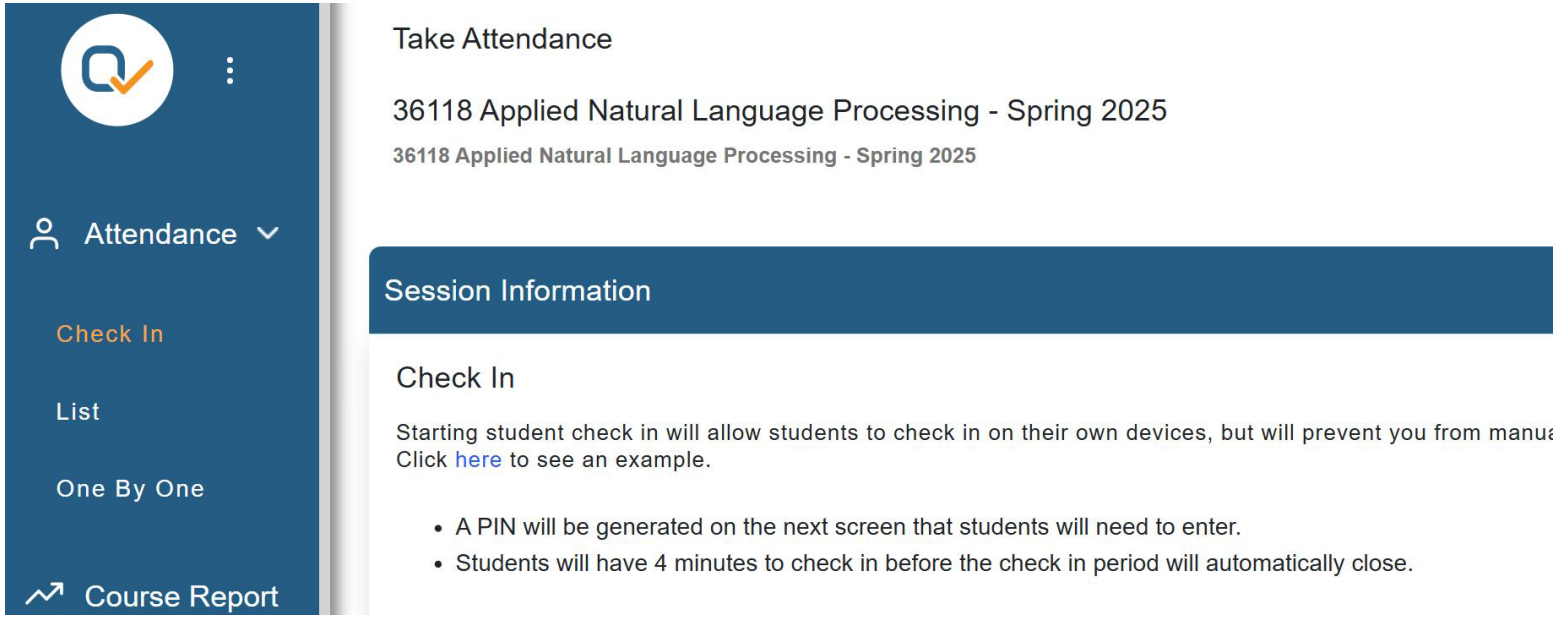
- Visualisations
- Storytelling for text data

Unsupervised techniques

- Topic modeling
- Clustering

This week's learning can be directly applied to AT1!

Qwickly attendance – Open on Canvas



N-grams

n-gram is a ***continuous sequence of n items*** from a given sample of text or speech

The items can be phonemes, syllables, letters, words, or base pairs according to the application (typically, we refer to sequences of words or characters)

Unigram (1-gram): A single item (word or character).
In the example sentence "I love learning," the unigrams are:

- "I"
- "love"
- "learning"

Bigram (2-gram): A sequence of two adjacent items.
For the same sentence "I love learning”, the bigrams are:

- "I love"
- "love learning"

Trigram (3-gram): A sequence of three adjacent items.
For the same sentence, the trigrams are:

- "I love learning"

N-gram: A sequence of n adjacent items

Most common bigrams and trigrams from our news data set

```
Top 10 Bigrams:
('legal', 'analyst'): 34
('new', 'york'): 20
('nicol', 'hemmer'): 20
('hemmer', 'associ'): 20
('associ', 'research'): 20
('research', 'scholar'): 20
('analyst', 'former'): 18
('frida', 'ghiti'): 16
('ghiti', 'fridagh'): 16
('fridagh', 'former'): 16
```

```
Top 10 Trigrams:
('nicol', 'hemmer', 'associ'): 20
('hemmer', 'associ', 'research'): 20
('associ', 'research', 'scholar'): 20
('legal', 'analyst', 'former'): 18
('frida', 'ghiti', 'fridagh'): 16
('ghiti', 'fridagh', 'former'): 16
('fridagh', 'former', 'prod'): 16
('michael', 'dantonio', 'author'): 16
('dantonio', 'author', 'book'): 16
('peniel', 'joseph', 'barbara'): 16
```

Notice the stemmed words from our cleaned data set!

We can dig into the words and their meaning a lot more….

Fundamental relations

Paradigmatic: The words have a paradigmatic relation if they are in the same class and can be substituted for each other.

E.g. “Monday” and “Friday”

Syntagmatic: The words can be combined with each other in a syntagmatic relation (co-occurrence).

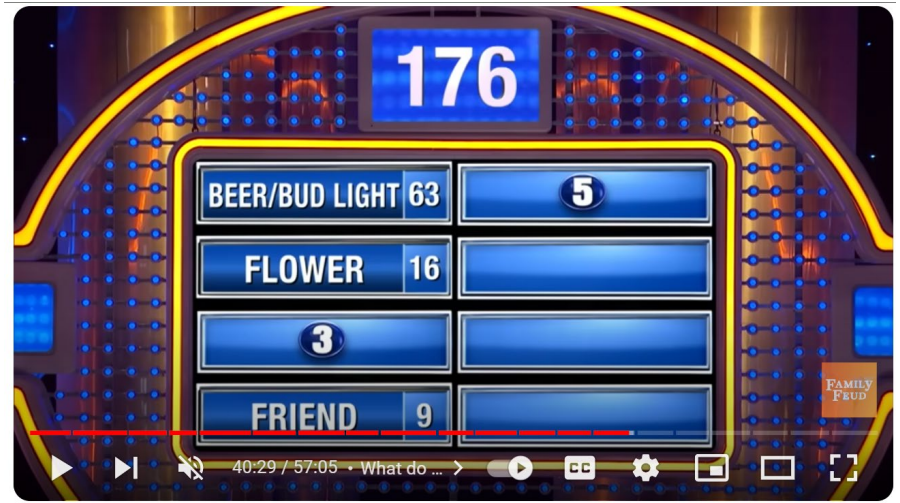
E.g. “bike” and “ride”

These relations could be extended to words, phrases, and other units, and are useful in many NLP applications.

Phonetic: Words that sound similar (e.g., "cat" - "hat")

Word association

Refers to the mental connection or relationship between words in a person's mind or in language use



Top 20 Family Feud rounds CRUSH Steve Harvey!!

Q: What do you think of when you hear the word “bud”?

Word association

Semantic relationship between a word and other words

- A person's lexical response to a lexical stimulus. *Stimulus* and *response* are the basic structural components of word association.

e.g. if one says *cat* the reply might be *dog*, or if the stimulus would be *bread* the response could be *butter*.

- Note that the responses may vary over the respondents (e.g. *bread* may evoke *butter* but also *breakfast* etc.

NLP methods

There are many methods to calculate word associations (they generally measure independence between words):

- Pointwise Mutual Information (PMI)
- Chi-square Test
- Cosine Similarity
- Latent Semantic Analysis (LSA)

Try them out!

Extension topic: PMI

Pointwise Mutual Information (PMI) measures how much the actual probability of two events (or words) occurring together differs from what we would expect if they were independent.

$$PMI(x, y) = \log_2(P(x,y) / (P(x) * P(y)))$$

where

- P(x,y) is the probability of x and y occurring together
- P(x) is the probability of x occurring
- P(y) is the probability of y occurring

PMI Example

Let's say we have a corpus of 1000 sentences:

"machine" appears in 100 sentences

"learning" appears in 80 sentences

"machine learning" appears together in 70 sentences

P(machine) = 100/1000 = 0.1

P(learning) = 80/1000 = 0.08

P(machine, learning) = 70/1000 = 0.07

PMI("machine", "learning") = $\log_2(0.07 / (0.1 * 0.08)) \approx 3.13$

PMI interpretation

PMI > 0:
Words co-occur more often than expected by chance

PMI = 0:
Words co-occur exactly as often as expected by chance

PMI < 0:
Words co-occur less often than expected by chance

Read more: https://eranraviv.com/understanding-pointwise-mutual-information-in-statistics/

Word collocation

- Collocations are **phrases or expressions containing multiple words, that are highly likely to co-occur** (Syntagmatic association)

Examples: ‘social media’, ‘machine learning’, ‘dog barks’

- More meaningful than bigrams and trigrams using a Pointwise Mutual Information (PMI) score., removing low frequency candidates.

E.g. I learned NLP

- *“learned NLP”* is arguably more meaningful than *“I learned”*

Word collocation from our news data set

```
from nltk import BigramAssocMeasures
bigram_measures = BigramAssocMeasures()
finder = BigramCollocationFinder.from_words(tokenized_words)

finder.nbest(bigram_measures.likelihood_ratio, 20)
```

```
[('legal', 'analyst'),
 ('nicol', 'hemmer'),
 ('research', 'scholar'),
 ('hemmer', 'associ'),
 ('associ', 'research'),
 ('ghiti', 'fridagh'),
 ('jill', 'filipov'),
 ('frida', 'ghiti'),
 ('joseph', 'barbara'),
 ('peniel', 'joseph'),
 ('barbara', 'jordan'),
 ('michael', 'dantonio'),
 ('eli', 'honig'),
 ('joe', 'biden'),
 ('new', 'york'),
 ('jordan', 'chair'),
 ('jeffrey', 'sach'),
 ('tess', 'taylor'),
 ('gene', 'seymour'),
 ('poetri', 'collec')]
```

Collocation vs association

- Collocation originates from patterns of language use in actual texts (beyond chance), whereas association originates from mental connections and cognitive processes.
- Both consist of two structural members and asymmetry laid upon them:

»access member« (AM)

»related member« (RM).

These two are called »stimulus« and »response« in word associations and »node« and »collocate« in the case of collocations

- Generally a moderate overlap between the two

De Deyne, S., & Storms, G. (2015). Word associations. In Taylor (Ed.), *The Oxford Handbook of the Word (Oxford Handbooks)* (p. 471). OUP Oxford: Kindle Edition.

Concordance

- Words used in a body of text, with their immediate contexts
- Useful to examine how words are used in context and to study the patterns of language use surrounding specific words

Concordances

We can further look up the locations at which a given word occurs in the news articles using a concordance analysis.

```
from nltk.text import Text
textlist = Text(tokenized_words)
print(textlist)
textlist.concordance('fbi')
textlist.concordance("fbi", width=100, lines=10)
```

```
<Text: 0 washington fbi director christoph wray 1 whether...>
Displaying 5 of 5 matches:
0 washington fbi director christoph wray 1 whether com
mob trump support storm 32 washington fbi director christoph wray 33 whether co
l digit decept project 244 washington fbi director christoph wray 245 olymp fen
bayern 295 deadli week 296 washington fbi director christoph wray 297 eli honig
ader among proud boy b 491 washington fbi director chri wray told l 492 school
Displaying 5 of 5 matches:
0 washington fbi director christoph wray 1 whether comment hair
mpleif 31 mob trump support storm 32 washington fbi director christoph wray 33 whether comment hair
3 ann ravel digit decept project 244 washington fbi director christoph wray 245 olymp fencer ryo mi
aghi call bayern 295 deadli week 296 washington fbi director christoph wray 297 eli honig legal ana
90 four leader among proud boy b 491 washington fbi director chri wray told l 492 school district v
```

Similarity metrics

- Useful to analyse which documents are similar (Think search engine queries and matching results)!
- Based on different distance measures for comparing texts

E.g. Jaccard distance, Cosine distance, Euclidian, Manhattan etc.

- Run it on Python: https://towardsdatascience.com/overview-of-text-similarity-metrics-3397c4601f50

You can use them in custom ways

An application I've used cosine similarity for - Automated revision graphs to compare revisions on a document (Research on writing)

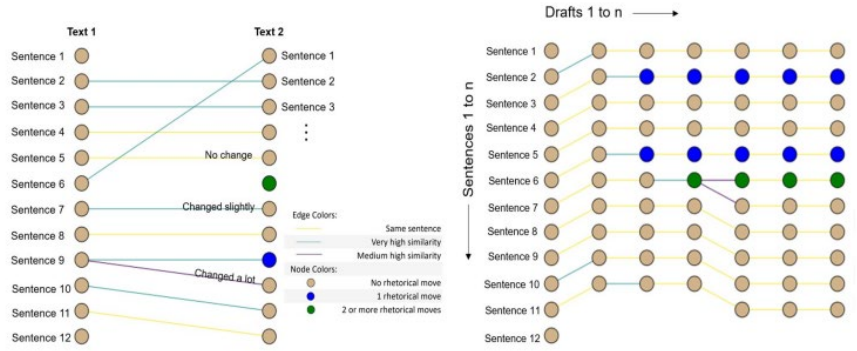
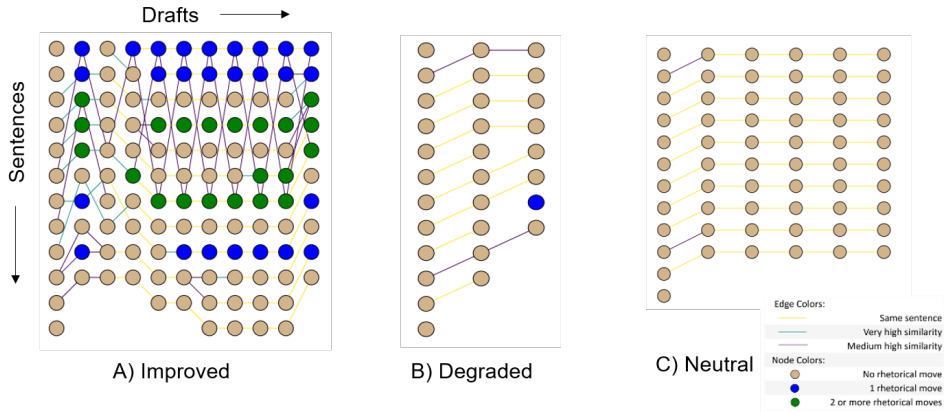


Fig 1: a) A simple revision graph example and **b)** Sample multi-stage revision graph with iterative changes.



Antonette Shibani (2020) Constructing Automated Revision Graphs: A Novel Visualization Technique to Study Student Writing. In: I. Bittencourt, M. Cukurova, K. Muldner, R. Luckin, & E. Millán (Eds.), *Artificial intelligence in Education – AIED 2020*

Text pre-processing

Generally, this is the first step in a text analysis pipeline

Clean up text and **remove inessential information** before performing automated analysis

Steps

- Remove extraneous symbols (punctuation, special symbols)
- Transform to lower case
- Remove digits (unless you need them!)
- Remove stop words (a, an, the etc.)
- Remove whitespace
- *Stem* to standardise variants (e.g organisation, organise, organising -→ organi). *Lemmatization* is a more sophisticated technique.
- Tokenise

The order matters!

You may need additional steps depending on the application

Stemming and Lemmatization of words

Form	Suffix	Stem
studies	-es	studi
studying	-ing	study

Stemming
Reducing words to their word stem, base or root form

Form	Morphological information	Lemma
studies	Third person, singular number, present tense of the verb study	study
studying	Gerund of the verb study	study

Lemmatization

Lemma is **the form of a word under which it is registered in a dictionary**. This is helpful because not all possible word forms of a word get their own entry in a lexicon.

Lemmatization

Original text: The cats are running quickly. They are better than dogs at catching mice

Lemmatized text:

The cat be run quickly. They be good than dog at catch mouse

- Provides more linguistically accurate base forms of words compared to stemming -> More interpretable, Can better preserve word's meaning
- Generally, more computationally intensive than stemming, so it might be slower for very large datasets

Main differences

Aspect	Stemming	Lemmatization
Definition	Reduces words to their base form	Reduces words to their lemma
Result	May not be a valid word	Always a valid word
Context Consideration	No	Yes
Example	"running" → "run"	"running" → "run"
Example	"better" → "bett"	"better" → "good"

String manipulations

There are many useful string methods in NLTK that you can use to manipulate texts as below - all methods produce a new string or list:

s.find(t) - index of first instance of string t inside s (-1 if not found)
s.rfind(t) - index of last instance of string t inside s (-1 if not found)
s.index(t) - like s.find(t) except it raises ValueError if not found
s.rindex(t) - like s.rfind(t) except it raises ValueError if not found
s.join(text) - combine the words of the text into a string using s as the glue
s.split(t) - split s into a list wherever a t is found (whitespace by default)
s.splitlines() - split s into a list of strings, one per line
s.lower() - a lowercased version of the string s
s.upper() - an uppercased version of the string s
s.title() - a titlecased version of the string s
s.strip() - a copy of s without leading or trailing whitespace
s.replace(t, u) - replace instances of t with u inside s

Quiz time!

Join at menti.com | use code 4324 2425

Consider the following sentence: "The cat sat on the mat." Which of the following statements is most accurate about word frequencies, and why?

A. The most frequent word in the sentence is "the".	B. Word frequencies are calculated by counting the total number of words in a text.	C. Word frequencies are used to identify the most important words in a text.	D. The least frequent word in the sentence is "cat".
0	0	0	0

Feedback on word frequencies

A. Correct. In this sentence, the most frequent word is "the", as it appears twice. This is an accurate application of word frequency.

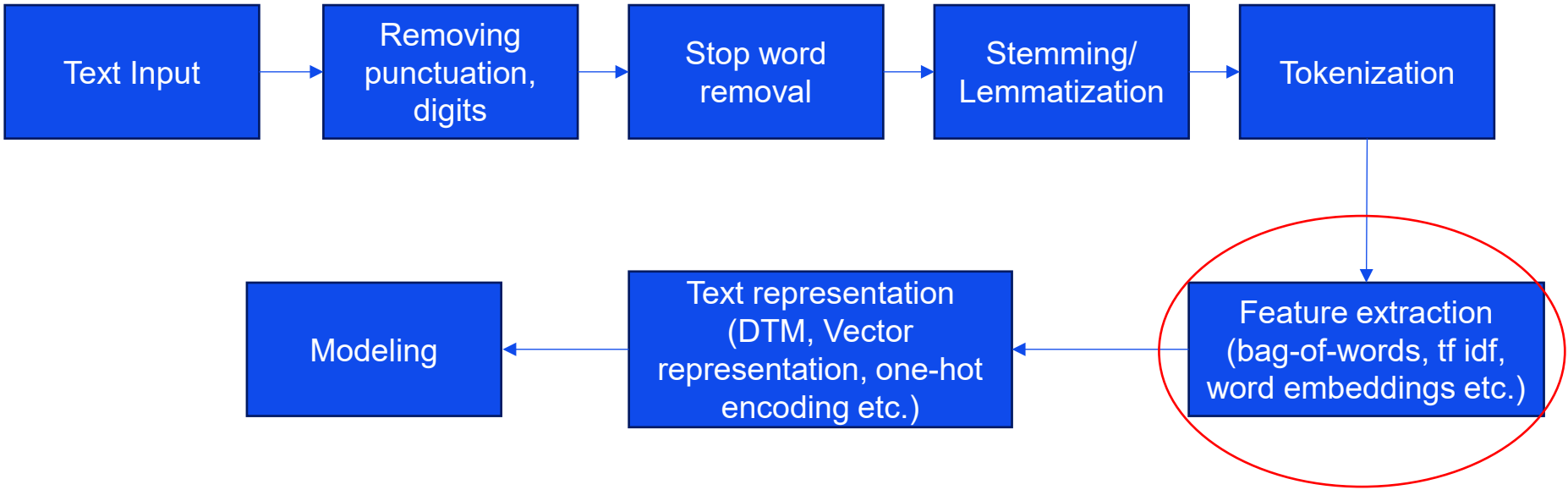
B. Incorrect. Word frequencies are calculated by counting the number of occurrences of each unique word in a text, not by counting the total number of words in a text.

C. Incorrect. While word frequencies can be used to identify the most frequent words in a text, they are not necessarily the most important words. Other techniques, such as TF-IDF, can be used to identify the importance of words in a text.

D. Incorrect. In this sentence, the least frequent word is "mat", as it appears only once.

Alternative Conception: Word frequencies are calculated by counting the number of occurrences of each unique word in a text.

Basic text analysis pipeline



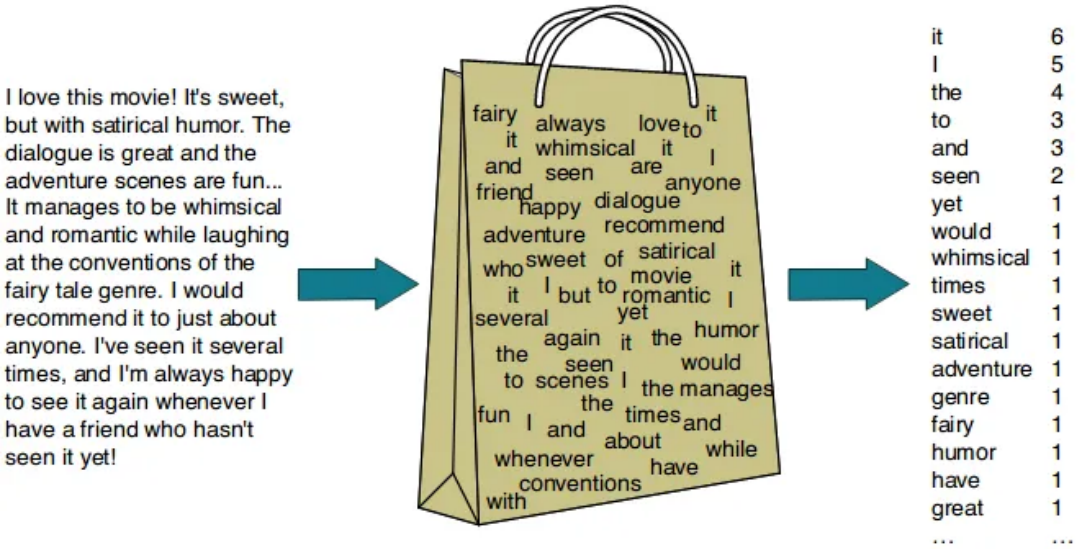
Feature extraction

- Transforming raw text into numerical or categorical features that the machine can process (converting unstructured to structured data)
- Many feature extraction techniques are available:

  Bag-of-words (BoW), Term Frequency-Inverse Document Frequency (TF-IDF), Word embeddings, N-grams, topic modeling features, deep learning based features etc.
- The choice of feature extraction method depends on the specific NLP task, the nature of the text data, and the ML algorithm being used (Often evaluated indirectly through the performance of downstream NLP tasks)

Bag of words (BoW) model

Treats a document as a collection of words, disregarding grammar, word order, and context



Source: https://ogre51.medium.com/nlp-explain-bag-of-words-3b9fc4f211e8

Text representation – DTM/ TDM

A Document-Term Matrix (DTM), also known as a Term-Document Matrix (TDM), is a mathematical matrix that represents a collection of documents as vectors in a high-dimensional space.

In a DTM:

- Rows represent individual documents in the corpus
- Columns represent unique terms (words) from the entire vocabulary (could be other tokens too E.g. n-grams)
- Cells contain a number representing term counts (or some other metric)

DTM

A toy example using two documents:

Doc1: bananas are yellow

Doc2: bananas are good

The DTM is:

	bananas	*are*	*yellow*	*good*
Doc1	1	1	1	0
Doc2	1	1	0	1

A DTM is a *mathematical representation of text*, but one in which *order does not matter* (“Bag of Words”)

Another example of a DTM using BoW

Document 1

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

Document 2

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

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

Stopword List

for
is
of
the
to

Count vs Tf-idf

- Corpus is usually created based on the frequency of words in the document
- However, most frequent words are not always the most important words! Frequently occurring words may not be indicative of meaning

Remember our example from the quiz?

The cat sat on *the* mat.

Term-frequency Inverse document frequency (Tf-idf)

- TF-IDF is a weighting method based on the following intuition:

terms that occur less often are more likely to be more descriptive of specific documents

TF = number of times word occurs in a document. Need to normalize by doc wordcount

- **TF = # of occurrences of word in doc / # of words in doc**

IDF – weigh down frequent terms and scale up rare terms.

- **IDF = Ln (# of docs / # of docs with word)**

- Vectorizer has a TF-IDF option that we can use in our models

See http://www.tfidf.com/ for an explanation of Tf-idf and look at tm documentation for details on how to get it working

Try it on Python: https://towardsdatascience.com/tf-idf-for-document-ranking-from-scratch-in-python-on-real-world-dataset-796d339a4089

Tf-idf example

Consider a document containing 100 words wherein the word *cat* appears 3 times.

The term frequency (i.e., tf) for *cat* is then (3 / 100) = 0.03.

Now, assume we have 10 million documents and the word *cat* appears in one thousand of these. Then, the inverse document frequency (i.e., idf) is calculated as log(10,000,000 / 1,000) = 4.

The Tf-idf weight is the product of these quantities: 0.03 * 4 = 0.12.

Building NLP pipelines with pre-processing steps

- Automated pipelines are best practice because they allow reproducible code: every step along the way is performed without human intervention
- Every choice is documented in the code and easy to change
- You can create pipelines for specific tasks (and adjust pre-processing steps as needed)

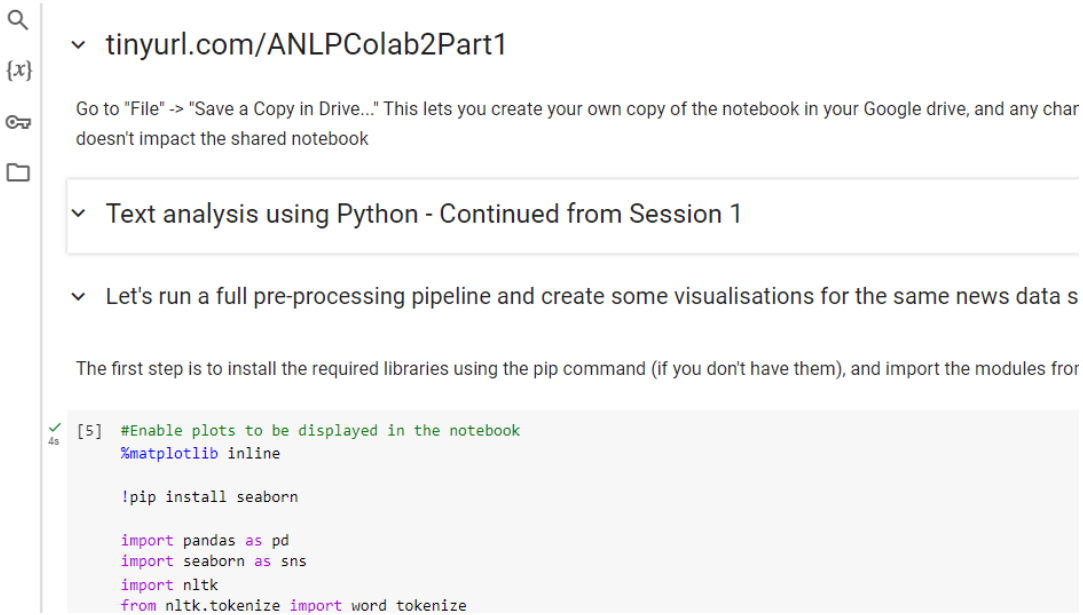
E.g. You can get rid of non-linguistic information such as URLs and hashtags if you want to learn what language a Tweet is written in

Remember

Don’t strip away all context!

Time to run the notebook!

tinyurl.com/ANLPColab2Part1



BREAK

Visualization and storytelling

Visualization in NLP

- Enhances understanding of complex data patterns and model behaviors
- Communicates results to a broader audience, including those without a technical background
- Visual tools can highlight trends, outliers, and anomalies that might be missed in raw data

We’ve already seen many kinds of visualisations….

- Different visualization techniques are suited for specific data types
- Thoughtfully select appropriate plot types!

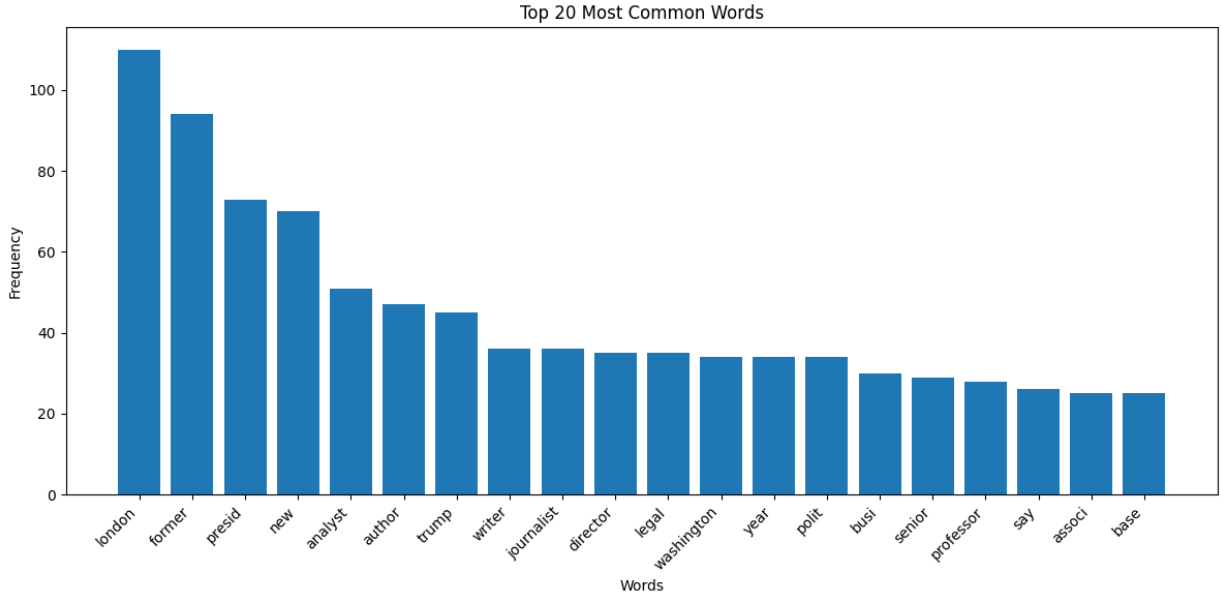
Wordclouds

Word Clouds to display the most frequent words in a text corpus – Useful for getting a quick sense of the main themes in the data

Remember how it helped spot issues in raw data?

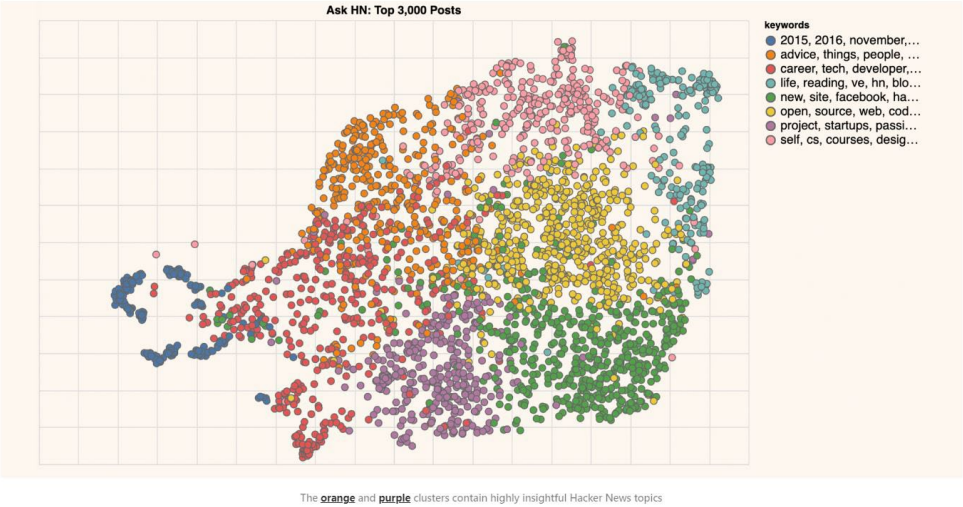
Bar Charts and Histograms

Show the frequency distribution of words, phrases, or other features – Useful for comparing occurrences



Scatter plots

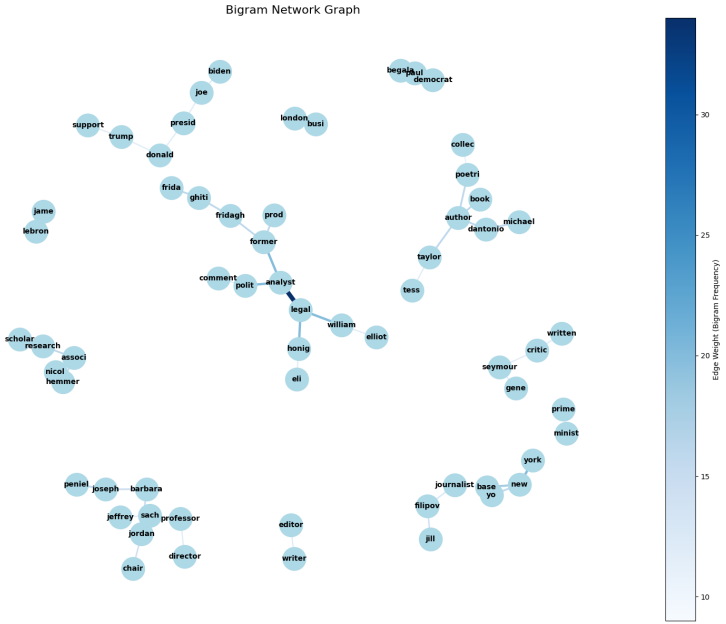
Display relationships between variables – Useful to identify correlations or clusters in the data



https://txt.cohere.ai/combing-for-insight-in-10-000-hacker-news-posts-with-text-clustering/

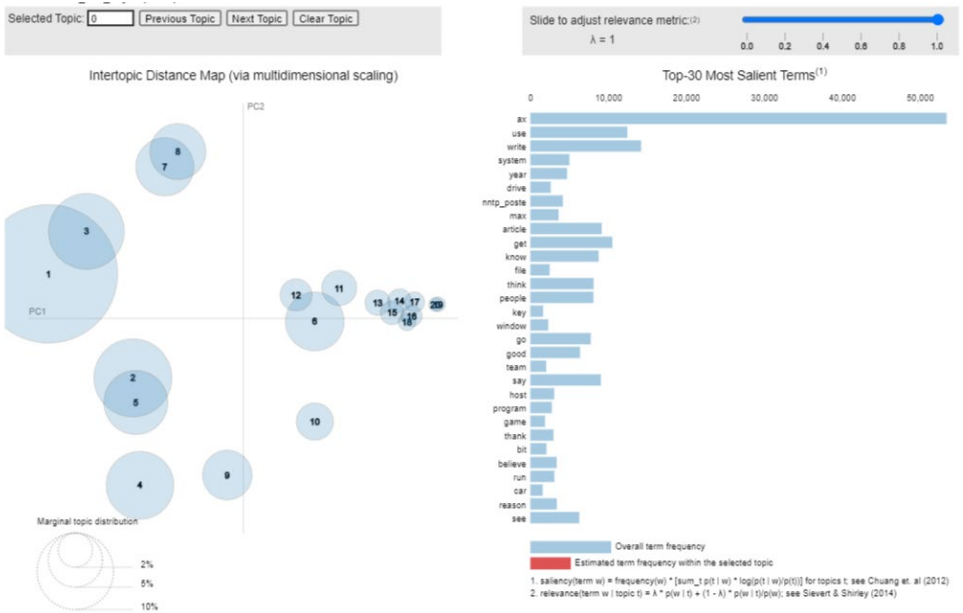
Network graphs

Represent connections and structures within a body of text – Useful to understand relationships between different textual elements



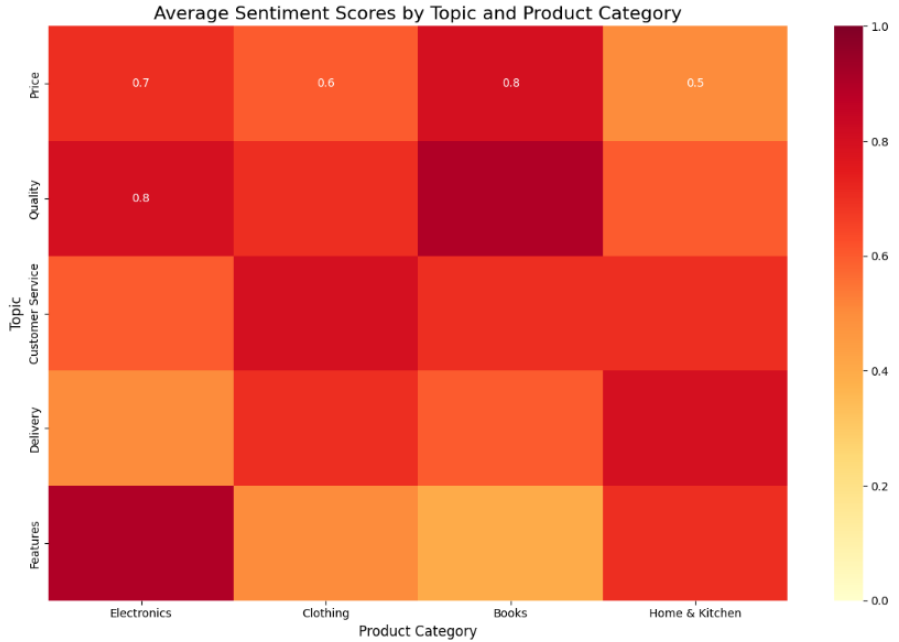
Topic model visualization

Helps interpret the underlying themes or topics within a large corpus of text – Useful for identifying the most probable or relevant words for each topic



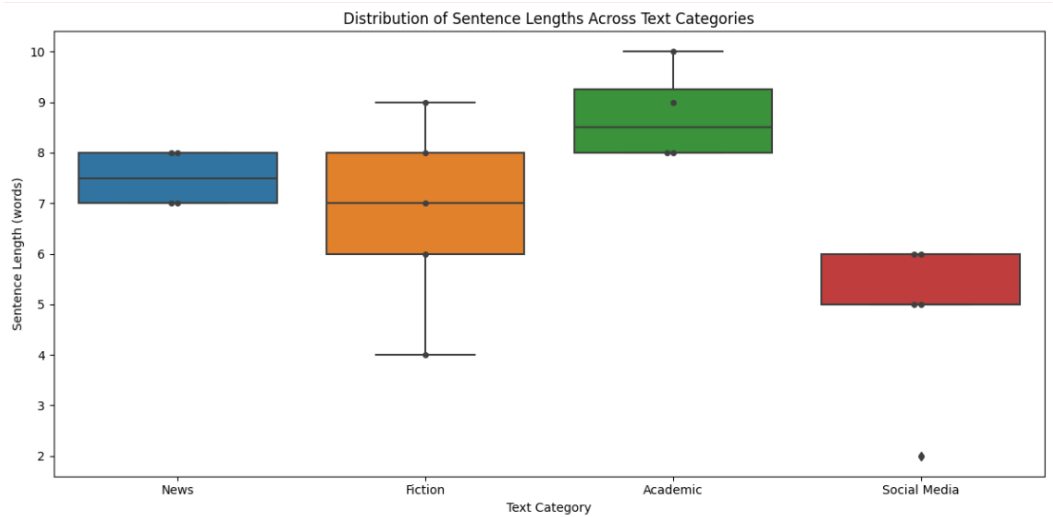
Heatmaps

Show the intensity of values in a matrix format – Useful for visualizing relationships between different features, correlations etc.



Box plots

Visualize the distribution of text-related metrics across different categories or groups



Category	count	mean	std	min	25%	50%	75%	max
Academic	4.0	8.75	0.957427	8.0	8.0	8.5	9.25	10.0
Fiction	5.0	6.80	1.923538	4.0	6.0	7.0	8.00	9.0
News	4.0	7.50	0.577350	7.0	7.0	7.5	8.00	8.0
Social Media	5.0	4.80	1.643168	2.0	5.0	5.0	6.00	6.0

There was a not-so-ideal form of viz in our tutorial…

Who can spot it?

Explain why it isn’t ideal

Take another look at the line graph!

Sensemaking and Storytelling

Presenting charts and graphs is not enough

(You cannot leave the sensemaking to your audience!)

Storytelling cheat sheet:
https://res.cloudinary.com/dyd911kmh/image/upload/v1662633286/Marketing/Blog/Data_Storytelling_Cheat_Sheet.pdf

Let’s walk through a well-executed data story

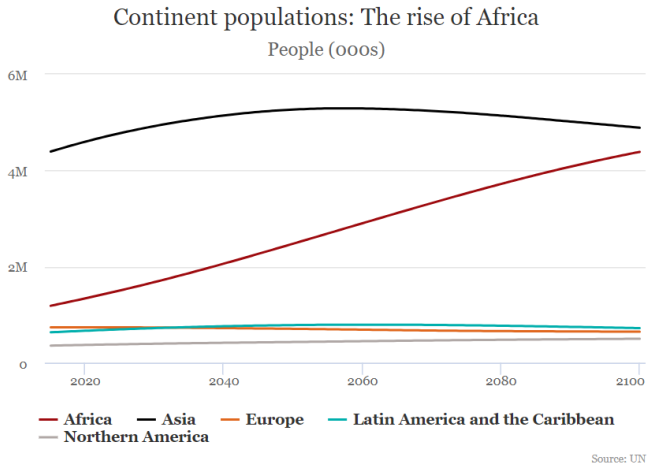
http://s.telegraph.co.uk/graphics/projects/Africa-in-100-years/index.html

Africa's population is booming. By 2100, it will be home to 4.4 billion people - four times its current population.

Comparison to current levels

Such an increase - far larger than the global population increase of 53 per cent by 2100 - will pose significant challenges. Poverty, conflict, disease and access to education are all issues African governments will continue to face, having to build states that can support ever-increasing amounts of people.

Comparison to global metrics highlight the significance of the problem



By 2050, more than half of Africa's 2.2bn people will live in its rapidly expanding cities. That's the equivalent of the population of China.

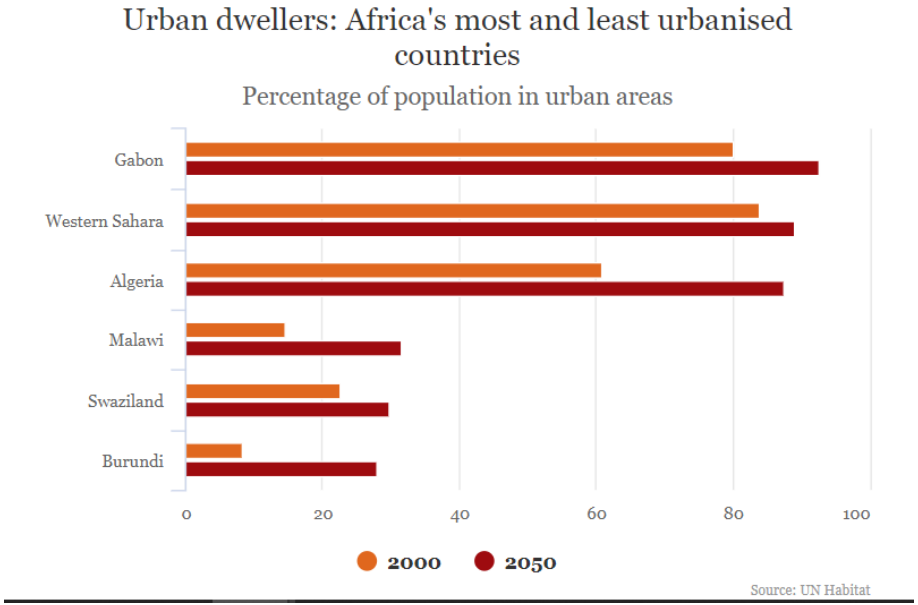
Demonstrating population growth with numbers and predictions

The UN has counted 71 African cities with a population higher than 750,000, many of which lack the infrastructure to support large populations. These cities are growing at an unstoppable pace - expected to hold 100m more people in 2025 than they did in 2010.

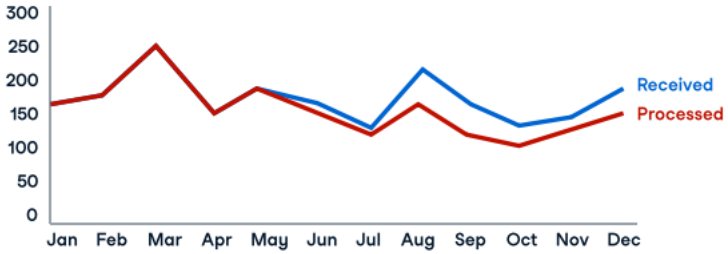
Past vs Future comparison

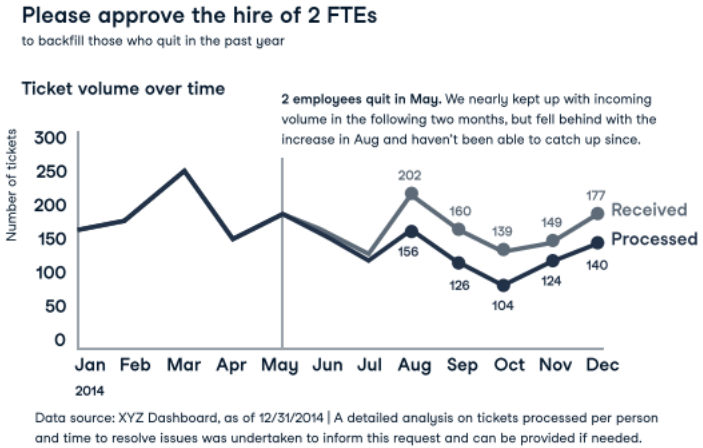
Just as governments will have to cope with higher populations, greater levels of urbanisation will also challenge countries as they seek to develop.

This massive population growth means rapid urbanisation. 864m more people will live in its urban areas in 2050 than in 2010 - the equivalent of adding the population of Europe. While just over a third of Africans lived in cities in 2010, 56.5 per cent will be urban-dwellers by 2050.



Give a clear indication of what the reader should focus on in your viz! (the accompanying text is powerful)





How text can be a useful visual tool when crafting effective visuals
(Source: Storytelling with Data by Cole Nussbaumer Knaflic)

Topic modeling

Why topic modeling?

- The standard way to search for documents is via keywords

….but often, we don’t know what we’re looking for

- You may have a large collection of documents, but you have no idea what they are about. *Topic modeling* is a technique that can help in this situation!

E.g. Documents containing words like *doctor, patient, health, hospital* can fall under a topic, and documents with words like *wheat, farm, rice, vegetables, cattle* can fall under another topic

Topics

Output: List of topics with associated clusters of words

Topic 1: *doctor, patient, health, hospital*

Topic 2: *wheat, farm, rice, vegetables, cattle*

Your have to make sense of the topic – What do you think the topics might be in the above?

doctor, patient, health, hospital
→
Healthcare

wheat, farm, rice, vegetables, cattle
→
Farming

Topic modeling

- A technique that enables the *unsupervised* discovery of topics in a collection of documents
- There are many algorithms for topic modelling - we'll use *Latent Dirichlet Allocation* (LDA)

Why *latent*?

- Because it discovers *hidden* topics by looking for important keywords, both in documents and across the entire corpus.

The number of topics (denoted by k) has to be specified upfront.

Say you have the following sentences:

1. I like ham and cheese
2. I ate a cheese sandwich for lunch
3. Rats and cockroaches are pests
4. Rats like cheese
5. Cats and rats don't get along

- You want to group them into 2 topics (# of topics is user specified)
- Here's what you might get

  {cheese, ham, sandwich} (topic A); {rats, cats, cockroaches} (topic B)

  - Sentences 1 and 2 - 100% topic A
  - Sentences 3 and 5 - 100% topic B
  - Sentence 4 - 50% topic A 50% topic B

Distribution of topics in the sentences

- Aim is to discover automatically discover topics in a collection of documents.

- Documents are known but
  - Topics
  - Per-document topic distributions
  - Per-document per-word topic assignments
- are *latent* (hidden). The word “topic” should not be taken literally – it is simply a bunch of words that occur together.

- The computational problem is to generate a latent topic structure which is consistent with the (known) document structure.

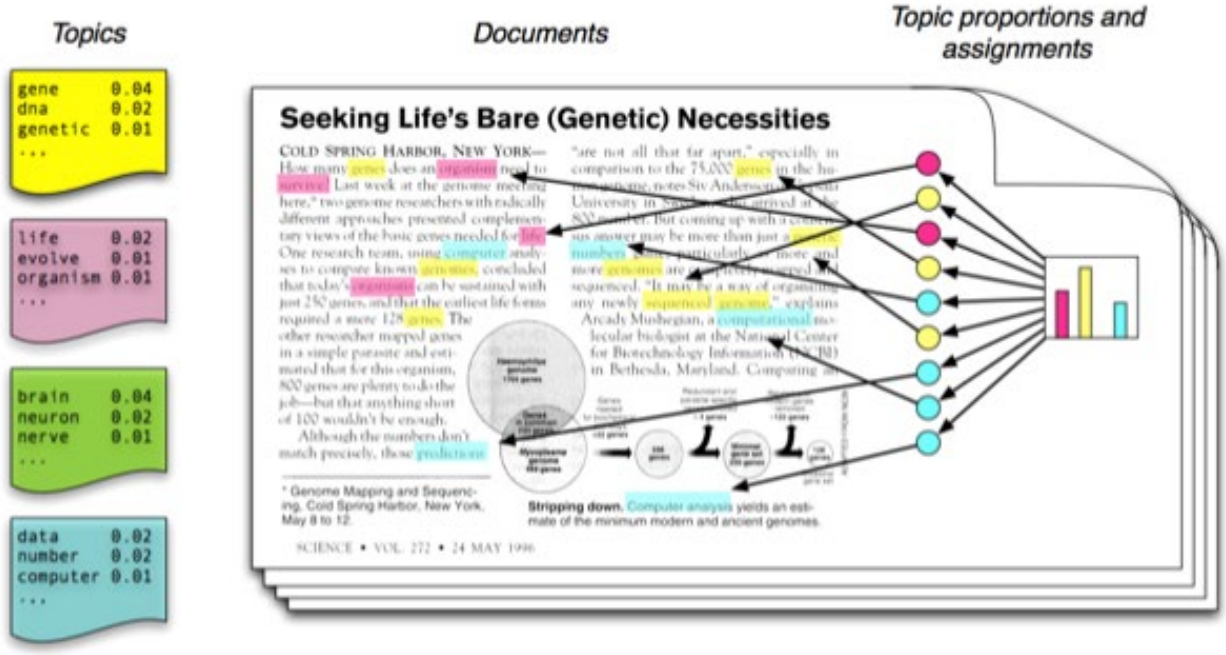
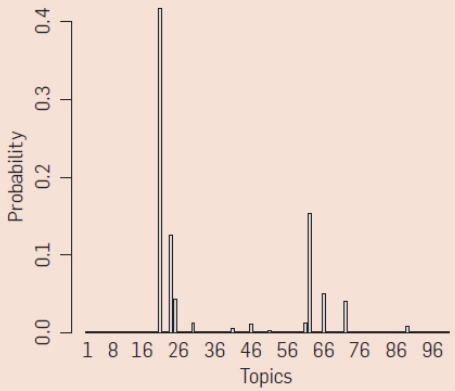


Figure source: Blei, D. M. (2012). Probabilistic topic models. *Communications of the ACM*, *55*(4), 77-84.

Wanli Xing, Hee-Sun Lee and **Antonette Shibani** (2020). Identifying patterns in students' scientific argumentation: content analysis through text mining using Latent Dirichlet Allocation. Educational Technology Research and Development. *https://doi.org/10.1007/s11423-020-09761-w.*



"Genetics"	"Evolution"	"Disease"	"Computers"
human	evolution	disease	computer
genome	evolutionary	host	models
dna	species	bacteria	information
genetic	organisms	diseases	data
genes	life	resistance	computers
sequence	origin	bacterial	system
gene	biology	new	network
molecular	groups	strains	systems
sequencing	phylogenetic	control	model
map	living	infectious	parallel
information	diversity	malaria	methods
genetics	group	parasite	networks
mapping	new	parasites	software
project	two	united	new
sequences	common	tuberculosis	simulations

Real inference with LDA fitting a 100-topic LDA model to 17,000 articles from the journal Science. At left are the inferred topic proportions for the example article in previous Figure. At right are the top 15 most frequent words from the most frequent topics found in this article.

How it works (simplified)

- Randomly assign every word in the corpus the k topics.
- For each document and word (in the document)

  Update the topic assignment for the word based on:

  - The relative importance of different topics in the current document.
  - How often the current word is associated with this topic across all documents.

  Iterate through the above steps until topic assignments become stable.
- See the following article for more details (at an intuitive level) -> must read!

  http://blog.echen.me/2011/08/22/introduction-to-latent-dirichlet-allocation/
- Why Dirichlet?

  Per-document topics and per-topic words are assumed to follow Dirichlet probability distributions.

Topic modeling in Python

LDA using Gensim

Let's use a library called Gensim for topic modeling. We start with text preprocessing over a number of steps as shown below:

Tokenization

```
[8] # Import function to convert a document into a list of tokens
    # This lowercases, tokenizes, and optionally de-accents the text
    # The output are tokens (unicode strings)
    from gensim.utils import simple_preprocess

    # Function to tokenize the text
    def tokenize(text):
        # Apply the simple_preprocess function
        text = simple_preprocess(str(text), deacc=True)
        return text

    # Tokenize all texts and store it in a variable
    tokenized_data = df["text"].apply(tokenize)

    print("Sample tokenized text:")
    print(" ".join(tokenized_data[0]))
```

```
Sample tokenized text:
was wondering if anyone out there could enlighten me on this car saw the other day it was door sports car looked to be from the late early
```

Building Bigram & Trigram Models

```
[9] # Function to detect phrases, based on collected collocation counts
    from gensim.models import Phrases
    from gensim.models.phrases import Phraser

    # Collect the bigrams
    bigram = Phrases(tokenized_data, min_count=3, threshold=100)
    bigram_model = Phraser(bigram)

    print("Sample bigrams:")
    print(", ".join(list(bigram.export_phrases().keys())[:5]))

    # Collect the trigrams
    trigram = Phrases(bigram[tokenized_data], threshold=100)
    trigram_model = Phraser(trigram)

    print("Sample trigrams:")
```

```
# This module implements the concept of a Dictionary
# i.e. a mapping between words and their integer ids
from gensim.corpora import Dictionary

# Create a dictionary
dictionary = Dictionary(lemmatized_data)

# Print sample mapping
keys = list(dictionary.token2id.keys())[:5]
vals = list(dictionary.token2id.values())[:5]
print("Sample dictionary mapping:")
for (key, val) in zip(keys, vals):
    print(f"{key} => {val}")
```

```
Sample dictionary mapping:
addition => 0
body => 1
bring => 2
call => 3
car => 4
```

```
[15] # Build a corpus for the topic model

     # Convert the document into a Bag-Of-Words format by producing
     # a list of tuples in (token_id, token_count) format
     corpus = [dictionary.doc2bow(text) for text in lemmatized_data]

     # Print a sample from the created corpus
     print(f"Text: {' '.join(lemmatized_data[0])}")
     print(f"Corpus: {corpus[0]}")
```

```
Text: wonder enlighten car see day door sport car look call door small addition separate rest body know model name engine spec year
Corpus: [(0, 1), (1, 1), (2, 1), (3, 1), (4, 4), (5, 1), (6, 2), (7, 1), (8, 1), (9, 1), (10, 1), (11, 1), (12, 1), (13, 1), (14, 2
```

Creating the Model

```
[16] # Module to train and use the Latent Dirichlet Allocation model
     from gensim.models.ldamodel import LdaModel

     # Initialize the model with the corpus and dictionary - this will take a while to run, be patient!
     lda_model = LdaModel(
        corpus=corpus, id2word=dictionary, num_topics=20,
        random_state=100, update_every=1, chunksize=100,
        passes=10, alpha='auto', per_word_topics=True
     )

     #You may fine tune parameters from the documentation here: https://radimrehurek.com/gensim/models/ldamodel.html. Try them out.
```

```
[17] # Save the model

     lda_model.save("lda_model") # Save the model as lda_model
```

Create a LDA models using tf (word counts) – this might take a while to run!

Creating the Model

```
[16] # Module to train and use the Latent Dirichlet Allocation model
     from gensim.models.ldamodel import LdaModel

     # Initialize the model with the corpus and dictionary - this will take a while to run, be patient!
     lda_model = LdaModel(
        corpus=corpus, id2word=dictionary, num_topics=20,
        random_state=100, update_every=1, chunksize=100,
        passes=10, alpha='auto', per_word_topics=True
     )

     #You may fine tune parameters from the documentation here: https://radimrehurek.com/gensim/models/ldamodel.html. Try them out.
```

K = 20

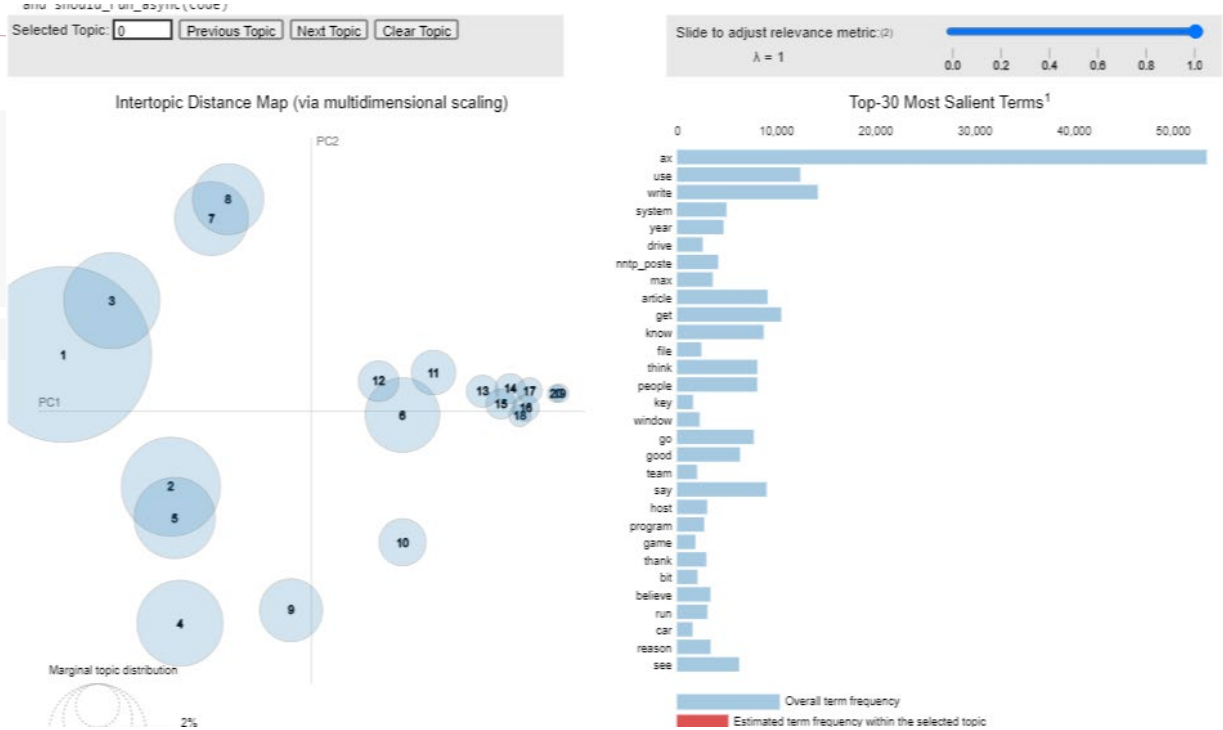
```
[17] # Save the model

     lda_model.save("lda_model") # Save the model as lda_model
```

Let’s visualise the LDA model (using counts) and explore the topics generated

Visualize with pyLDAvis

```
[19] import pyLDAvis # Module for interactive topic model visualization
     # Function to prepare LDA model for visualiztion
     from pyLDAvis.gensim import prepare

     # Enable the display of visualizations in IPython Notebooks
     pyLDAvis.enable_notebook()

     # Prepare and transform and LDA model
     pyLDAvis_data = prepare(lda_model, corpus, dictionary)
```

```
[20] pyLDAvis.display(pyLDAvis_data)
```



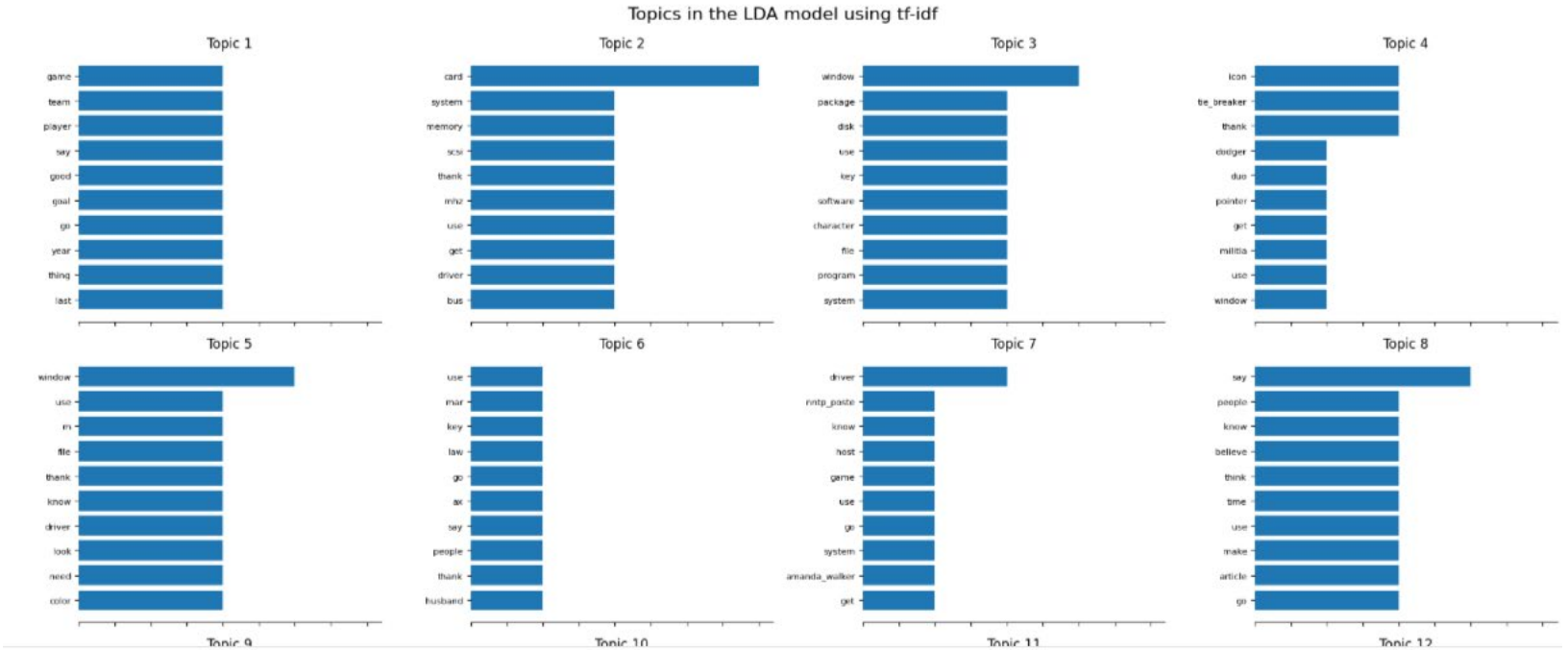
Create another LDA model using ***tf-idf*** to compare results!

```
from gensim.models import LdaMulticore
from gensim.models import TfidfModel
import matplotlib.pyplot as plt
import math

# Create TF-IDF model using the same pre-processed corpus
tfidf = TfidfModel(corpus)
corpus_tfidf = tfidf[corpus]

# Train the LDA model
num_topics = 20  # You can adjust this number
lda_model = LdaMulticore(corpus=corpus_tfidf, id2word=dictionary, num_topics=num_topics, random_state=42)

# Print the topics
print("Topics:")
for idx, topic in lda_model.print_topics(-1):
    print(f"Topic: {idx} \nWords: {topic}\n")
```

```
  and should_run_async(code)
WARNING:gensim.models.ldamulticore:too few updates, training might not converge; consider increasing the number of passes or iterations to improve accuracy
Topics:
Topic: 0
Words: 0.002*"game" + 0.002*"team" + 0.002*"player" + 0.002*"say" + 0.002*"good" + 0.002*"goal" + 0.002*"go" + 0.002*"year" + 0.002*"thing" + 0.002*"last"

Topic: 1
Words: 0.004*"card" + 0.002*"system" + 0.002*"memory" + 0.002*"scsi" + 0.002*"thank" + 0.002*"mhz" + 0.002*"use" + 0.002*"get" + 0.002*"driver" + 0.002*"bus"

Topic: 2
Words: 0.003*"window" + 0.002*"package" + 0.002*"disk" + 0.002*"use" + 0.002*"key" + 0.002*"software" + 0.002*"character" + 0.002*"file" + 0.002*"program" + 0.002*"system"

Topic: 3
Words: 0.002*"icon" + 0.002*"tie_breaker" + 0.002*"thank" + 0.001*"dodger" + 0.001*"duo" + 0.001*"pointer" + 0.001*"get" + 0.001*"militia" + 0.001*"use" + 0.001*"window"

Topic: 4
Words: 0.003*"window" + 0.002*"use" + 0.002*"m" + 0.002*"file" + 0.002*"thank" + 0.002*"know" + 0.002*"driver" + 0.002*"look" + 0.002*"need" + 0.002*"color"

Topic: 5
Words: 0.001*"use" + 0.001*"mar" + 0.001*"key" + 0.001*"law" + 0.001*"go" + 0.001*"ax" + 0.001*"say" + 0.001*"people" + 0.001*"thank" + 0.001*"husband"

Topic: 6
Words: 0.002*"driver" + 0.001*"nntp_poste" + 0.001*"know" + 0.001*"host" + 0.001*"game" + 0.001*"use" + 0.001*"go" + 0.001*"system" + 0.001*"amanda_walker" + 0.001*"get"

Topic: 7
Words: 0.003*"say" + 0.002*"people" + 0.002*"know" + 0.002*"believe" + 0.002*"think" + 0.002*"time" + 0.002*"use" + 0.002*"make" + 0.002*"article" + 0.002*"go"

Topic: 8
Words: 0.002*"use" + 0.002*"key" + 0.001*"get" + 0.001*"jewish" + 0.001*"article" + 0.001*"people" + 0.001*"arab" + 0.001*"chip" + 0.001*"right" + 0.001*"know"

Topic: 9
Words: 0.002*"think" + 0.001*"nntp_poste" + 0.001*"club" + 0.001*"team" + 0.001*"use" + 0.001*"idle" + 0.001*"people" + 0.001*"greek" + 0.001*"crt" + 0.001*"go"

Topic: 10
Words: 0.002*"helmet" + 0.002*"ride" + 0.002*"car" + 0.001*"saturn" + 0.001*"motorcycle" + 0.001*"bike" + 0.001*"pin" + 0.001*"honda" + 0.001*"accelerate" + 0.001*"get"

Topic: 11
Words: 0.002*"say" + 0.001*"scope" + 0.001*"get" + 0.001*"ground" + 0.001*"article" + 0.001*"good" + 0.001*"know" + 0.001*"reward" + 0.001*"email" + 0.001*"go"

Topic: 12
Words: 0.002*"[illegible]" + 0.001*"[illegible]" + 0.001*"packet" + 0.001*"lciii" + 0.001*"nntp_poste" + 0.001*"amateur_radio" + 0.001*"host" + 0.001*"thank"
```

https://stackoverflow.com/questions/44781047/necessary-to-apply-tf-idf-to-new-documents-in-gensim-lda-model#:~:text=As%20can%20be%20read%20in,to%20use%20the%20entire%20vocabulary

Terms contributing to topics visualised from our data set



Quiz time!

Join at menti.com | use code 4324 2425

Consider the following statements about Latent Dirichlet Allocation (LDA). Which of the following statements is most accurate about LDA, and why?

0

A. LDA is a clustering algorithm that groups similar documents together.

0

B. LDA is a supervised learning algorithm that requires labeled data.

0

C. LDA is a generative probabilistic model that represents documents as a mixture of topics.

Feedback on LDA

A. Incorrect. LDA is a topic modeling algorithm that groups words together based on their probability of co-occurring within a given topic, not documents.

B. Incorrect. LDA is an unsupervised learning algorithm that does not require labeled data.

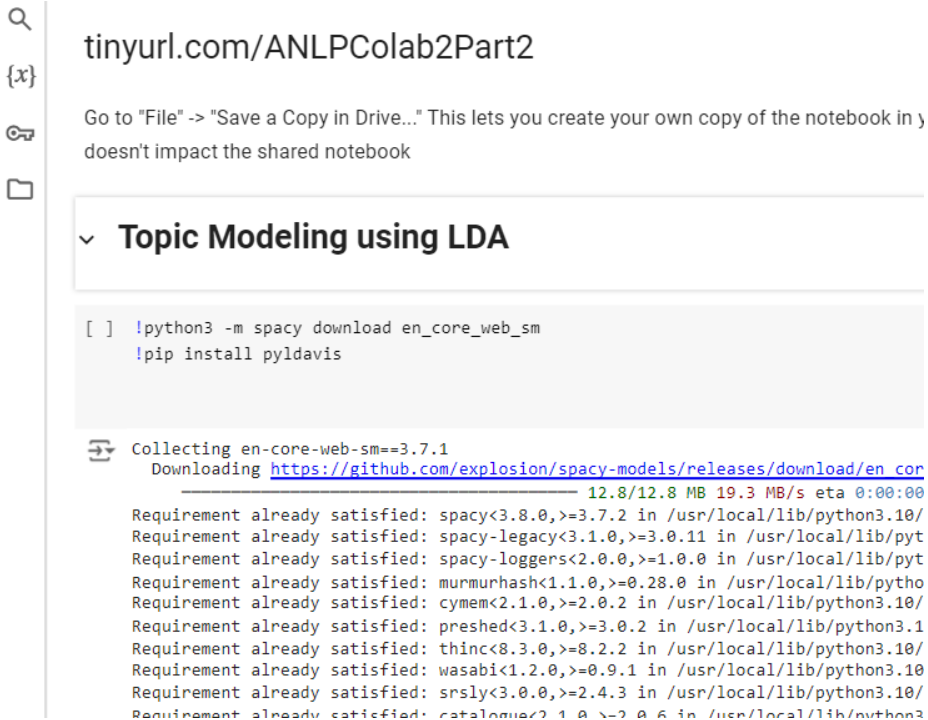
C. Correct. LDA is a generative probabilistic model that represents documents as a mixture of topics. It assumes that each document is a mixture of topics, and each topic is a probability distribution over words.

Alternative Conception: LDA is based on the assumption that documents are a mixture of topics and that topics are a probability distribution over words. The algorithm is unsupervised, meaning it does not require labeled data, and it groups words together based on their probability of co-occurring within a given topic.

Tutorial

Open Colab notebook:

tinyurl.com/ANLPColab2Part2



Clustering

Unsupervised machine learning

- The machine discovers patterns or groupings in data without the need for human intervention
- We do not know what the output will be
- No need for labelled training data

Examples:

- Topic modelling
- Clustering
- Association rules

Unsupervised machine learning

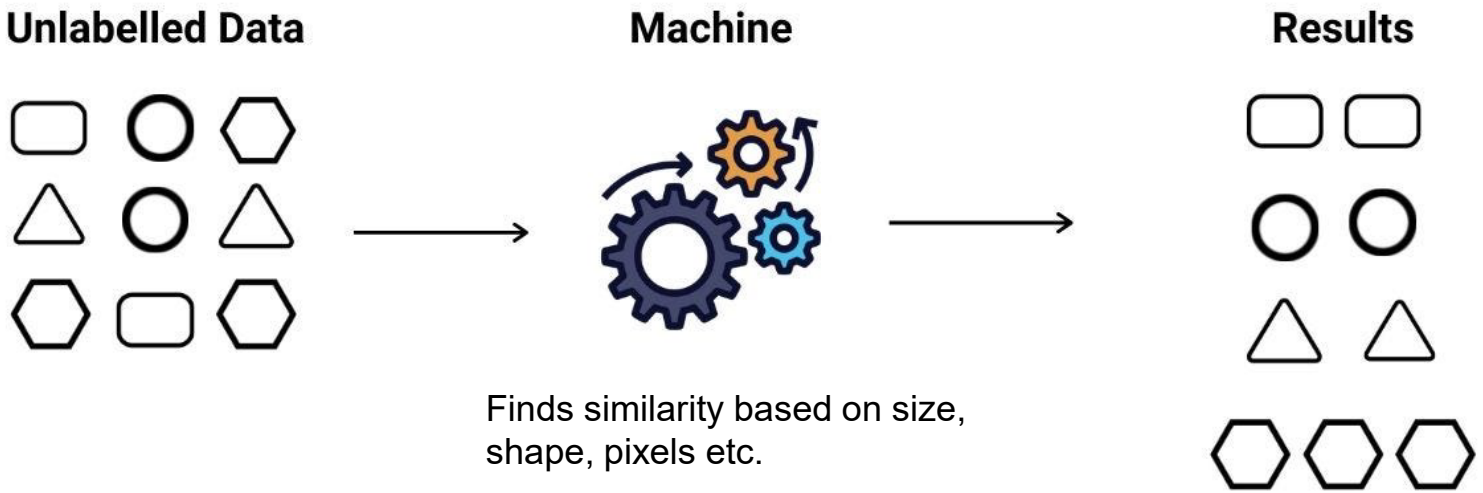
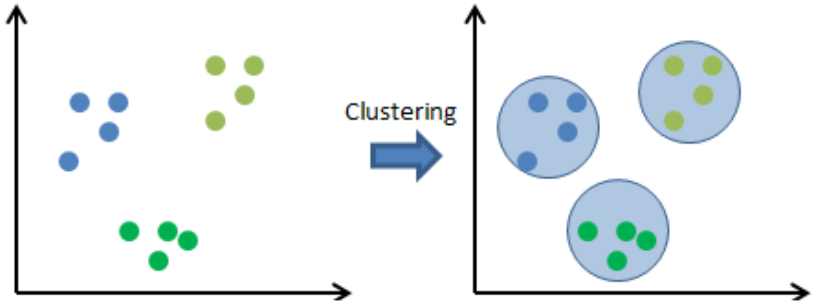


Image source: https://www.enjoyalgorithms.com/blogs/supervised-unsupervised-and-semisupervised-learning

An introduction to clustering

- Given a set of objects, the basic idea is to create subsets (clusters) that contain similar objects.

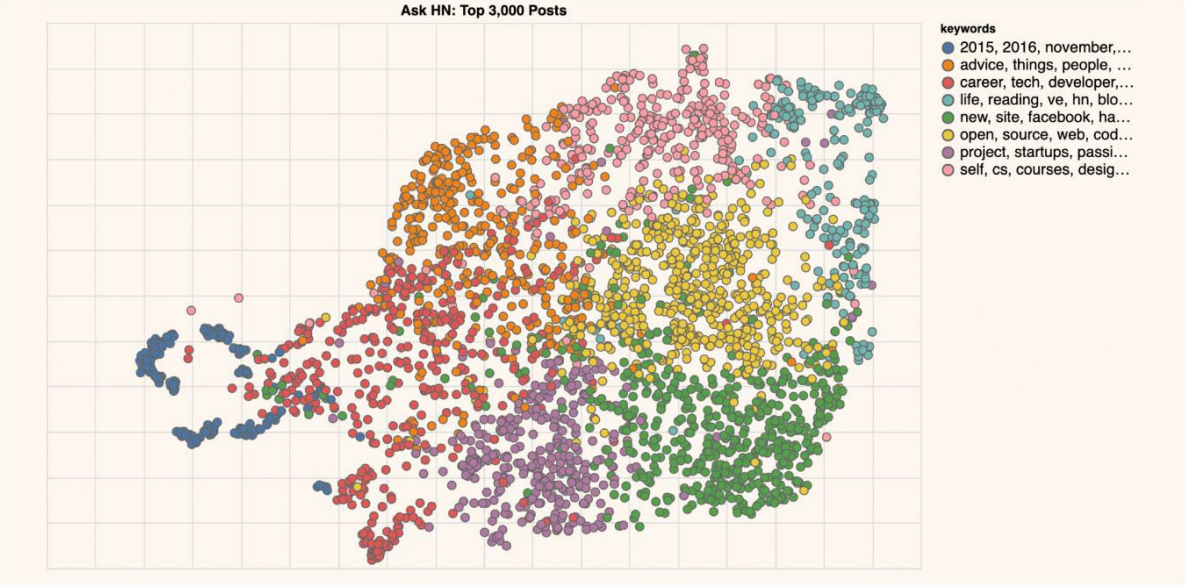


- Objects are grouped according to some distance measure
- Objects in a particular cluster are:
  - Similar to each other
  - Different from objects in other clusters

Looks easy?

Clustering in high-dimensional spaces look different:

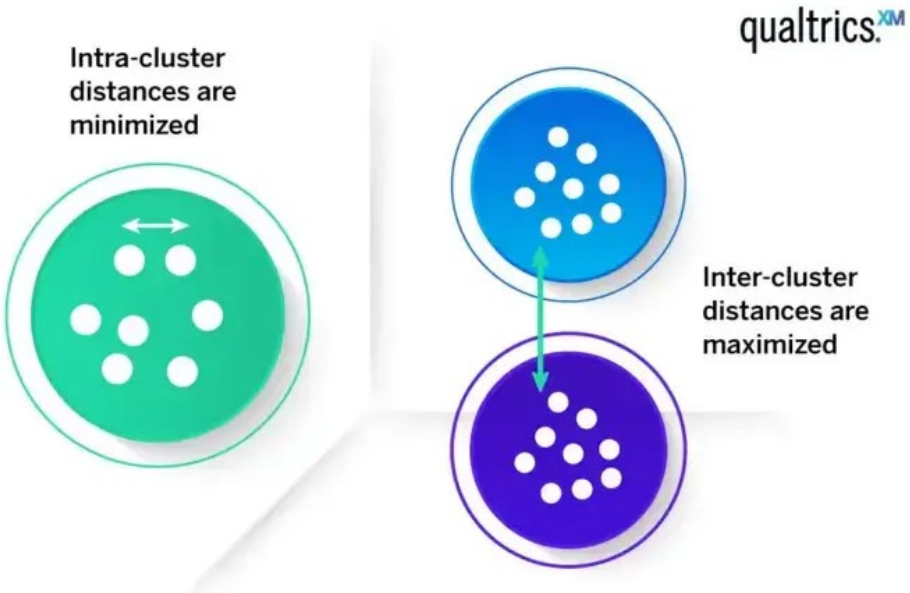
- Almost all pairs of points are at about the same distance



https://txt.cohere.ai/combing-for-insight-in-10-000-hacker-news-posts-with-text-clustering/

Additional reading: http://infolab.stanford.edu/~ullman/mmds/ch7.pdf

In high quality clustering



Intracluster distance is the distance between the data points inside the cluster. If there is a strong clustering effect present, this should be small (more homogenous)

Intercluster distance is the distance between data points in different clusters. Where strong clustering exists, these should be large (more heterogenous)

Source: https://www.qualtrics.com/experience-management/research/cluster-analysis/

- How clusters are formed depends on the criteria used for clustering. We'll look at 2 algorithms:

  Hierarchical clustering

  K-means clustering
- There are many, many more. Google them, explore them
- Clustering is an *unsupervised* learning method..

An introduction to clustering

- Many (but not all!) clustering algorithms are *distance-based*.

  Once we define a distance, the problem of clustering boils down to finding objects that are **close** to each other

- So, how do we define a distance between documents?

  A DTM essentially describes a “document space”

  The dimension of this space = number of distinct words

- Let’s look at a simple example consisting of 2 documents

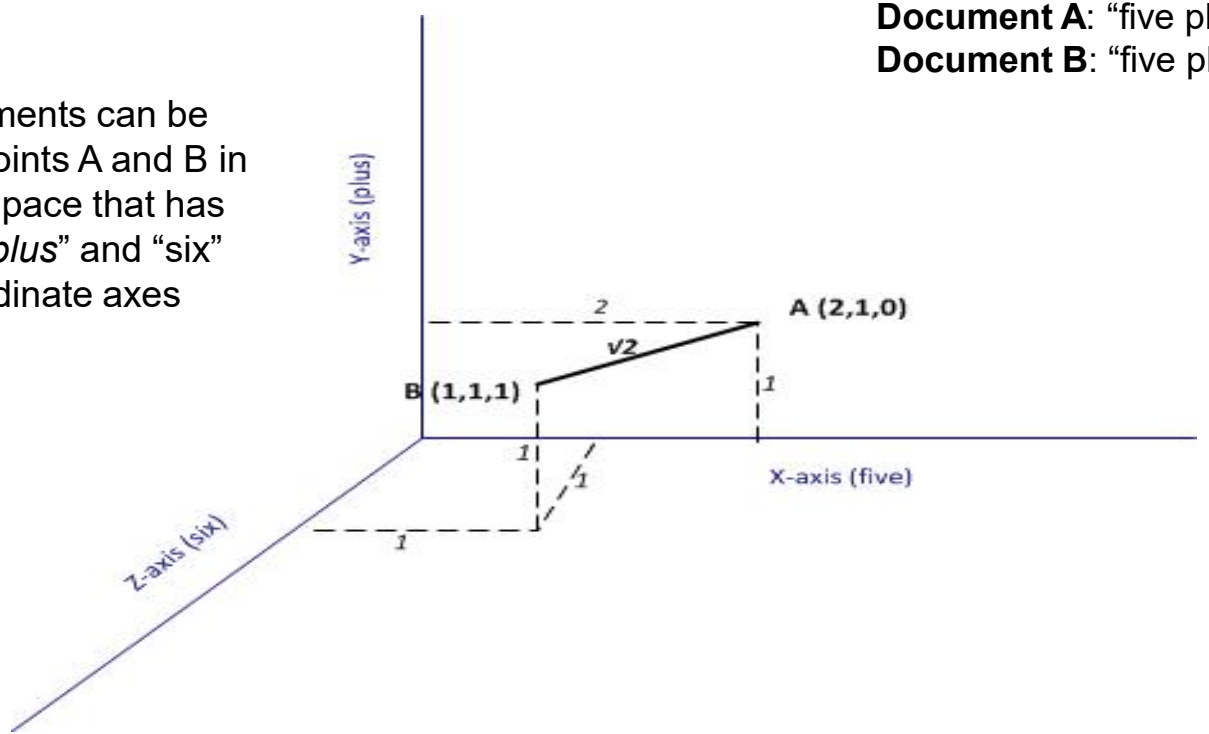
  Document A: “five plus five”

  Document B: “five plus six”

Euclidean distance

Document A: “five plus five”
Document B: “five plus six”

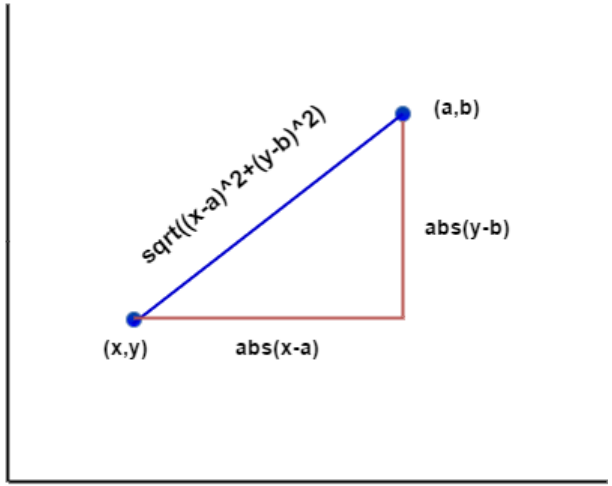
These two documents can be represented as points A and B in a 3 dimensional space that has the words “*five*” “*plus*” and “six” as the three coordinate axes



Euclidean vs manhattan distance

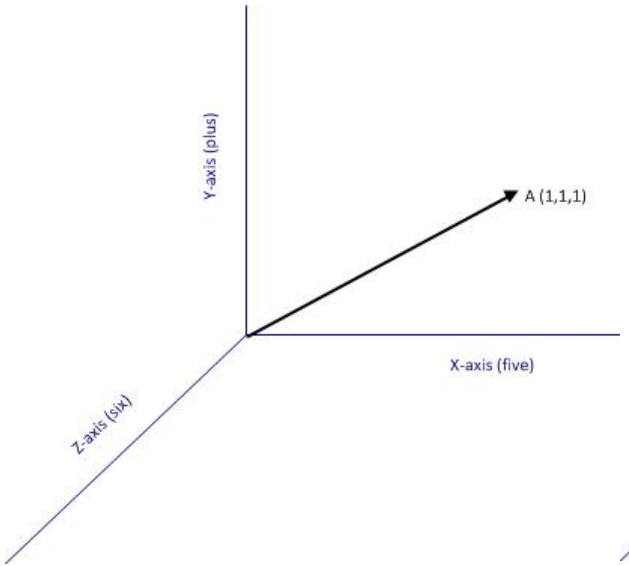
Euclidean distance = **sqrt((x-a)^2 + (y-b)^2)** *("As the crow flies" distance*)

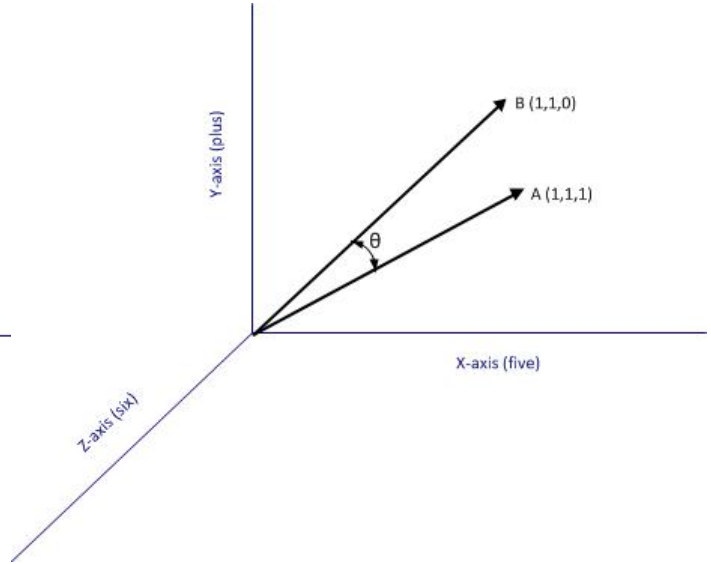
Manhattan distance = **|x-a| + |y-b|** (*Block distance*)



Cosine distance

Cosine **similarity** = cosine between vectors representing two data points





$$\cos(\theta) = \frac{x_1 x_2 + y_1 y_2 + z_1 z_2}{\sqrt{x_1^2 + y_1^2 + z_1^2}\sqrt{x_2^2 + y_2^2 + z_2^2}} \ldots (1)$$

Cos(***θ***) = 1, if ***θ*** =0 – parallel (similar) vectors

Cos(***θ***) = 0, if ***θ*** =90 – perpendicular (dissimilar) vectors

Cosine distance = 1 – Cos(*θ*)

Hierarchical clustering

- Two approaches:

  Agglomerative – start with each object in its own cluster and build clusters starting with objects that are closest

  Divisive – start with all objects in one cluster and split clusters recursively until each document is in its own cluster

  Clusters are then determined via comparing the **distances** at which merges or splits occur. Widely separated merges / splits are indicative of a cluster.

- We'll use the *hclust* algorithm which uses the agglomerative approach. Works as follows:

  1. Assign each document to its own (single member) cluster
  2. Find the pair of clusters that are closest to each other and merge them.
  3. Compute distances between the new cluster and each of the old clusters.
  4. Repeat steps 2 and 3 until you have a single cluster containing all documents.

Hierarchical clustering

- A note on clustering mechanism

How do we decide to merge clusters? This is defined by the *linkage criteria* which is determined by the *linkage function* which computes the distance between clusters as a function of the observations within any two pairs of clusters.

Some possible linkage functions:

The *single* linkage function will merge two clusters based on the closest two elements within both clusters

The *complete* linkage function will merge two clusters based on the farthest two elements within both clusters.

The ward linkage function is known as "**Ward's minimum variance method**" and works by minimizing the total within-cluster variance. The pair of clusters with the smallest cluster distance are then merged. (Generally, a **good go-to** and often produces more balanced clusters)

There are also average, weighted, centroid, median

```python
# Create TF-IDF vectorizer
vectorizer = TfidfVectorizer(stop_words='english')

# Convert documents to TF-IDF matrix
tfidf_matrix = vectorizer.fit_transform(documents)

# Convert sparse matrix to dense array
dense_tfidf_matrix = tfidf_matrix.toarray()

# Perform hierarchical clustering
linkage_matrix = linkage(dense_tfidf_matrix, method='ward')

# Plot dendrogram
plt.figure(figsize=(10, 7))
dendrogram(linkage_matrix)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Document Index')
plt.ylabel('Distance')
plt.show()

# Print cluster assignments
def print_clusters(linkage_matrix, n_clusters):
    from scipy.cluster.hierarchy import fcluster
    clusters = fcluster(linkage_matrix, n_clusters, criterion='maxclust')
    for i, cluster in enumerate(clusters):
        print(f"Document {i}: Cluster {cluster}")
    return clusters

# Choose the number of clusters (you can adjust this)
n_clusters = 3
clusters = print_clusters(linkage_matrix, n_clusters)
```

Visualising hierarchical clusters – cluster dendrogram

Document 0:"Machine learning is a subset of artificial intelligence"

Document 1: "Deep learning is a part of machine learning"

Document 2: "Neural networks are used in deep learning"

Document 3: "Artificial intelligence includes machine learning and deep learning"

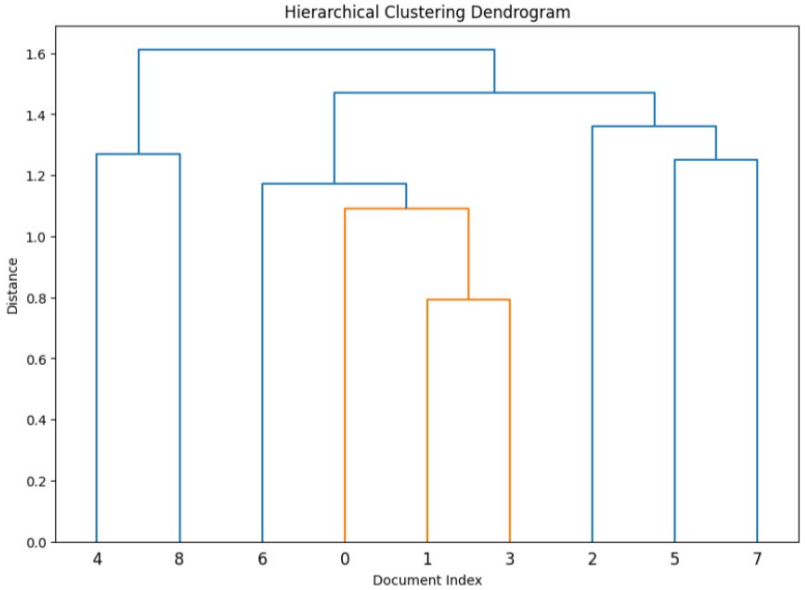
Document 4: "Natural language processing is an application of artificial intelligence"

Document 5: "Computer vision uses deep learning techniques"

Document 6: "Reinforcement learning is another branch of machine learning"

Document 7: "Data science often uses machine learning algorithms"

Document 8: "Artificial intelligence is transforming various industries"

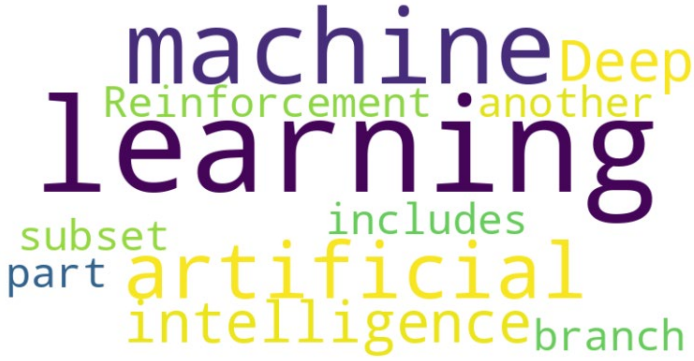



```
Document 0: Cluster 2
Document 1: Cluster 2
Document 2: Cluster 3
Document 3: Cluster 2
Document 4: Cluster 1
Document 5: Cluster 3
Document 6: Cluster 2
Document 7: Cluster 3
Document 8: Cluster 1
```

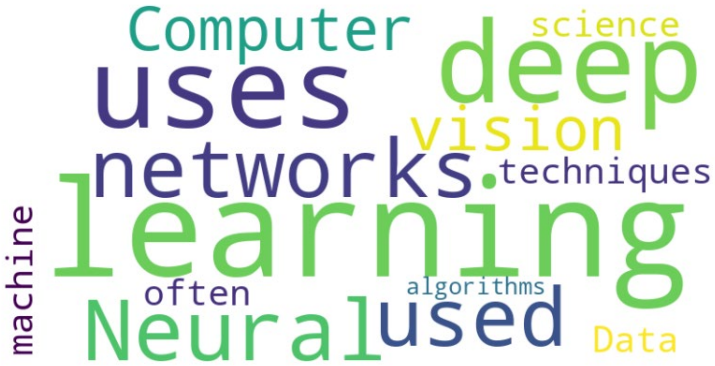
```
Document 0: Cluster 2
Document 1: Cluster 2
Document 2: Cluster 3
Document 3: Cluster 2
Document 4: Cluster 1
Document 5: Cluster 3
Document 6: Cluster 2
Document 7: Cluster 3
Document 8: Cluster 1
```

Word Cloud for Cluster 1

Word Cloud for Cluster 2



Word Cloud for Cluster 3



Other distances

Ward uses Euclidian distance measure by default.

If you're creating the linkage matrix from raw data (not a distance matrix), you can specify an alternate distance metric:

E.g. linkage_matrix = linkage(dense_tfidf_matrix, method='ward', metric='cosine’)

You can try other similarity measures and see the difference in clusters created:

Manhattan distance

Jaccard distance

…

Extension topic: K-Means clustering

- Generates a specified number (K) of clusters centred around the average (Mean) of each cluster. We'll use the *kmeans* algorithm.
- Works as follows:

1. Assign the documents randomly to k bins
2. Compute the location of the centroid (geometric center) of each bin.
3. Compute the distance between each document and each centroid
4. Assign each document to the bin corresponding to the centroid closest to it.
5. Stop if no document is moved to a new bin, else go to step 2.

- Limitations

  Local optimum – so try many different starting configs (controlled by *nstart* parameter)

  Difficult to figure out what K should be.

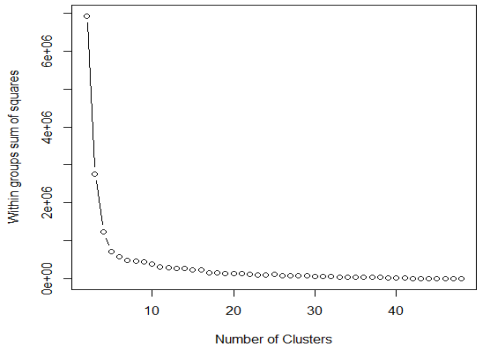
Choosing k

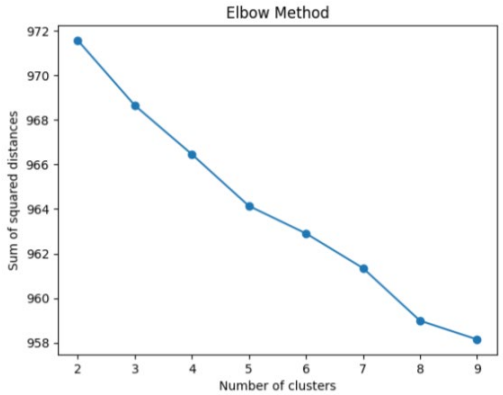
- No fail safe method to choose K. The elbow method is a heuristic approach based on the following intuition:

  The quantity that is minimised by kmeans is the total of the within-cluster sum of squares (WSS) between each point and the mean.

  One expects that this quantity will be maximum when k=1 and then will decrease as k increases, sharply at first and then less sharply as k reaches its optimal value.

  There should be an “elbow” in the plot of k vs WSS.



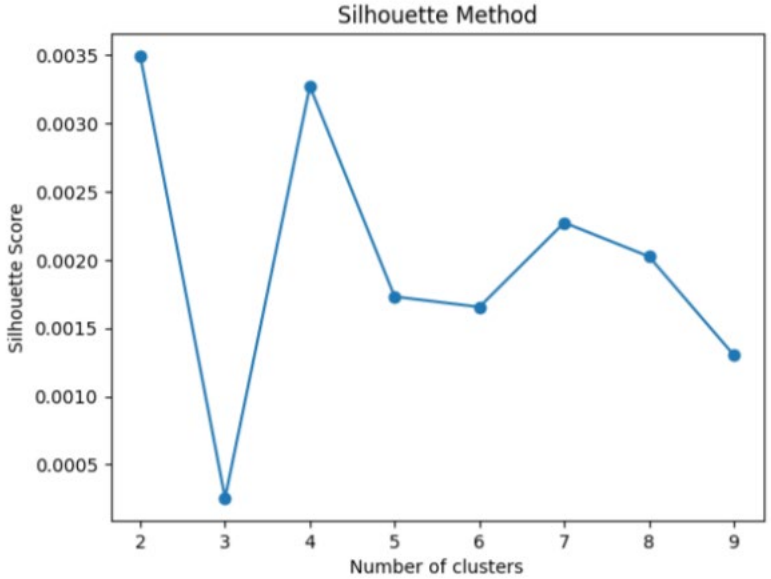


Choosing k Pt.2 (K-Means++)

- The traditional K-means algorithm can result in bad clustering based on poor choice of initial centroids.

  K-means++ places initial centroids far from each other to avoid the issue of poor initial centroid choice.

  Additionally, the algorithm is configured to run 10 times with different centroid seeds and the best cluster inertia is recorded.



Look for the peak in the plot, where the silhouette score is maximized. This point suggests the best separation between clusters.

Some things to experiment with

- Some things to try:

  Try different values of nstart and k, plot the results. Interpret them

  Find the optimal value of k using the elbow method. If there is no obvious optimum, make a reasonable guess.

  Do you have any ideas on how we can do better? (*Hint*: think about the potential effect of very long posts on distance measures)

  – Repeat exercise using different distance measures

- Ref: https://eight2late.wordpress.com/2015/07/22/a-gentle-introduction-to-cluster-analysis-using-r/

Some general points about Clustering

- *Unsupervised* learning method

  Predefined category labels are not required.

  In contrast, most of the other algorithms you'll learn in ML are *supervised* learning algorithms.

- Very general

  Can be applied to pretty much any kind of data (providing you can define a clustering criterion)

- Clustering criteria are not necessarily distance-based

Analysing Cluster ‘Quality’ - Crosstabs

- We will discuss later metrics to analyse classification tasks, but how do we say that the clustering was ‘good’?

  Depends on the business case. Remember that this is *unsupervised*

  Cross-tabulation allows for validation of the cluster labels aligning with the *known class labels*. Good clustering would see each column and each row distinctly separated.

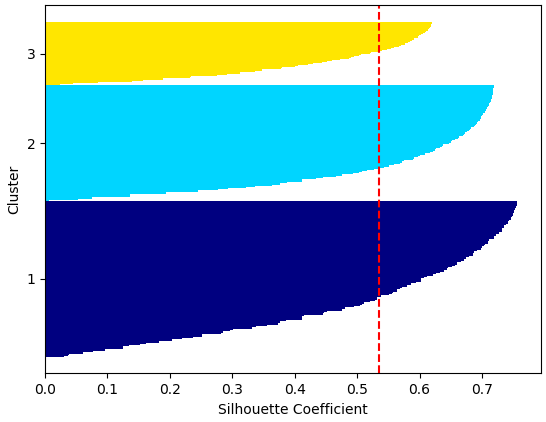
  Let us say that ’good’ would being able to separate ‘high value’ and ‘low value’ customers into distinct clusters. You have 2 clusters and 100 samples. The following crosstab would be a ‘perfect’ clustering.

	Value	
	High	Low
Cluster 1	50	0
Cluster 2	0	50

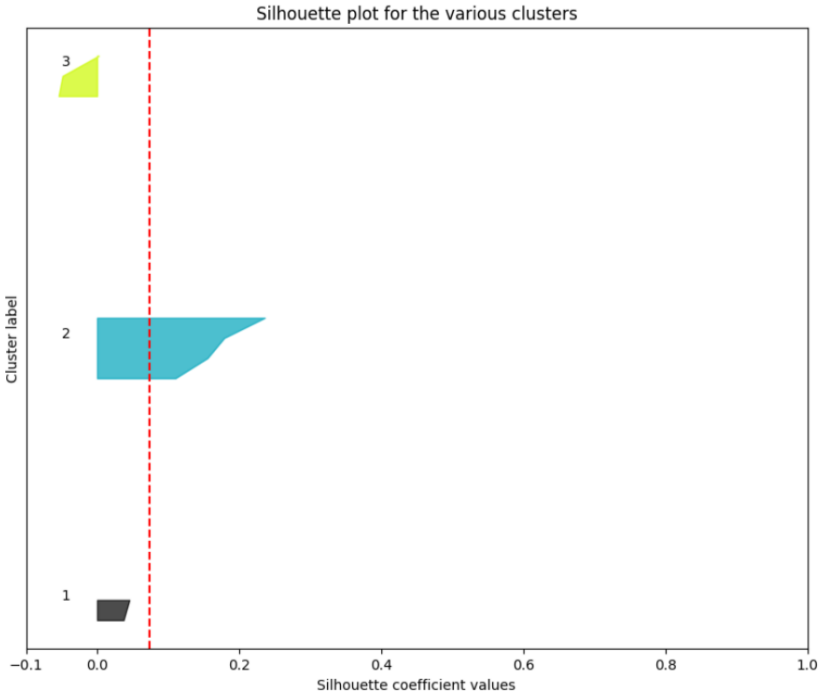
Analysing Cluster ‘Quality’ – Silhouette Coefficient

- Silhouette plots provide the opportunity for visual and numerical analysis

  Visually, the width (y-axis length) of each coloured spike indicates the size of the cluster. For a balanced dataset, good clustering would have these as roughly equal in width. In contrast, in an analysis in which the dataset is unbalanced, good clustering would have clusters of differing sizes.



Silhouette plot for our toy example



- The silhouette coefficient ranges from -1 to 1, where higher values indicate that the object is well-matched to its own cluster and poorly-matched to neighboring clusters
- A high average silhouette width indicates good clustering
- If many points have a negative silhouette, it may indicate too many or too few clusters

Analysing Cluster ‘Quality’ – Silhouette Coefficient

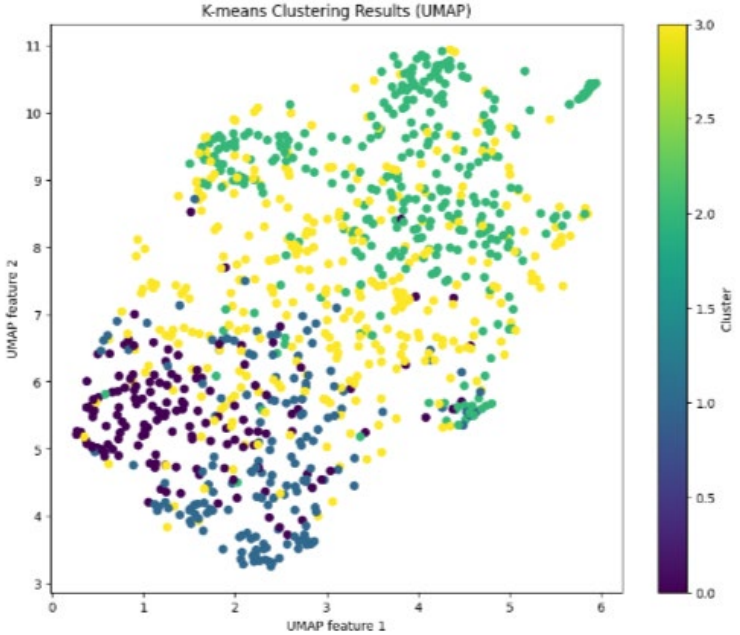
- Silhouette plots provide the opportunity for visual and numerical analysis

  Numerically, silhouette plots also provide the silhouette coefficient. This metric is on a scale of [-1,1] and provides a numeric way to analyse the how close each point in one cluster is to points in the neighbouring clusters. Negative silhouette coefficient scores are interpreted as ‘the observations within the cluster are probably in the wrong cluster’. Values closer to 1 indicate better clustering. A guide to interpreting different levels of this coefficient is provided in (source), page 88.

Silhouette Coefficient	Interpretation
0.71 - 1.00	A strong structure has been found
0.51-0.70	A reasonable structure has been found
0.26-0.5	The structure is weak and could be artificial; please try additional methods on this dataset
<= 0.25	No substantial structure has been found

Visualising clusters in high-dimensional spaces

Might require dimensionality reduction E.g. Umap, t-SNE



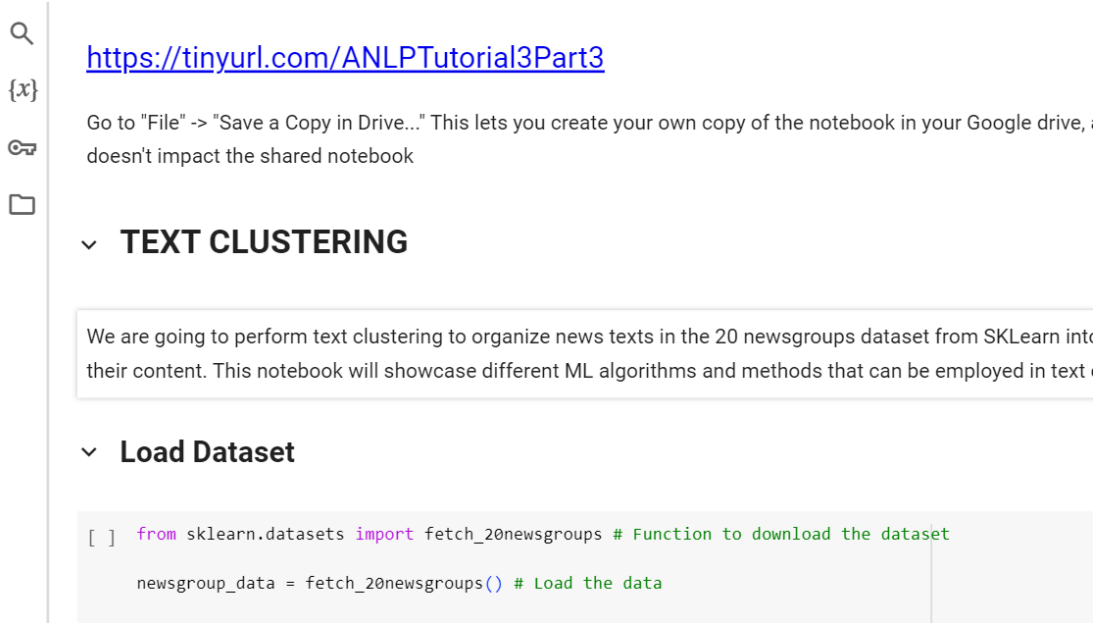
Clustering

Can you think of any areas where clustering would be useful?

Tutorial

Open Colab notebook:

tinyurl.com/ANLPColab2Part3



General Q & A