Chair of Process and Data Science
RWTH AACHEN UNIVERSITY

Elements of Machine Learning & Data Science

Text Mining

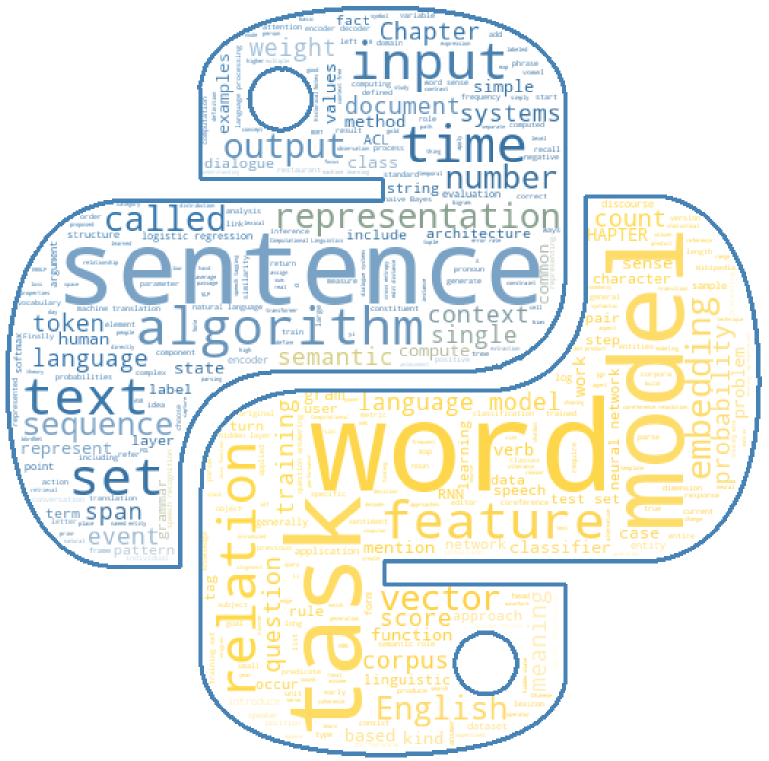
Lecture 21

Prof. Wil van der Aalst

Marco Pegoraro, M.Sc.
Christian Rennert, M.Sc.

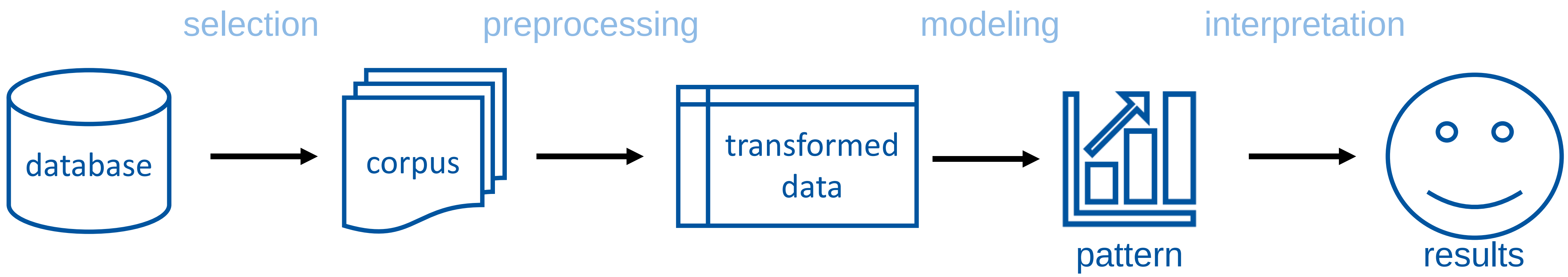
Text Mining

1. **Introduction to Text Mining**
2. Text Preprocessing
3. Modeling
4. Enriching Text: Linked Data
5. N-gram Model
6. Learning a Representation
7. Word2vec

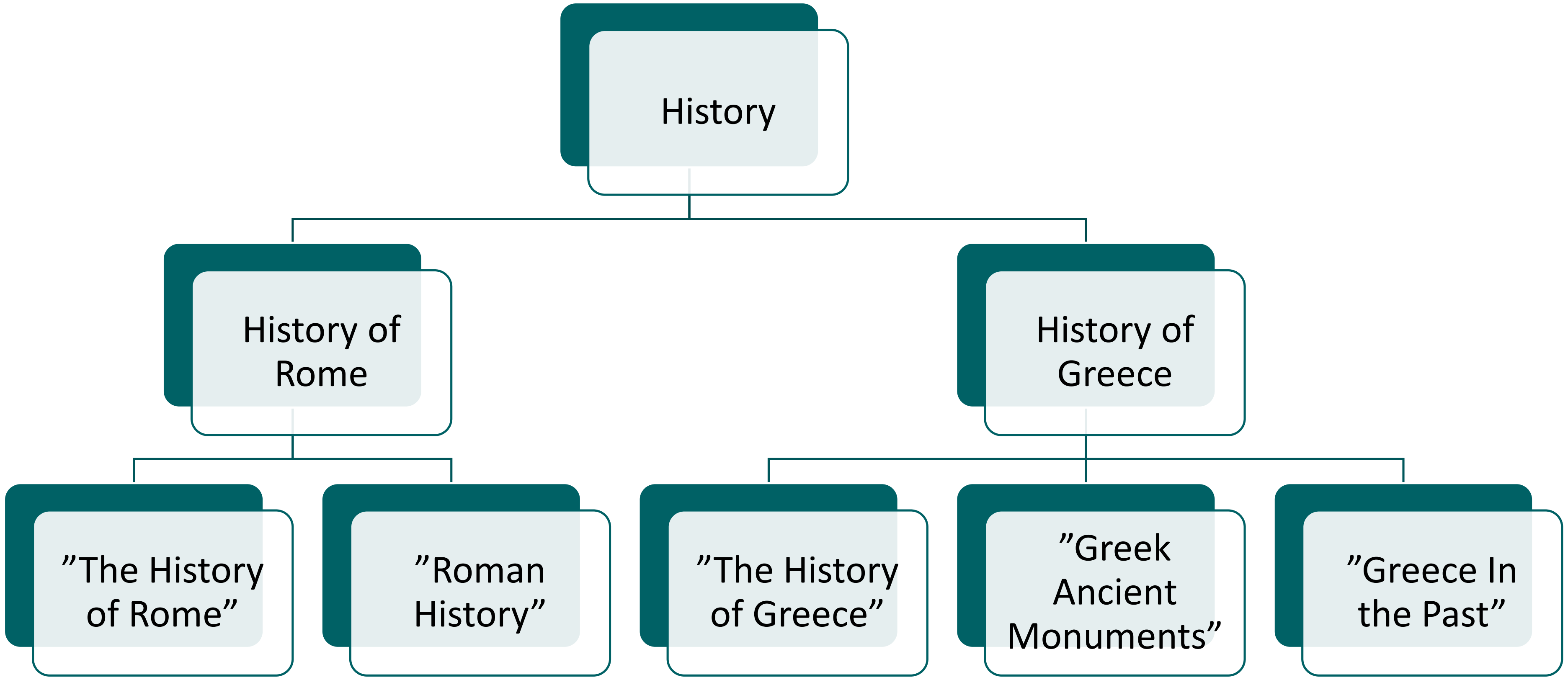


Text Mining Pipeline

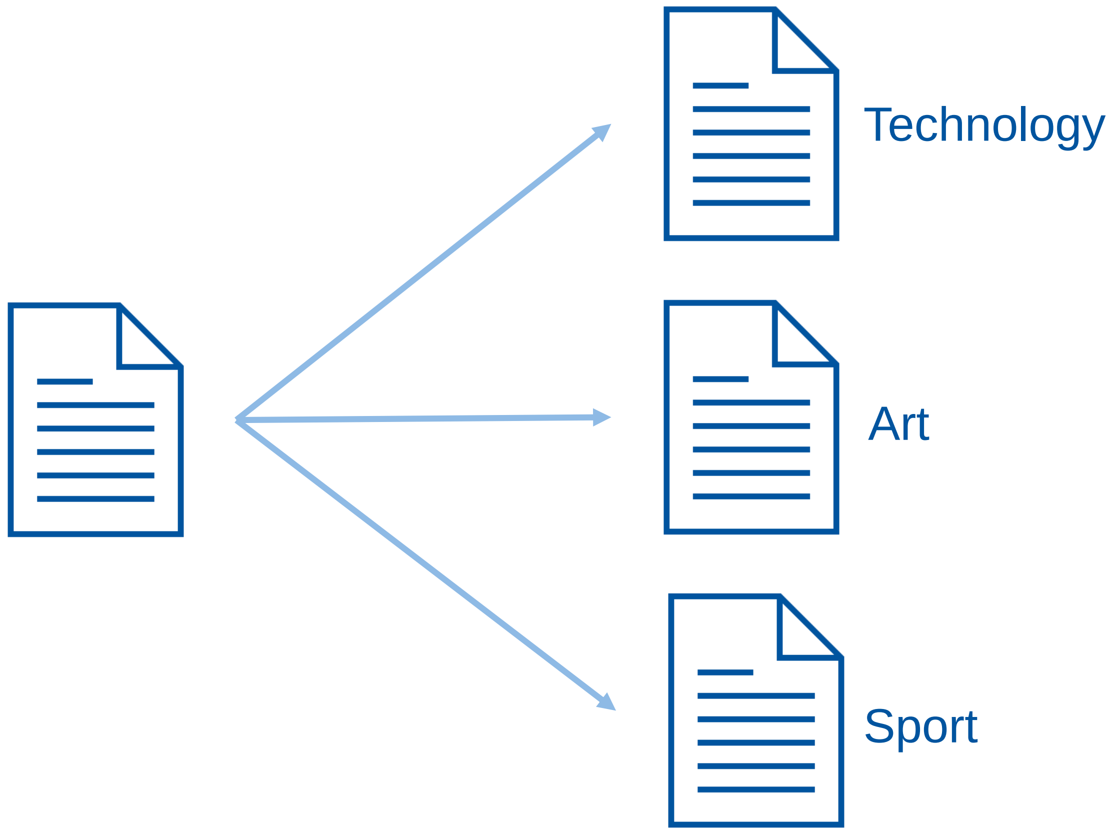
Text mining is the extraction of (structured) knowledge from (unstructured) text



Text Mining Applications – Document Clustering



Text Mining Applications – Document Classification



Text Mining Applications – Named Entity Recognition

Named Entity Recognition (NER)

- recognizes named entities in a text
- classifies them according to their contextual information

”I was at this beach”, said Jenny. She had been traveling through Florida before. ”I’m sure that Max would love to visit Miami as well”

person

place

Text Mining Applications – Sentiment Analysis

Sentiment Analysis:

Identifies positive/negative attitudes through recognition of emotions, opinions or moods

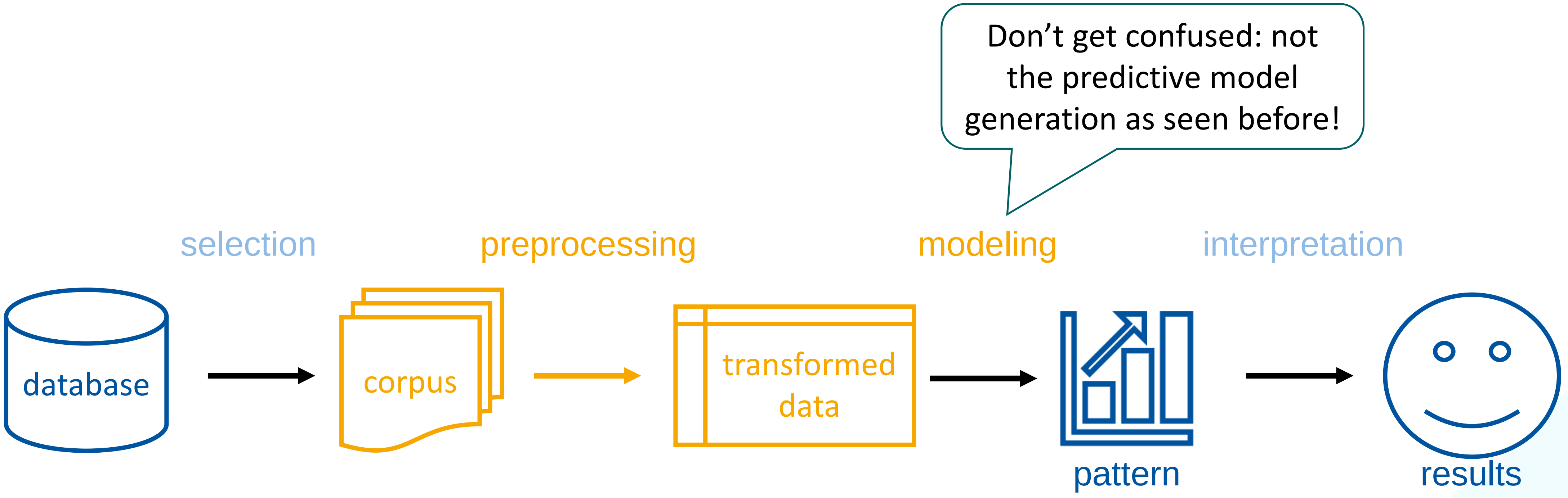
Trust	”Amazing news for Bitcoin!”
Fear	”Suspicious assault in Berlin”
Surprise	”What happened with the President of the US?”

Text Mining Applications – Other Examples

- Part-Of-Speech (POS) tagging – label words with their corresponding part of a speech (noun, verb, adjective, etc.)
- Coreference resolution – identification of words and expressions that refer to the same entity in a text (Max – he, Jenny – her, etc.)
- Keyword extraction – automatic identification of significant terms in a text
- Machine translation – automatic translation from one language to another

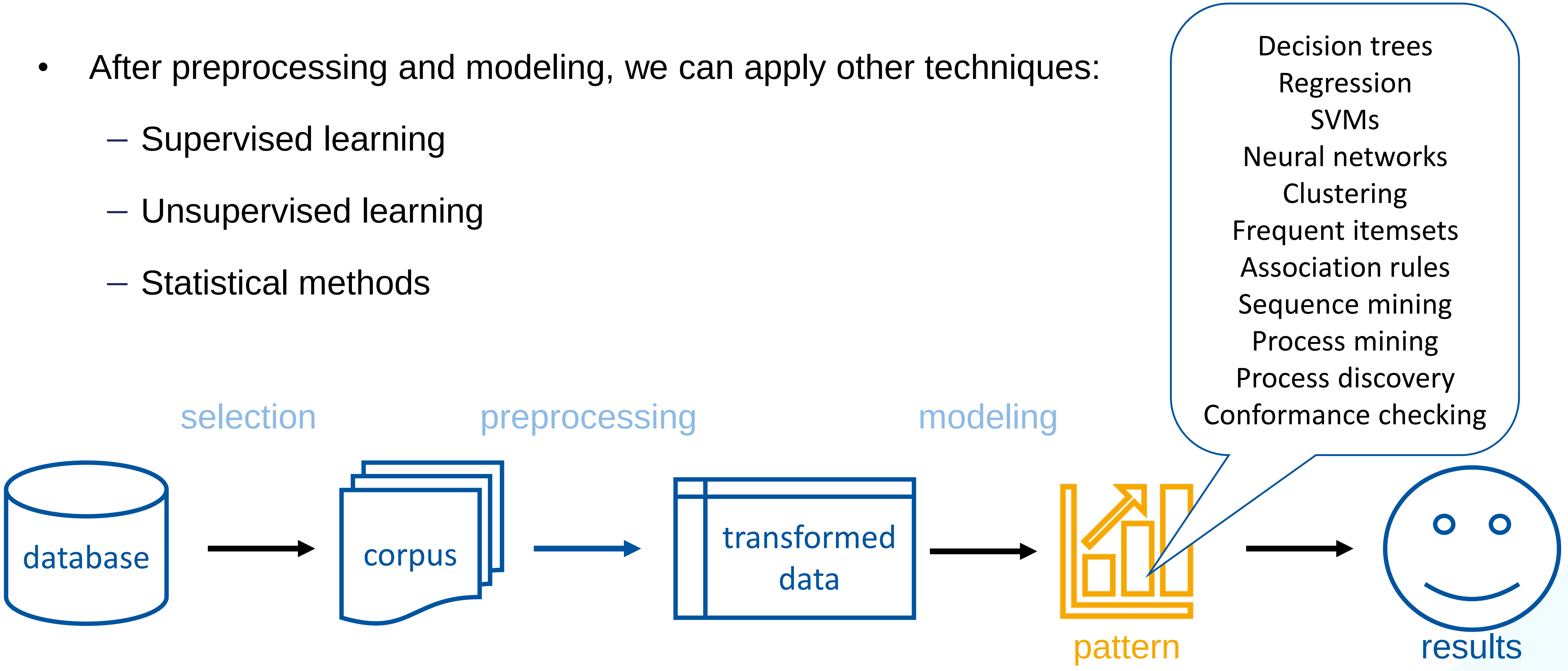
Important steps

It's crucial to preprocess and model the text properly



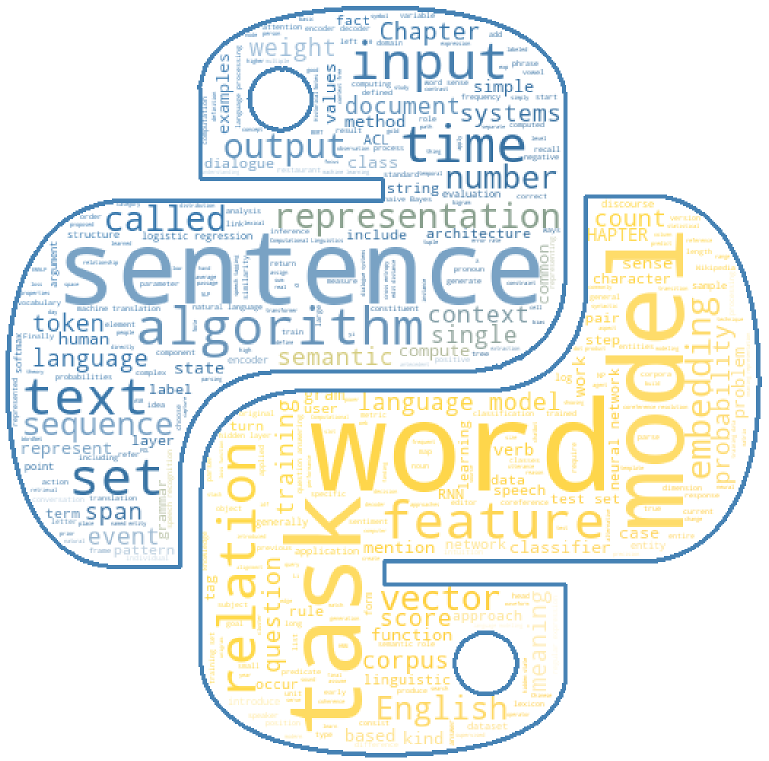
Important steps

- After preprocessing and modeling, we can apply other techniques:
  - Supervised learning
  - Unsupervised learning
  - Statistical methods



Text Mining

1. Introduction to Text Mining
2. **Text Preprocessing**
3. Modeling
4. Enriching Text: Linked Data
5. N-gram Model
6. Learning a Representation
7. Word2vec



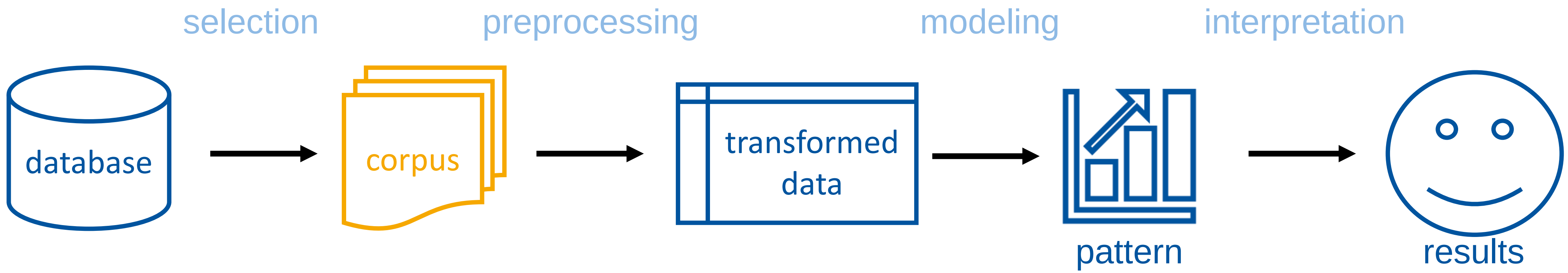
Structuring Text

- Challenge: from unstructured data (text) to structured data (ideally numbers)

- Texts are usually unstructured:
  - Do we consider sentences or words?
  - Lengths of single units of information vary.
  - What are text features and instances?

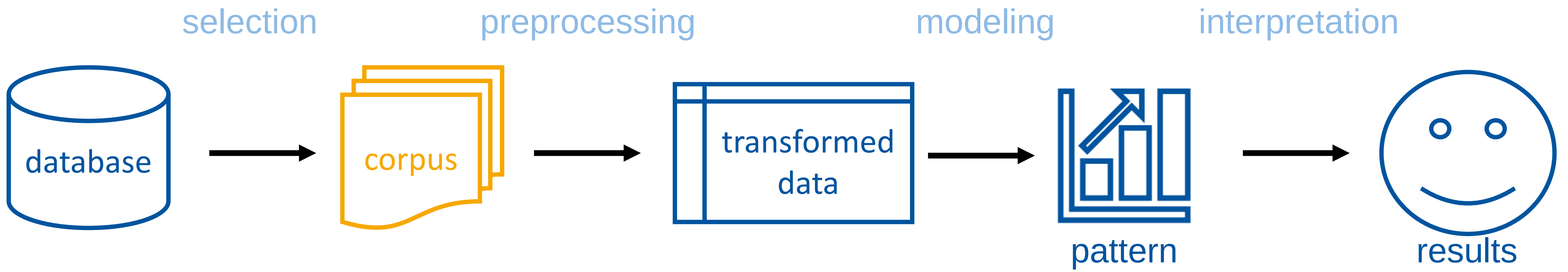
Corpus

Text mining pipeline – Step 1: Extract a corpus



Corpus

Text mining pipeline – Step 1: Extract a corpus



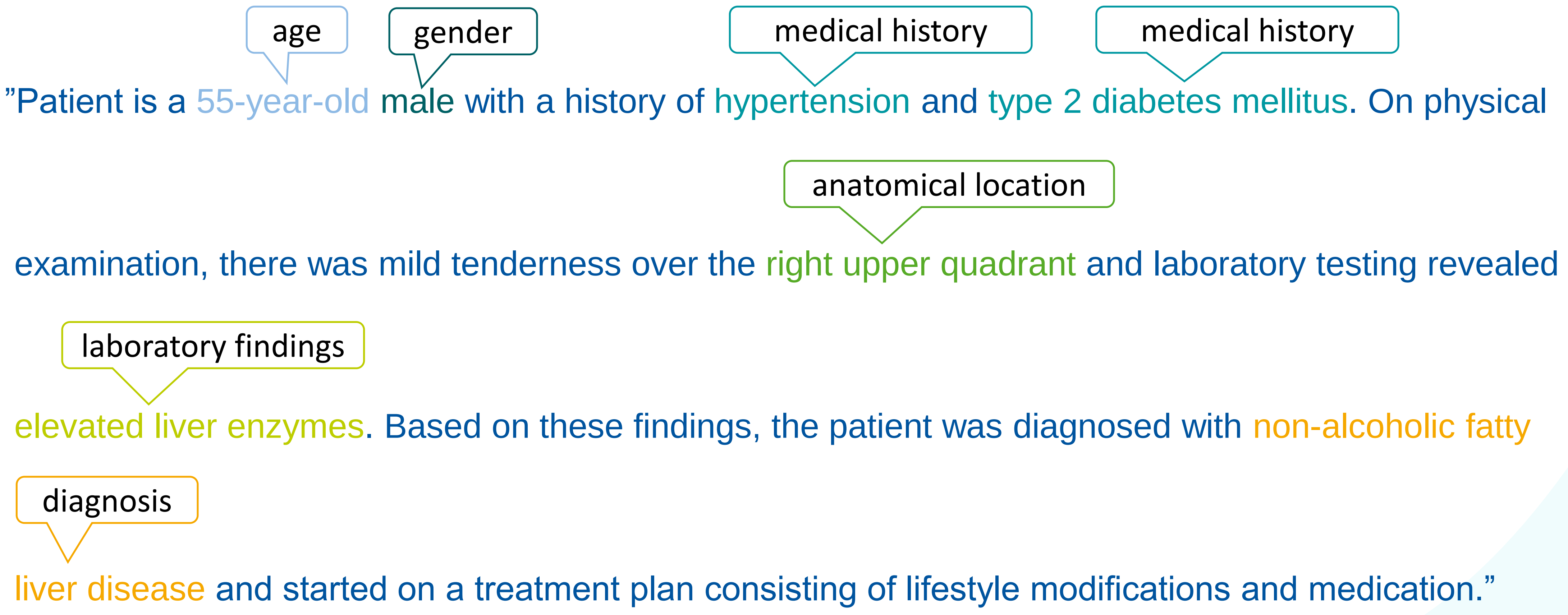
- A corpus is a collection of pieces of a text
- Pieces can be words, sentences, paragraphs, tweets, posts, …
- Pieces in the corpus are often called documents (regardless of their nature and size)

Annotated Corpus

- Annotated corpus: units (or fractions of units) of a text are annotated with additional information in order to work as a training set for a specific application
- Corpora are usually annotated by hand

Annotated Corpus

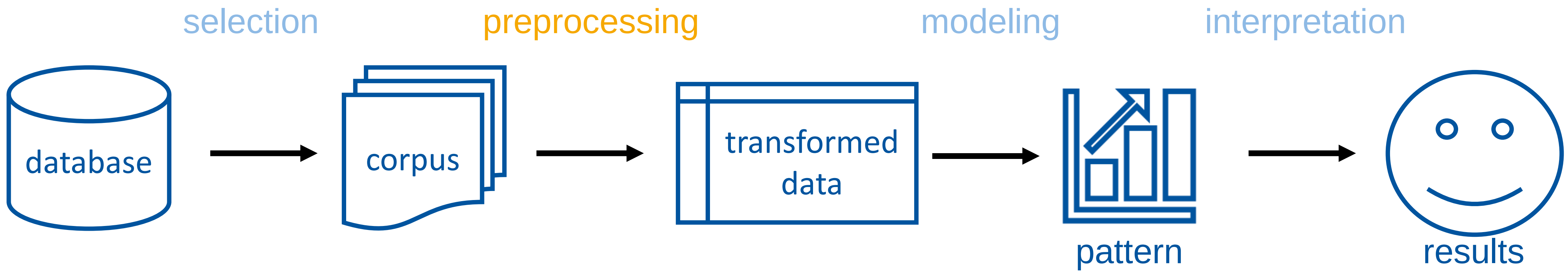
for domain-specific (medical) named entity recognition



”Patient is a 55-year-old male with a history of hypertension and type 2 diabetes mellitus. On physical examination, there was mild tenderness over the right upper quadrant and laboratory testing revealed elevated liver enzymes. Based on these findings, the patient was diagnosed with non-alcoholic fatty liver disease and started on a treatment plan consisting of lifestyle modifications and medication.”

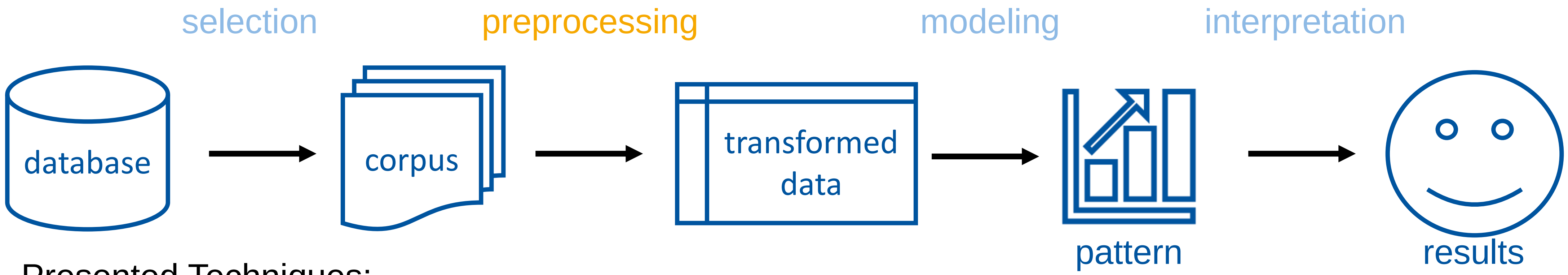
Preprocessing

Text mining pipeline – Step 2: the corpus goes through the preprocessing



Preprocessing

Text mining pipeline – Step 2: the corpus goes through the preprocessing



Presented Techniques:

- Tokenization
- Stop word removal
- Token normalization: stemming or lemmatization

Tokenization

- Splitting the text into smaller units: tokens
- Usually tokens are words (word tokenization)
- Could also be characters, ideograms, phonemes, syllables, sentences, phrases, clauses, …

Word Tokenization

Easy! Just split after spaces, right?

“He’s been talking to Bill de Blasio, the 109th New York City mayor.”

‘He’s’, ‘been’, ‘talking’ ‘to’ ‘Bill’ ‘de’ ‘Blasio,’, ‘the’, ‘109th’, ‘New’, ‘York’, ‘City’, ‘mayor.’

Word Tokenization

Not always that easy…

“He’s been talking to Bill de Blasio, the 109th New York City mayor.”

This one token should actually be 2

Should there be 3 tokens or 1?

3 tokens or 1?

‘He’s’, ‘been’, ‘talking’ ‘to’ ‘Bill’ ‘de’ ‘Blasio,’, ‘the’, ‘109th’, ‘New’, ‘York’, ‘City’, ‘mayor.’

Shouldn’t be there

Shouldn’t be there

Does this mean something? Should we have a token for each number?

Tokenization Is Not Trivial

- Other languages differ from English and can have 40- or even 60-letter words that are a combination of a few single words
- **Example**: German

  Is this one token?

$$\underbrace{\text{Arbeiter}}_{\text{employee}}\overbrace{\text{unfall}}^{\text{accident}}\underbrace{\text{versicherung}}_{\text{insurance}}\text{s}\overbrace{\text{gesetz}}^{\text{law}}$$

Tokenization Is Not Trivial

- Other languages differ from English and can have 40- or even 60-letter words that are a combination of a few single words

- There exists specific software to tokenize
- Usually tokenization has to be specifically designed ad-hoc for a certain task
- Tokenization is application-dependent

Stop Word Removal

- Removing words that are not informative
- A stop list commonly contains:
  - ”to be” and ”to have” verbs: is, are, am, was, have, has, ...
  - articles: the, a, an, …
  - auxiliary verbs: will, should, would, shall, must, …
  - prepositions: in, to, from, of, by, on, …
  - interrogative words: who, what, which, where, when, how, why, …
- Stop lists are language-dependent

Stop Word Removal

Why do we remove stop words?

What happens if we compare the following two sentences by counting the number of identical words?

“The cats, which were seven, started to climb the tree.”

“The Saxons, which were outnumbered, started to prepare the siege.”

Stop Word Removal

Why do we remove stop words?

6/10 words are the same! → These sentences should be very similar.

"The cats, which were seven, started to climb the tree."

"The Saxons, which were outnumbered, started to prepare the siege."

Stop Word Removal

Why do we remove stop words?

However, 5 out of 6 common words are common stop words…

"The cats, which were seven, started to climb the tree."

"The Saxons, which were outnumbered, started to prepare the siege."

Stop Word Removal

Why do we remove stop words?

- After removing stop words, sentences are not so similar anymore
- We went from 60% overlap to 10% overlap

"~~The~~ cats, ~~which~~ ~~were~~ seven, started ~~to~~ climb ~~the~~ tree."

"~~The~~ Saxons, ~~which~~ ~~were~~ outnumbered, started ~~to~~ prepare ~~the~~ siege."

Stop Word Removal

- Be careful about what you remove!
- Assuming our previous stop list:

~~The Who’s~~ seventh studio album ~~is~~ titled ”~~The Who by~~ Numbers”

$\rightarrow$

seventh studio album titled Numbers

Stop Word Removal

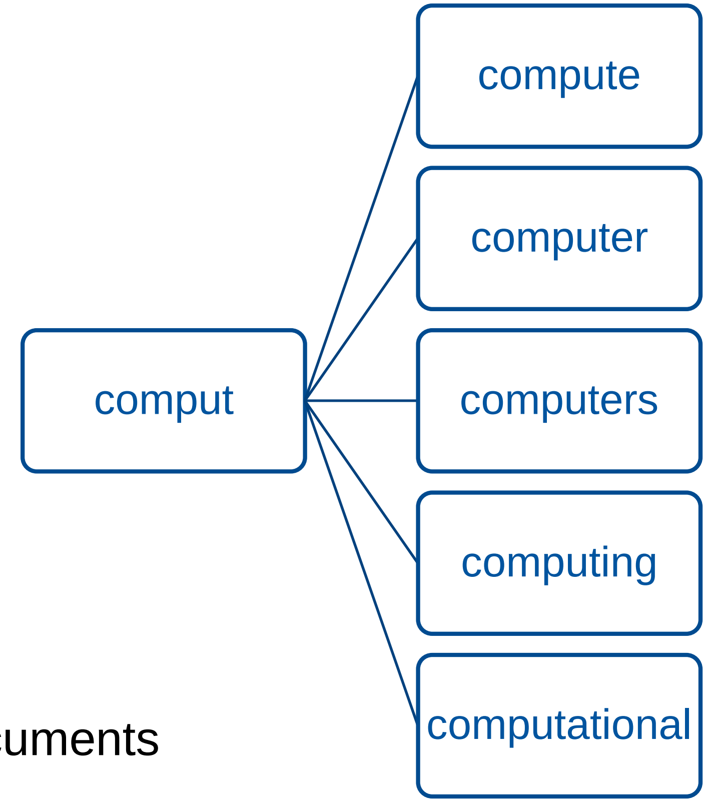
- Designing a good stop list is not a trivial task
- Deciding which words should be stop words depends on the context/goal
- Stop lists can be domain-dependent
  - Example: in a healthcare domain, ”patient” and ”hospital” could be stop words

Token Normalization

- Goal: Transforming tokens to make them comparable
- **Main forms:**
  - Stemming
  - Lemmatization
- Other forms:
  - Case-folding – converting everything into lowercase
  - Alternative spelling (color/colour)
  - Transliterations
    - ä → ae, ö → oe

Stemming

- Reducing words to their word stem (base or root form)
- Works *most* of the time
- A stem does not need to be a word
- Usually a word stem carries the meaning



- Stemming algorithms can be aggressive or conservative
  - Aggressive stemmers focus on similarities between documents
  - Conservative stemmers carefully consider differences

Stemming - Example

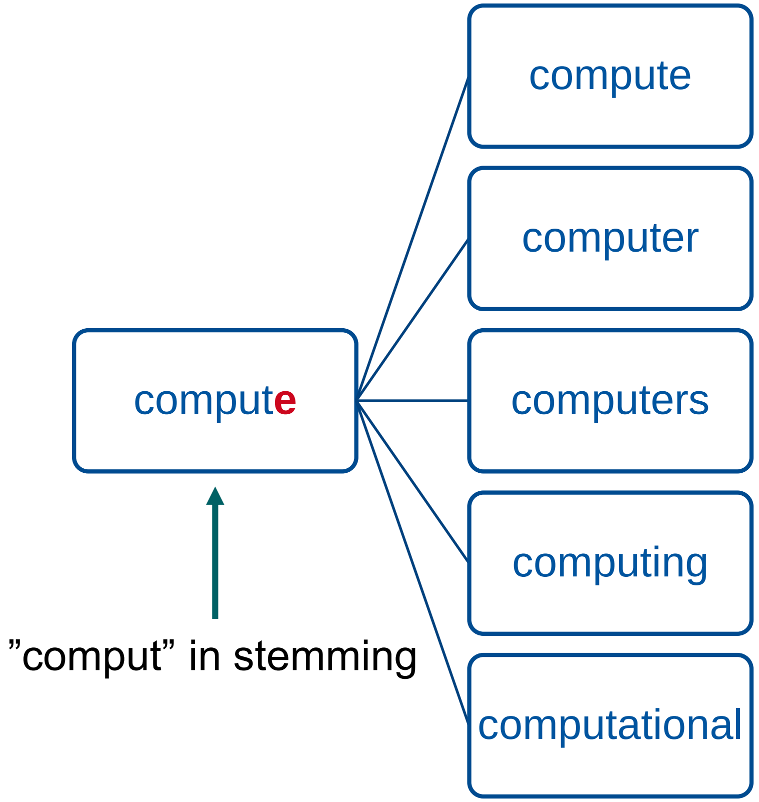
” As I walked through the dark forest, the rustling of leaves under my feet echoed through the silent night. I could feel the chill of the autumn air seeping through my jacket, and the moonlight shone down in a pale glow. Suddenly, a twig snapped, and I froze. Was someone or something watching me? I held my breath, waiting for a sign, but all I heard was the sound of my own heartbeat, thumping in my ears. Slowly, I continued on, my nerves on edge, unsure of what lays ahead.”

$\longrightarrow$

as i walk through the dark forest the rustl of leav under my feet echo through the silent night i could feel the chill of the autumn air seep through my jacket and the moonlight shone down in a pale glow sudden a twig snap and i froze was someon or someth watch me i held my breath wait for a sign but all i heard was the sound of my own heartbeat thump in my ear slowli i continu on my nerv on edg unsur of what lay ahead

Lemmatization

- Lemmatization applies vocabulary and morphological analysis
- Goal:
  - To remove inflectional endings only
  - To return the base or dictionary form of a word
  - This form is known as the lemma
- More sophisticated than stemming
- Often rule-based (every language has exceptions)



Lemmatization - Example

” As I walked through the dark forest, the rustling of leaves under my feet echoed through the silent night. I could feel the chill of the autumn air seeping through my jacket, and the moonlight shone down in a pale glow. Suddenly, a twig snapped, and I froze. Was someone or something watching me? I held my breath, waiting for a sign, but all I heard was the sound of my own heartbeat, thumping in my ears. Slowly, I continued on, my nerves on edge, unsure of what lay ahead.”

$\longrightarrow$

as I walk through the dark forest the rustle of leaf under my foot echo through the silent night I could feel the chill of the autumn air seep through my jacket and the moonlight shine down in a pale glow suddenly a twig snap and I freeze be someone or something watch me I hold my breath wait for a sign but all I hear be the sound of my own heartbeat thump in my ear slowly I continue on my nerve on edge unsure of what lay ahead

Stemming vs Lemmatization

as i walk through the dark forest the rustl of leav under my feet echo through the silent night i could feel the chill of the autumn air seep through my jacket and the moonlight shone down in a pale glow sudden a twig snap and i froze was someon or someth watch me i held my breath wait for a sign but all i heard was the sound of my own heartbeat thump in my ear slowli i continu on my nerv on edg unsur of what lay ahead

as I walk through the dark forest the rustle of leaf under my foot echo through the silent night I could feel the chill of the autumn air seep through my jacket and the moonlight shine down in a pale glow suddenly a twig snap and I freeze be someone or something watch me I hold my breath wait for a sign but all I hear be the sound of my own heartbeat thump in my ear slowly I continue on my nerve on edge unsure of what lay ahead

Other Challenges

- Prepositional attachment (e.g., what does a prepositional phrase "with" refer to)

  "eating spaghetti **with** chopsticks" vs "eating spaghetti **with** meatballs"

- Anaphora resolution (an expression whose interpretation depends on another expression)

  "He convinced his roommate to buy a TV for **himself**"

- Other hidden information

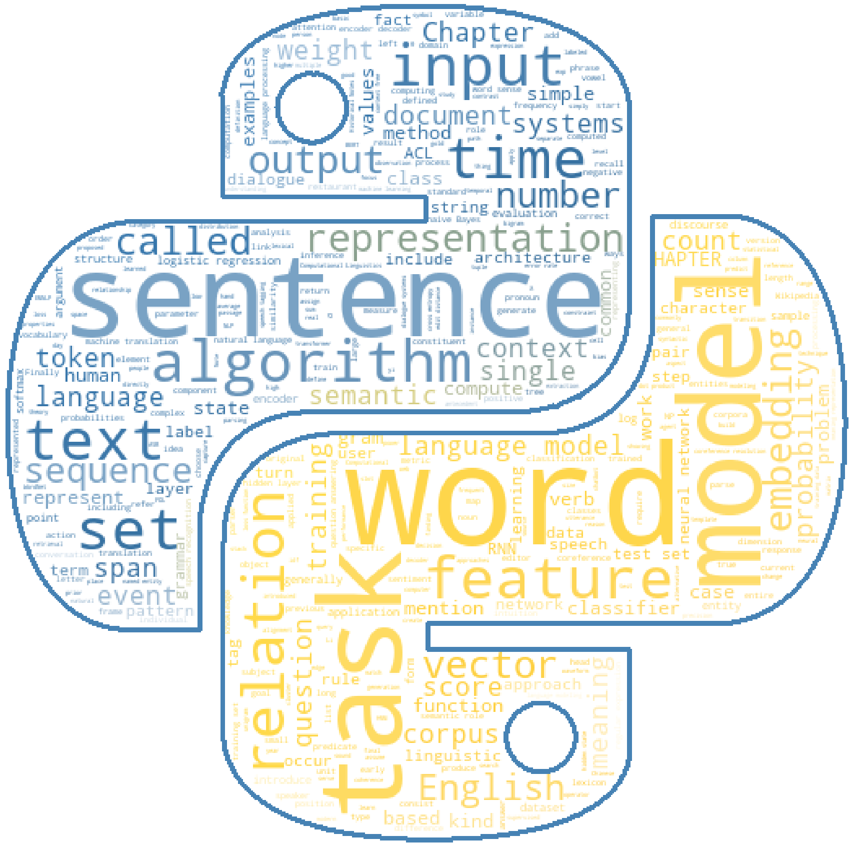
  "He quit smoking" → he used to smoke

- Homonyms (same word can have different meanings, be a noun or verb, etc.)

  sign, firm, tie, watch, tear, bark…

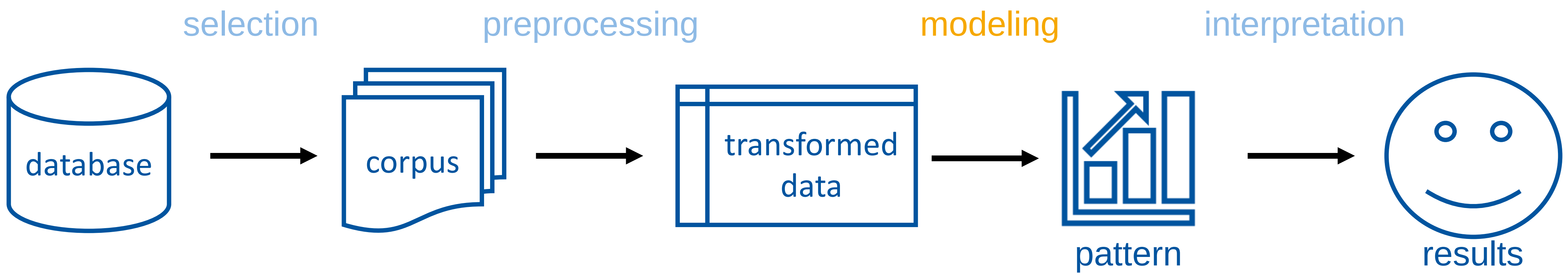
Text Mining

1. Introduction to Text Mining
2. Text Preprocessing
3. **Modeling**
4. Enriching Text: Linked Data
5. N-gram Model
6. Learning a Representation
7. Word2vec



Modeling

After preprocessing the data, we can focus on building a usable model (i.e., representation)



Bag of Words Model

- Bag of Words (BoW) is the simplest model
- It represents each document *d* in the corpus as a bag of words (a multisets of words)

Bag of Words Model – Examples

"Process discovery and conformance checking are part of process mining."

(apply stemming and stop word removal)

['process' ^{2}, 'discover' ^{1}, 'conform' ^{1}, 'check' ^{1}, 'part' ^{1}, 'min' ^{1}]

Bag of Words Model – Examples

“Process discovery and conformance checking are part of process mining.”

[‘process’^{2}, ‘discover’^{1}, ‘conform’^{1}, ‘check’^{1}, ‘part’^{1}, ‘min’^{1}]

”I love to eat pizza. Pizza is my favorite food. When I want to eat, I always choose pizza.”

[‘pizza’^{3}, ‘eat’^{2}, ‘love’^{1}, …]

Bag of Words Model

- A naïve model - multiset representation loses the order of the items:

“James loves watching movies, Kate hates it.”

“Kate loves watching movies, James hates it.”

[‘James’, ‘love’, ‘watch’, ‘movie’, ‘Kate’, ‘hate’]

“Chickens hatch from eggs.”

“Eggs hatch from chickens.”

[‘chicken’, ‘hatch’, ‘egg’]

$\rightarrow$ BoW can be unsuitable for applications that strongly depend on word order

Bag of Words Model

Why is it still useful?

- Easy to implement
- Works well enough in many applications
- Was successfully used in the past
  (e.g., for spam detection, information retrieval)
- Today, it’s often used as an intermediate step (feature extraction) in combination with more advanced techniques

Document-Term Matrix

Tabular form for text data representation – Bag of Words in matrix form

Document	**love**	**eat**	**pizza**	**favorite**	**food**
Doc1	1	2	2	1	1
Doc2	0	1	2	0	0
Doc3	1	1	1	0	1

Frequency of 'love' in Doc3

Term Frequency (TF)

- A document-term matrix is a mathematical matrix that contains frequencies of the terms (words) found in documents
- These frequencies are called term frequencies (tf)

$$tf(w, d) = \text{number of occurences of word } w \text{ in a document } d$$

Term Frequency (TF) - Example

$$tf(w, d) = \text{number of occurences of word } w \text{ in a document } d$$

$d =$ ”I love to eat pizza. Pizza is my favorite food. When I want to eat, I always choose pizza. Pizza toppings can vary, but my go-to toppings are pepperoni and mushrooms. I could eat pizza every day and I would never get tired of it.”

$$\begin{aligned} tf(\text{pizza, } d) &= 5 \\ tf(\text{I, } d) &= 5 \\ tf(\text{eat, } d) &= 3 \\ tf(\text{to, } d) &= 2 \\ tf(\text{toppings, } d) &= 2 \end{aligned}$$

$$\begin{aligned} tf(\text{my, } d) &= 2 \\ tf(\text{and, } d) &= 2 \\ tf(\text{mushrooms, } d) &= 1 \\ &\ldots \end{aligned}$$

Term Frequency (TF) - Example

$$tf(w,d) = \text{number of occurences of word } w \text{ in a document } d$$

$d =$ ”I love to eat pizza. Pizza is my favorite food. When I want to eat, I always choose pizza. Pizza toppings can vary, but my go-to toppings are pepperoni and mushrooms. I could eat pizza every day and I would never get tired of it.”

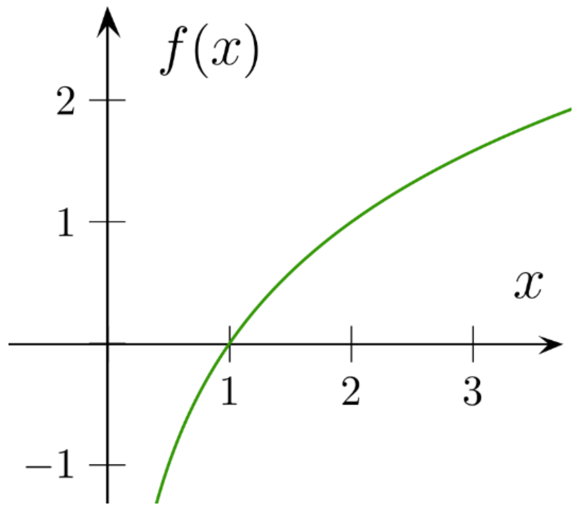
What is the topic of the document?

$$tf(\text{pizza},\ d) = 5$$
$$tf(\text{I},\ d) = 5$$
$$tf(\text{eat},\ d) = 3$$
$$tf(\text{to},\ d) = 2$$
$$tf(\text{toppings},\ d) = 2$$

$$tf(\text{my},\ d) = 2$$
$$tf(\text{and},\ d) = 2$$
$$tf(\text{mushrooms},\ d) = 1$$
$$\ldots$$

Inverse Document Frequency (IDF)

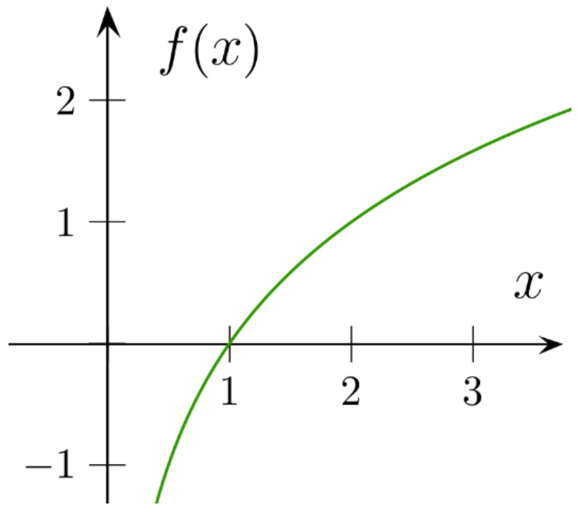
- IDF reflects how ”special” the word is in the corpus c in terms of its frequency
- Intuition: The more unlikely the word is, the higher the value



$$idf(w,c) = log_2\left(\frac{|c|}{\text{number of documents in } c \text{ that contain } w \text{ at least once}}\right)$$

Inverse Document Frequency (IDF) - Example

$$idf(w,c) = log_2\left(\frac{|c|}{\text{number of documents in } c \text{ that contain } w \text{ at least once}}\right)$$



Consider an imaginary corpus of 100 recipes from a cookbook c:

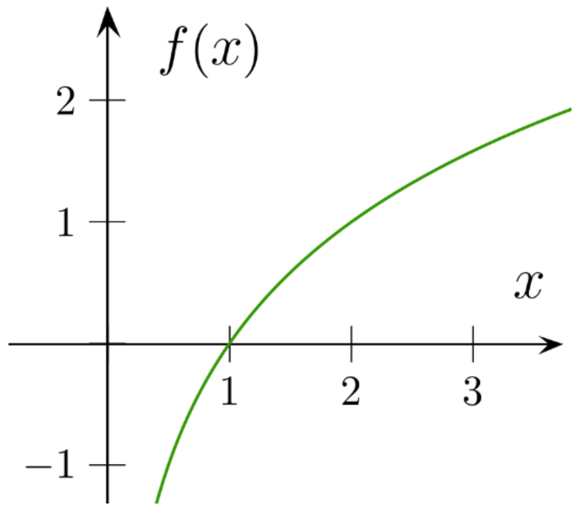
$$idf(\text{water}, c) = log_2\left(\frac{100}{92}\right) = 0.12$$

$$idf(\text{flour}, c) = log_2\left(\frac{100}{60}\right) = 0.74$$

$$idf(\text{strawberries}, c) = log_2\left(\frac{100}{15}\right) = 2.74$$

Inverse Document Frequency (IDF)

- Words with low idf score appear in many documents
- These words are not useful for distinguishing between the documents



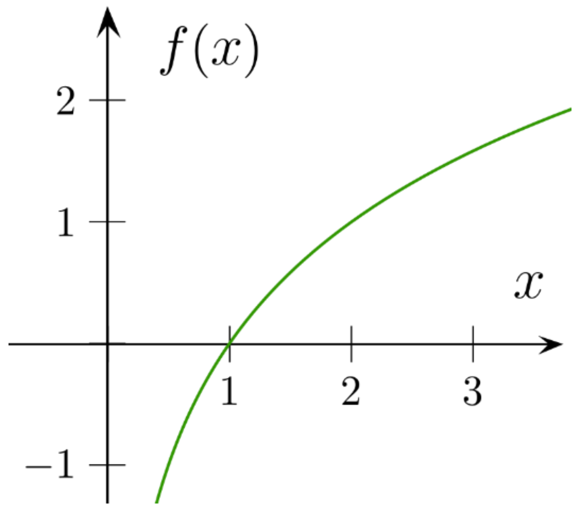
$$idf(\text{water}, c) = log_2(\frac{100}{92}) = 0.12$$

$$idf(\text{flour}, c) = log_2(\frac{100}{60}) = 0.74$$

$$idf(\text{strawberries}, c) = log_2(\frac{100}{15}) = 2.74$$

Inverse Document Frequency (IDF)

- Words with low idf score appear in many documents
- These words are not useful for distinguishing between the documents
- Low-scored words can be stop word candidates (add to a stop list)



$$idf(\text{water}, c) = log_2(\tfrac{100}{92}) = 0.12$$

$$idf(\text{flour}, c) = log_2(\tfrac{100}{60}) = 0.74$$

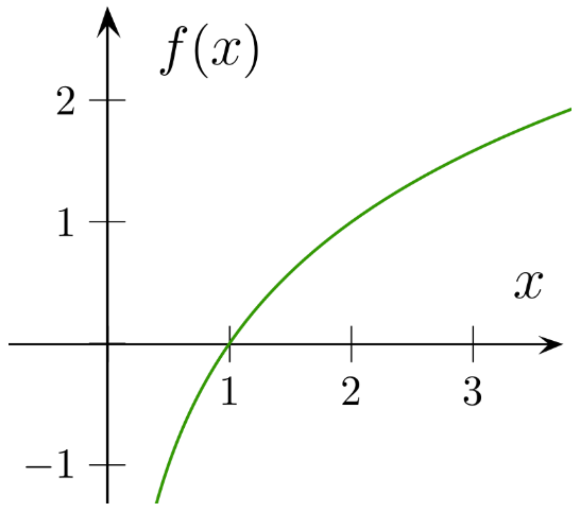
$$idf(\text{strawberries}, c) = log_2(\tfrac{100}{15}) = 2.74$$

TF-IDF Scoring

- Combination of the tf and idf scoring
  - tf: strength of the association between a word and a document
  - idf: the relevance of a word in a whole corpus (how “special“ it is)

$$tfidf(w,d,c) = tf(w,d) \cdot idf(w,c)$$

- Essential in information retrieval



idf(w, c) < 1: w occurs in more than half of the documents
idf(w, c) > 1: w occurs in less than half of the documents

TF-IDF Scoring - Example

Four documents d_1, d_2, d_3 and d_4 in corpus c

$d_1 =$ ‘Cats are the only pet of the felines family, while dogs are canids.’

stem: feline

$d_2 =$ ‘Cats are the third-most popular pet in the US.’

$d_3 =$ ‘Dogs have been selected for millennia as pet animals.’

$d_4 =$ ‘Normally, dogs are not aggressive towards other dogs outside their territory.’

$$tf(\text{feline}, d_1) = 1$$
$$tf(\text{feline}, d_2) = 0$$
$$tf(\text{feline}, d_3) = 0$$
$$tf(\text{feline}, d_4) = 0$$

$$idf(\text{feline}, c) = log_2(4/1) = 2$$

$$tfidf(\text{feline}, d_1, c) = 1 \cdot 2 = 2$$
$$tfidf(\text{feline}, d_2, c) = 0 \cdot 2 = 0$$
$$tfidf(\text{feline}, d_3, c) = 0 \cdot 2 = 0$$
$$tfidf(\text{feline}, d_4, c) = 0 \cdot 2 = 0$$

TF-IDF Scoring - Example

Four documents d_1, d_2, d_3 and d_4 in corpus c

stem: cat

$d_1 =$ ‘Cats are the only pet of the felines family, while dogs are canids.’

$d_2 =$ ‘Cats are the third-most popular pet in the US.’

$d_3 =$ ‘Dogs have been selected for millennia as pet animals.’

$d_4 =$ ‘Normally, dogs are not aggressive towards other dogs outside their territory.’

$$tf(\text{cat}, d_1) = 1$$
$$tf(\text{cat}, d_2) = 1$$
$$tf(\text{cat}, d_3) = 0$$
$$tf(\text{cat}, d_4) = 0$$

$$idf(\text{cat}, c) = log_2(4/2) = 1$$

$$tfidf(\text{cat}, d_1, c) = 1 \cdot 1 = 1$$
$$tfidf(\text{cat}, d_2, c) = 1 \cdot 1 = 1$$
$$tfidf(\text{cat}, d_3, c) = 0 \cdot 1 = 0$$
$$tfidf(\text{cat}, d_4, c) = 0 \cdot 1 = 0$$

TF-IDF Scoring - Example

Four documents d_1, d_2, d_3 and d_4 in corpus c

stem: dog

$d_1 =$ 'Cats are the only pet of the felines family, while dogs are canids.'

$d_2 =$ 'Cats are the third-most popular pet in the US.'

$d_3 =$ 'Dogs have been selected for millennia as pet animals.'

$d_4 =$ 'Normally, dogs are not aggressive towards other dogs outside their territory.'

$$tf(\text{dog}, d_1) = 1$$
$$tf(\text{dog}, d_2) = 0$$
$$tf(\text{dog}, d_3) = 1$$
$$tf(\text{dog}, d_4) = 2$$

$$idf(\text{dog}, c) = log_2(4/3) = 0.42$$

$$tfidf(\text{dog}, d_1, c) = 1 \cdot 0.42 = 0.42$$
$$tfidf(\text{dog}, d_2, c) = 0 \cdot 0.42 = 0$$
$$tfidf(\text{dog}, d_3, c) = 1 \cdot 0.42 = 0.42$$
$$tfidf(\text{dog}, d_4, c) = 2 \cdot 0.42 = 0.84$$

TF-IDF Scoring

- Many querying systems rely on TF-IDF scoring (or a variation)
- Simple algorithm:
  - Input: a query (a set of words) and a corpus c
  - For each document *d* in the corpus compute

    $$score(\text{query}, d, c) = \sum_{w \in \text{query}} tfidf(w, d, c)$$

    $$tf(w, d) \cdot idf(w, c)$$
  - Rank documents by their scores
  - Return first *x* documents

Document-Term Matrix with TF-IDF

- The document-term matrix can also be built with TF-IDF scores

each column is a word/term

	word1	**word2**	**word3**	**word4**	**…**
doc1					
doc2					
…					

each row is a document

each cell contains the TF-IDF score

- Tabular data with instances (documents) and features (words) – other features can be added
- The matrix allows to apply a wide range of data science techniques

Document Classification

Document	**love**	**eat**	**pizza**	**favorite**	**food**	**Article class**
Doc1	1	2	2	1	1	recipe
Doc2	0	1	2	0	0	restaurant menu
Doc3	1	1	1	0	1	cookbook

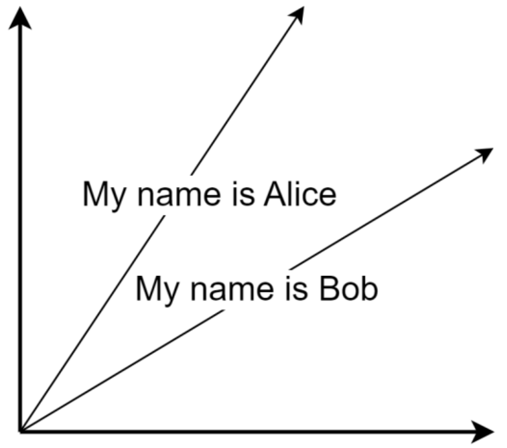
Every document is represented by a vector of a constant length (term frequencies or TF-IDF scores)

Target label for classification (e.g., to train a neural network)

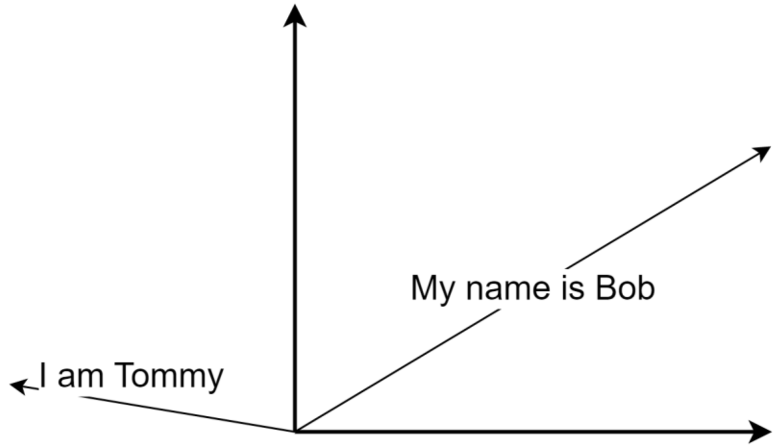
Document Clustering

- Having a fixed length vector, we need a distance/similarity measure to perform clustering
- Recall the Clustering lecture– we can use cosine similarity

$$sim(\mathbf{x_i}, \mathbf{x_j}) = \frac{\mathbf{x_i} \cdot \mathbf{x_j}}{||\mathbf{x_i}|| \, ||\mathbf{x_j}||} = cos(angle)$$



Similar vectors
Positive, approaches 1.0



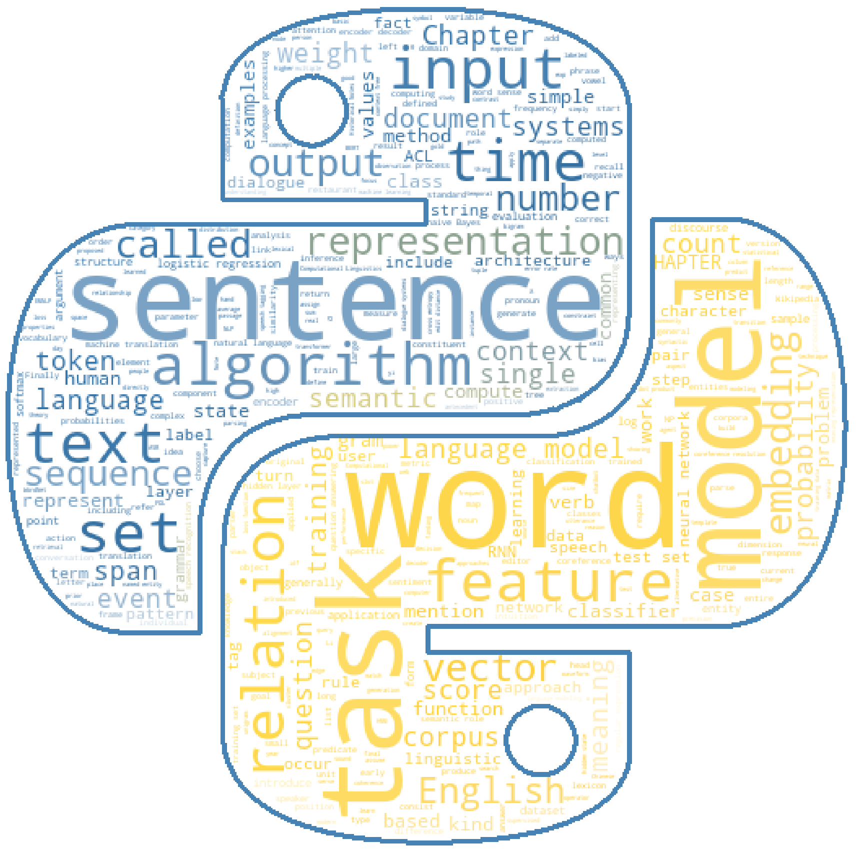
Different vectors
Negative, approaches -1.0

Document Clustering

- Cosine similarity is well-suited to compute for sparse vectors or vectors with different lengths (but in principle other metrics are possible)
- With a distance metric for text data, we can perform clustering (e.g., K-means, K-medoids, DBSCAN, etc.)

Text Mining

1. Introduction to Text Mining
2. Text Preprocessing
3. Modeling
4. **Enriching Text: Linked Data**
5. N-gram Model
6. Learning a Representation
7. Word2vec



Information Structure

- Representing text using a matrix makes it ”processable”, but connections between concepts and their meaning are missing

Information Structure

- Representing text using a matrix makes it ”processable”, but connections between concepts and their meaning are missing

- Solution: databases that are able to store connections between terms

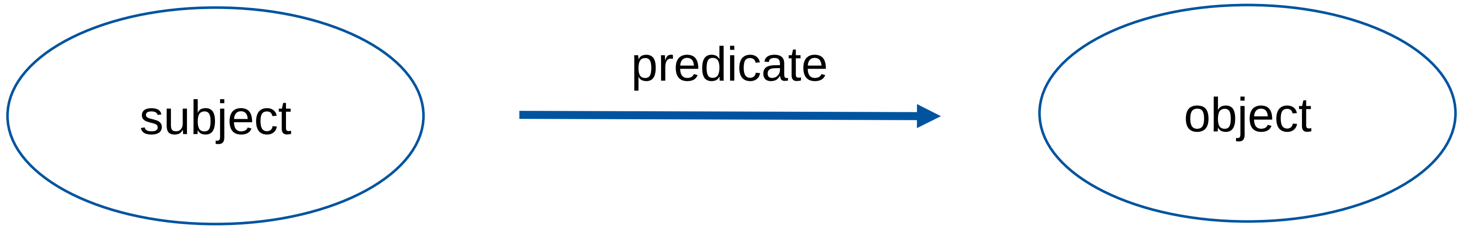
- Querying these databases allows to navigate words on the basis of lexical relationships

WordNet

- A lexical network for English
- 155.000+ words
- Organized in graphs and divided in synsets (sets of synonyms, i.e., semantically equivalent)
- Relationship information:
  - antonyms (cold vs warm),
  - hyponyms (daisy and rose are hyponyms of a flower, i.e., “is a” relation),
  - meronyms (wheel and engine are meronyms of a car, i.e., “part of” relation),
  - …

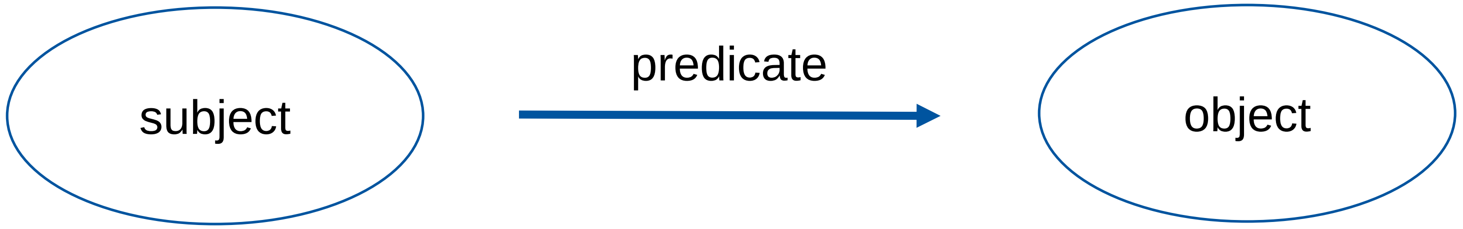
Resource Description Framework (RDF)

- One step further: a general data model
- Describes relationships between things – connections between concepts can carry any meaning
- Based on triples - statements of the following form:



- Looks simple, but with a database large enough, we can answer very complex queries

Resource Description Framework (RDF) - Example



<ent:Elizabeth II>

<rel:was_the_Queen_of>

<ent:United Kingdom>

<ent:Berlin>

<rel:located_in>

<ent:Germany>

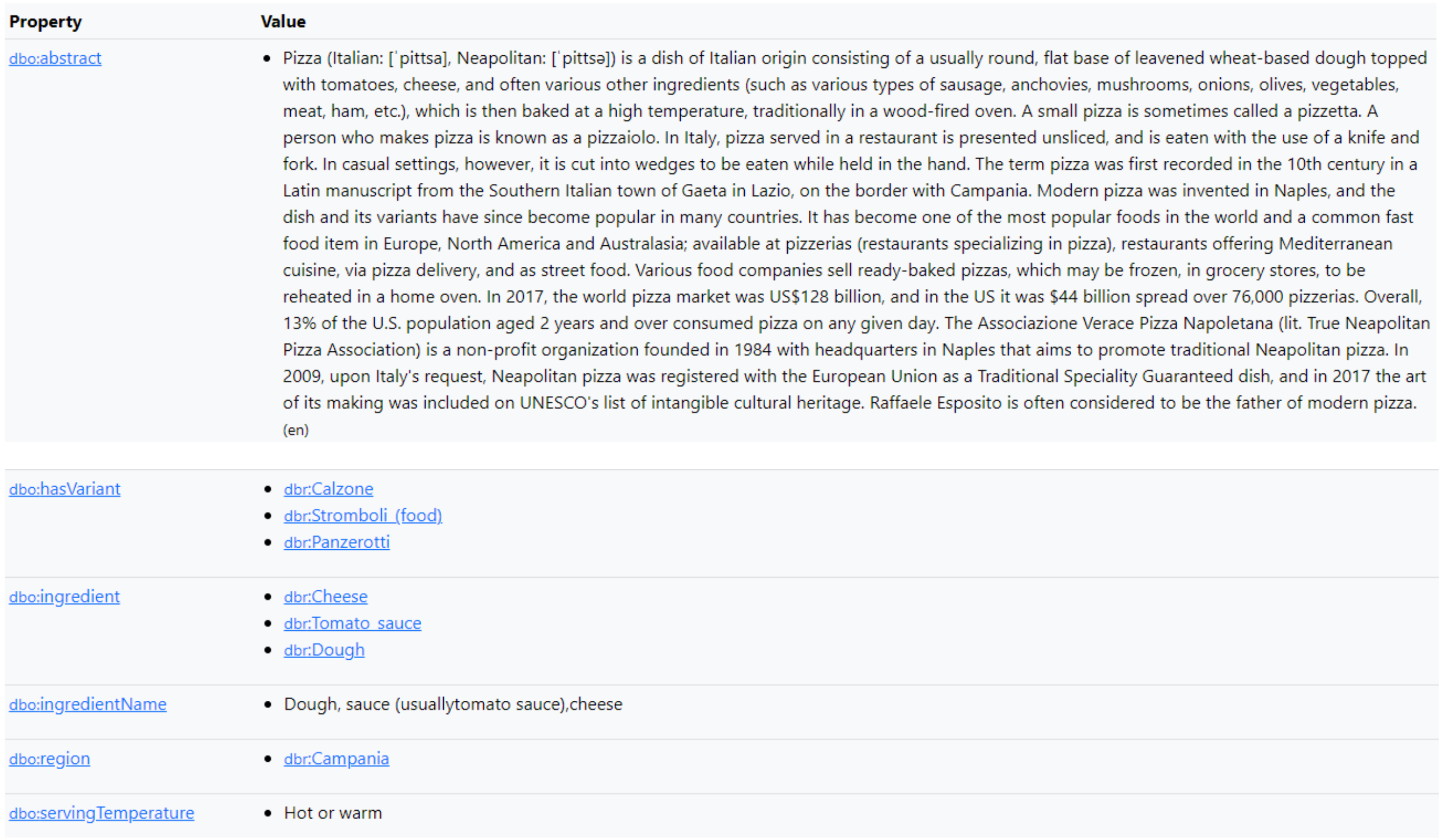
Uniform Resource Identifier (URI)

- Unique and unambiguous identifiers for entities
- Easy to define for limited domains
  - Student ID is a URI for students
  - ISBN is a URI for books
- If we want to identify all entities, it becomes more difficult

DBpedia

Wikipedia for machines

Information extracted from Wikipedia, but organized in RDF triples

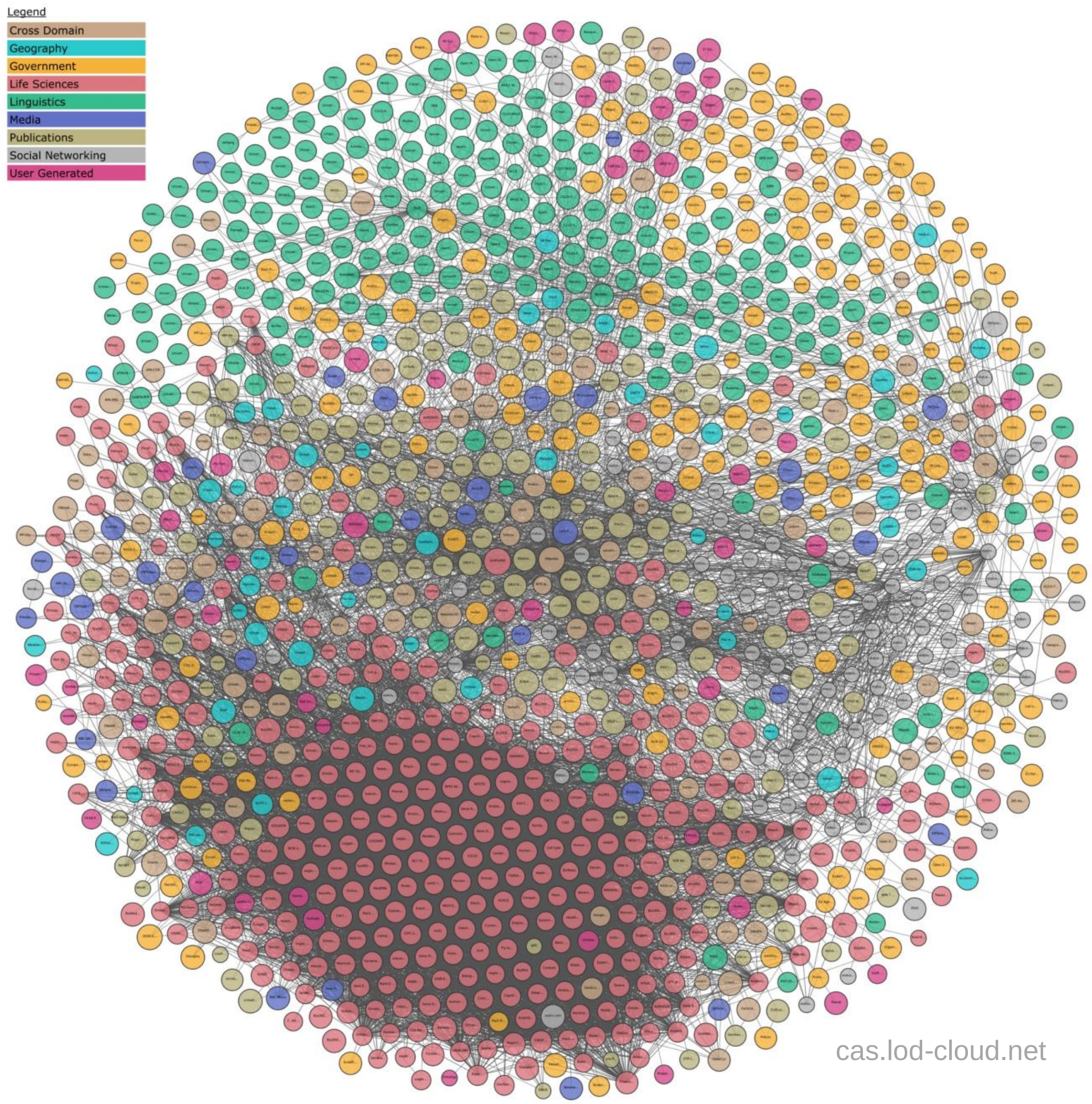


Property	Value
dbo:abstract	• Pizza (Italian: [ˈpittsa], Neapolitan: [ˈpittsə]) is a dish of Italian origin consisting of a usually round, flat base of leavened wheat-based dough topped with tomatoes, cheese, and often various other ingredients (such as various types of sausage, anchovies, mushrooms, onions, olives, vegetables, meat, ham, etc.), which is then baked at a high temperature, traditionally in a wood-fired oven. A small pizza is sometimes called a pizzetta. A person who makes pizza is known as a pizzaiolo. In Italy, pizza served in a restaurant is presented unsliced, and is eaten with the use of a knife and fork. In casual settings, however, it is cut into wedges to be eaten while held in the hand. The term pizza was first recorded in the 10th century in a Latin manuscript from the Southern Italian town of Gaeta in Lazio, on the border with Campania. Modern pizza was invented in Naples, and the dish and its variants have since become popular in many countries. It has become one of the most popular foods in the world and a common fast food item in Europe, North America and Australasia; available at pizzerias (restaurants specializing in pizza), restaurants offering Mediterranean cuisine, via pizza delivery, and as street food. Various food companies sell ready-baked pizzas, which may be frozen, in grocery stores, to be reheated in a home oven. In 2017, the world pizza market was US$128 billion, and in the US it was $44 billion spread over 76,000 pizzerias. Overall, 13% of the U.S. population aged 2 years and over consumed pizza on any given day. The Associazione Verace Pizza Napoletana (lit. True Neapolitan Pizza Association) is a non-profit organization founded in 1984 with headquarters in Naples that aims to promote traditional Neapolitan pizza. In 2009, upon Italy's request, Neapolitan pizza was registered with the European Union as a Traditional Speciality Guaranteed dish, and in 2017 the art of its making was included on UNESCO's list of intangible cultural heritage. Raffaele Esposito is often considered to be the father of modern pizza. (en)
dbo:hasVariant	• dbr:Calzone • dbr:Stromboli_(food) • dbr:Panzerotti
dbo:ingredient	• dbr:Cheese • dbr:Tomato_sauce • dbr:Dough
dbo:ingredientName	• Dough, sauce (usuallytomato sauce),cheese
dbo:region	• dbr:Campania
dbo:servingTemperature	• Hot or warm

https://dbpedia.org/page/Pizza, 07/03/2023

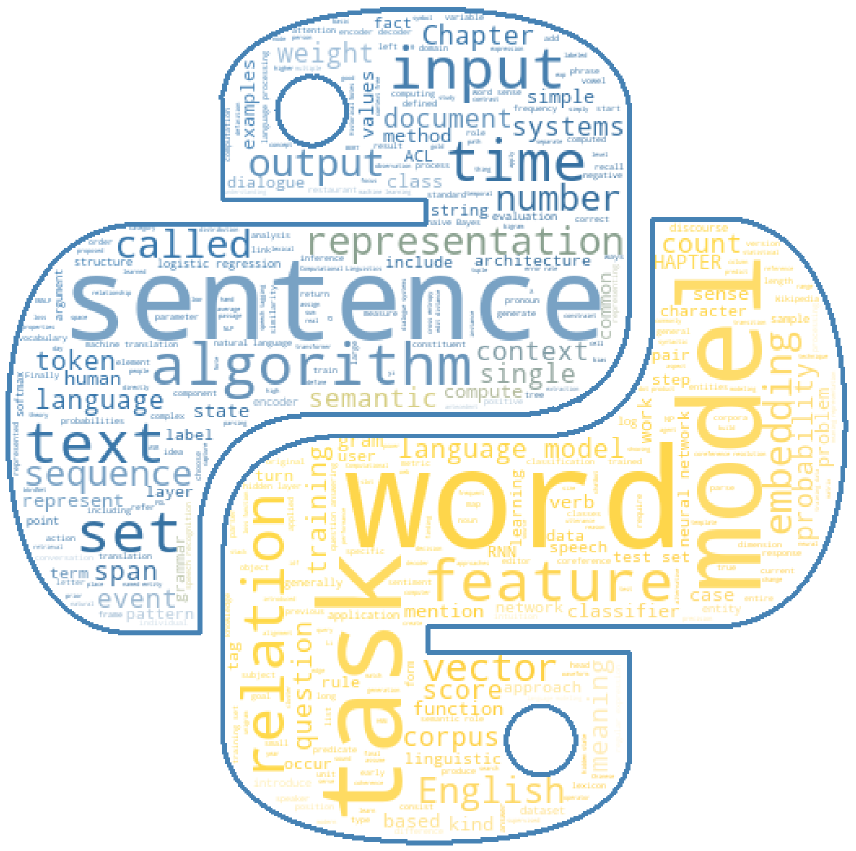
Linked Open Data

- A number of open source databases are interconnected and form the Linked Open Data – a large database of statements
- Publicly available and reusable
- Combined dimensions – tens of billions of RDF triples!
- Example: http://cas.lod-cloud.net/
- Word in text can be related to entities in such a database



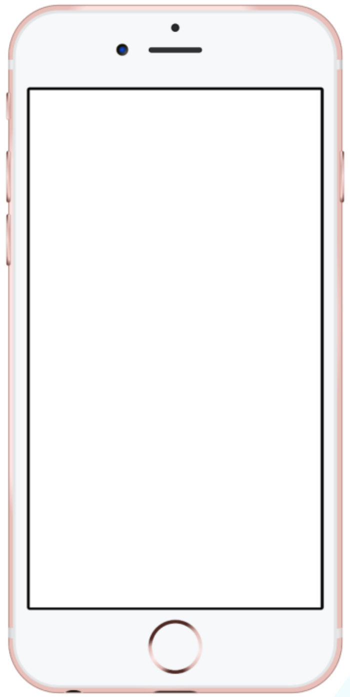
Text Mining

1. Introduction to Text Mining
2. Text Preprocessing
3. Modeling
4. Enriching Text: Linked Data
5. **N-gram Model**
6. Learning a Representation
7. Word2vec



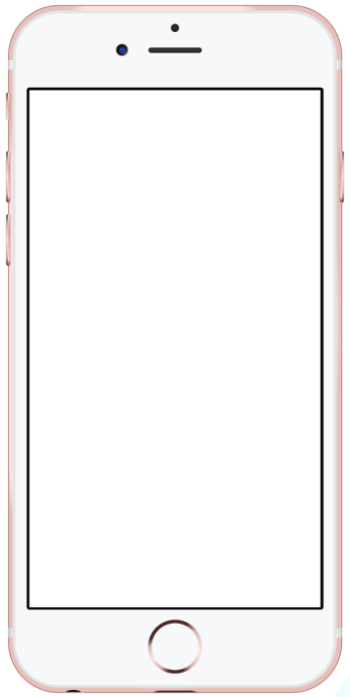
Completion Prediction

- Completion prediction is another text mining application
- Given a sequence of words, predict the next word
- Examples:
  - Apple gets the most of its revenue from selling cell _____
  - Your code crashed, it has a _____



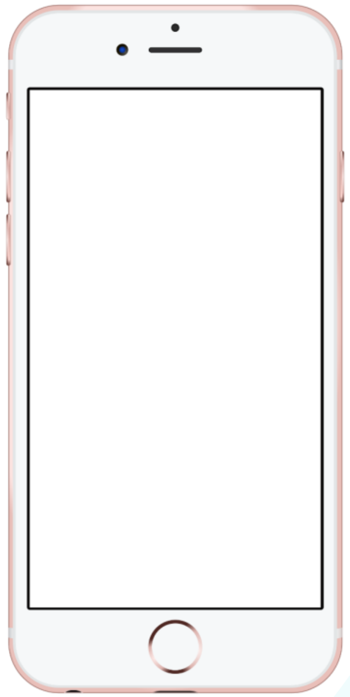
Completion Prediction

- Completion prediction is another text mining application
- Given a sequence of words, predict the next word
- Examples:
  - Apple gets the most of its revenue from selling cell ______ phones
  - Your code crashed, it has a ______



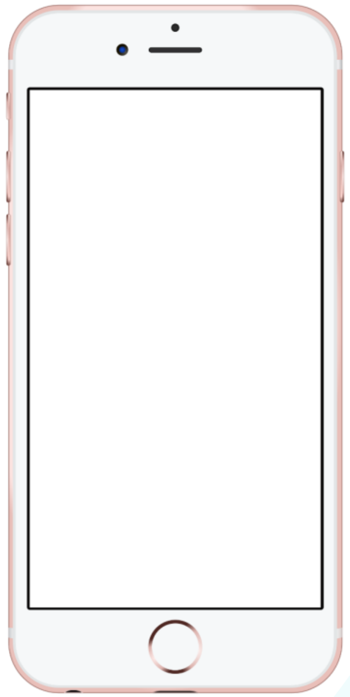
Completion Prediction

- Completion prediction is another text mining application
- Given a sequence of words, predict the next word
- Examples:
  - Apple gets the most of its revenue from selling cell ______ phones
  - Your code crashed, it has a ______ bug



Completion Prediction

- Completion prediction is another text mining application
- Given a sequence of words, predict the next word
- Examples:
  - Apple gets the most of its revenue from selling cell _____ phones
  - Your code crashed, it has a _____ bug
- The BoW model is not useful in this case
- We need a model that is capable to retain information about the word order



N-gram Model

- N-gram models use sequences of consecutive tokens, instead of individual tokens
- The N in N-gram indicates the length of a sequence

N-gram Model (N=1)

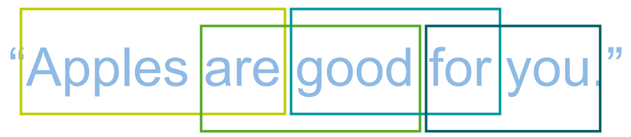
“Apples are good for you.”

Unigram model:

[‘apples’, ‘are’, ‘good’, ‘for’, ‘you’]

The unigram model is identical to BoW!

N-gram Model (N=2)



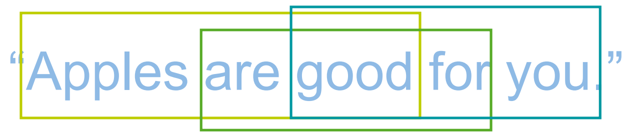
Unigram model:

[‘apples’, ‘are’, ‘good’, ‘for’, ‘you’]

Bigram model:

[(‘apples’, ‘are’), (‘are’, ‘good’), (‘good’, ‘for’), (‘for’, ‘you’)]

N-gram Model (N=3)



Unigram model:

[‘apples’, ‘are’, ‘good’, ‘for’, ‘you’]

Bigram model:

[(‘apples’, ‘are’), (‘are’, ‘good’), (‘good’, ‘for’), (‘for’, ‘you’)]

Trigram model:

[(‘apples’, ‘are’, ‘good’), (‘are’, ‘good’, ‘for’), (‘good’, ‘for’, ‘you’)]

Preprocessing

- Preprocessing steps are context and application dependent
- For example, stop word removal is useful for the document classification, however, it should not be utilized for completion prediction
- Examples in this video do not use any preprocessing steps
  (no stop word removal, stemming or lemmatization)

Motivation

- We use N-grams is to estimate the probability of a word occurrence given its prior context

- An N-gram uses N-1 tokens for the context
  - Unigram:

    P(phone) ← Probability that a random word in the corpus is "phone"
  - Bigram:

    P(phone | cell)
  - Trigram:

    P(phone | your cell)

Motivation

- We use N-grams is to estimate the probability of a word occurrence given its prior context

- An N-gram uses N-1 tokens for the context
  - Unigram:

    P(phone)
  - Bigram:

    P(phone | cell)
  - Trigram:

    P(phone | your cell)

Probability that a random word in the corpus is “phone” given that the previous is “cell”

Motivation

- We use N-grams is to estimate the probability of a word occurrence given its prior context

- An N-gram uses N-1 tokens for the context
  - Unigram:

    P(phone)
  - Bigram:

    P(phone | cell)
  - Trigram:

    P(phone | your cell)

Probability that a random word in the corpus is “phone” given that the two previous words are “your” and “cell”

Example

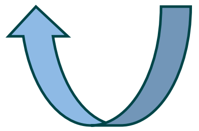
2-gram
If you wait too long for the perfect moment, the perfect moment will pass you by.

3-gram
If you wait too long for the perfect moment, the perfect moment will pass you by.

4-gram
If you wait too long for the perfect moment, the perfect moment will pass you by.

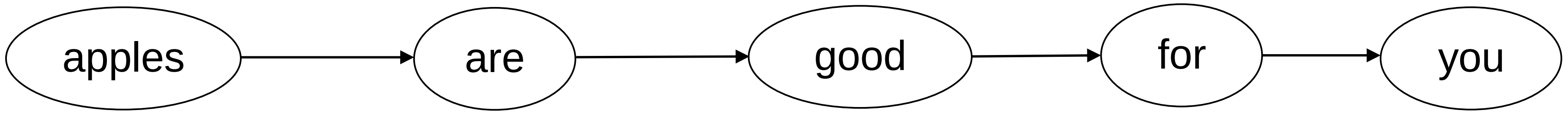
5-gram
If you wait too long for the perfect moment, the perfect moment will pass you by.

6-gram
If you wait too long for the perfect moment, the perfect moment will pass you by.



N-gram Model

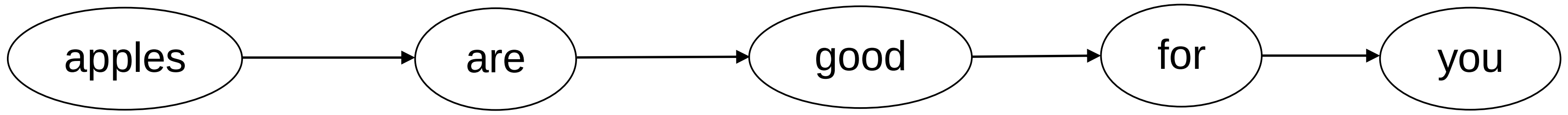
- Our goal is to quantify the strength of the relationship represented by the arrows
- Use N-grams to learn a function that quantifies the probability of a word “w” given its prior context



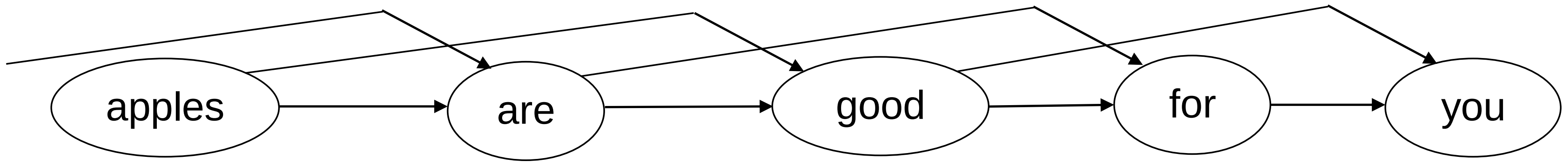
Bigram: arrows from w_1 to w_2 imply that “w_2 depends on w_1”.

N-gram Model

- Our goal is to quantify the strength of the relationship represented by the arrows
- Use N-grams to learn a function that quantifies the probability of a word "w" given its prior context



Bigram: arrows from w_1 to w_2 imply that "w_2 depends on w_1".



Trigram: "w_k depends on w_{k-1} and w_{k-2}"

N-gram Model – Computing Probabilities

Word sequences $w_1^n = w_1 \ldots w_n$

Chain rule: $P(w_1^n) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1^2)\ldots P(w_n \mid w_1^{n-1}) = \Pi_{k=1}^n P(w_k \mid w_1^{k-1})$

$w_1^n = w_1 \ldots w_n$

$w_1^{k-1} = w_1 \ldots w_{k-1}$

Example (*n* = *5*):

$$P(\overset{1}{I},\ \overset{2}{like},\ \overset{3}{pizza},\ \overset{4}{with},\ \overset{5}{mushrooms}) =$$

$$P(\underset{1}{I}) \cdot P(\underset{2}{like} \mid \underset{1}{I}) \cdot P(\underset{3}{pizza} \mid I,\ \underset{2}{like}) \cdot P(\underset{4}{with} \mid I,\ like,\ \underset{3}{pizza}) \cdot P(\underset{5}{mushrooms} \mid I,\ like,\ pizza,\ \underset{4}{with})$$

N-gram Model – Markov Assumption

- The Markov assumption: probability of a certain word depends only on a limited context
- Assumption on bigrams: $P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-1})$
- Assumption on N-grams: $P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-N+1}^{n-1})$

Bigrams: N=2
Trigrams: N=3
…

$w_{n-N+1}, w_{n-N+2} \dots w_{n-1}$

- Formalization of 'the last N-1 words matter'
- Note that we need to handle the special case $n - N + 1 < 1$ by using default values

N-gram Model – Computing Probabilities

- Based on the Markov assumption and the chain rule:

→ Chain rule: $P(w_1^n) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1^2)...P(w_n \mid w_1^{n-1}) = \Pi_{k=1}^{n} P(w_k \mid w_1^{k-1})$

Example (n=5)

$$P(\overset{1}{I},\ \overset{2}{like},\ \overset{3}{pizza},\ \overset{4}{with},\ \overset{5}{mushrooms}) =$$

$$P(\underset{1}{I}) \cdot P(\underset{2}{like} \mid \underset{1}{I}) \cdot P(\underset{3}{pizza} \mid I,\ \underset{2}{like}) \cdot P(\underset{4}{with} \mid I,\ like,\ \underset{3}{pizza}) \cdot P(\underset{5}{mushrooms} \mid I,\ like,\ pizza,\ \underset{4}{with})$$

N-gram Model – Computing Probabilities

- Based on the Markov assumption and the chain rule:
- Bigram approximation: $P(w_1^n) \approx \Pi_{k=1}^{n} P(w_k \mid w_{k-1})$

The last word

→ Chain rule: $P(w_1^n) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1^2)...P(w_n \mid w_1^{n-1}) = \Pi_{k=1}^{n} P(w_k \mid w_1^{k-1})$

Example (n=5)

$$P(\overset{1}{I},\ \overset{2}{like},\ \overset{3}{pizza},\ \overset{4}{with},\ \overset{5}{mushrooms}) =$$

$$P(\underset{1}{I}) \cdot P(\underset{2}{like} \mid \underset{1}{I}) \cdot P(\underset{3}{pizza} \mid \cancel{I},\ \underset{2}{like}) \cdot P(\underset{4}{with} \mid \cancel{I},\ \cancel{like},\ \underset{3}{pizza}) \cdot P(\underset{5}{mushrooms} \mid \cancel{I},\ \cancel{like},\ \cancel{pizza},\ \underset{4}{with})$$

Special case (missing context)

N-gram Model – Computing Probabilities

- Based on the Markov assumption and the chain rule:
- Trigram approximation: $P(w_1^n) \approx \Pi_{k=1}^{n} P(w_k \mid \underbrace{w_{k-2}^{k-1}}_{\text{The last 2 words}})$

→ Chain rule: $P(w_1^n) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1^2)...P(w_n \mid w_1^{n-1}) = \Pi_{k=1}^{n} P(w_k \mid w_1^{k-1})$

Example (n=5)

$$P(\overset{1}{I},\ \overset{2}{like},\ \overset{3}{pizza},\ \overset{4}{with},\ \overset{5}{mushrooms}) =$$

$$\underbrace{P(\underset{1}{I})}_{\text{Special case (missing context)}} \cdot P(\underset{2}{like} \mid \underbrace{\underset{1}{I}}_{\text{Special case (missing context)}}) \cdot P(\underset{3}{pizza} \mid I,\ \underset{2}{like}) \cdot P(\underset{4}{with} \mid \cancel{I},\ like,\ \underset{3}{pizza}) \cdot P(\underset{5}{mushrooms} \mid \cancel{I},\ \cancel{like},\ pizza,\ \underset{4}{with})$$

N-gram Model – Computing Probabilities

- Based on the Markov assumption and the chain rule:
- N-gram approximation: $P(w_1^n) \approx \Pi_{k=1}^{n} P(w_k \mid w_{k-N+1}^{k-1})$

The last N-1 words

→ Chain rule: $P(w_1^n) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1^2)...P(w_n \mid w_1^{n-1}) = \Pi_{k=1}^{n} P(w_k \mid w_1^{k-1})$

Maximum Likelihood Estimation in N-gram Models

- How to estimate these probabilities?

  → Maximum Likelihood Estimation method (MLE)

- Count function: $C(w_1^n)$ → the number of occurrences of w_1^n in the corpus

- Bigram MLE estimation: $P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$

the most likely w_n given the last word

Maximum Likelihood Estimation in N-gram Models

- How to estimate these probabilities?

  → Maximum Likelihood Estimation method (MLE)

- Count function: $C(w_1^n)$ → the number of occurrences of w_1^n in the corpus

- Bigram MLE estimation: $P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$

  the most likely w_n given the last word

- N-gram MLE estimation: $P(w_n \mid w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}w_n)}{C(w_{n-N+1}^{n-1})}$

  the most likely w_n given the last N-1 words

Example

Bigrams (2-grams)

Corpus of 3 sentences:

```
<s> we want to eat <e>
<s> we have pizza <e>
<s> today we have pizza for dinner <e>
```

Bigram MLE estimation:

$$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1} w_n)}{C(w_{n-1})}$$

→ Ensure context for N-grams: add special ‘words’ on sentence beginning and end

Example

Bigrams (2-grams)

Corpus of 3 sentences:

```
<s> we want to eat <e>
<s> we have pizza <e>
<s> today we have pizza for dinner <e>
```

Bigram MLE estimation:

$$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

$$\boxed{P(\texttt{we} \mid \texttt{<s>}) = \tfrac{2}{3}}$$
$$P(\texttt{<e>} \mid \texttt{pizza}) = \tfrac{1}{2}$$
$$P(\texttt{today} \mid \texttt{<s>}) = \tfrac{1}{3}$$

$$P(\texttt{pizza} \mid \texttt{have}) = \tfrac{2}{2}$$
$$P(\texttt{have} \mid \texttt{we}) = \tfrac{2}{3}$$
$$P(\texttt{want} \mid \texttt{we}) = \tfrac{1}{3}$$

Example

Bigrams (2-grams)

Corpus of 3 sentences:

```
<s> we want to eat <e>
<s> we have pizza <e>
<s> today we have pizza for dinner <e>
```

Bigram MLE estimation:

$$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

$$P(\texttt{we} \mid \texttt{<s>}) = \frac{2}{3}$$
$$\boxed{P(\texttt{<e>} \mid \texttt{pizza}) = \frac{1}{2}}$$
$$P(\texttt{today} \mid \texttt{<s>}) = \frac{1}{3}$$

$$P(\texttt{pizza} \mid \texttt{have}) = \frac{2}{2}$$
$$P(\texttt{have} \mid \texttt{we}) = \frac{2}{3}$$
$$P(\texttt{want} \mid \texttt{we}) = \frac{1}{3}$$

Example

Bigrams (2-grams)

Corpus of 3 sentences:

```
<s> we want to eat <e>
<s> we have pizza <e>
<s> today we have pizza for dinner <e>
```

Bigram MLE estimation:

$$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

$$P(\texttt{we} \mid \texttt{<s>}) = \frac{2}{3}$$
$$P(\texttt{<e>} \mid \texttt{pizza}) = \frac{1}{2}$$
$$\boxed{P(\texttt{today} \mid \texttt{<s>}) = \frac{1}{3}}$$

$$P(\texttt{pizza} \mid \texttt{have}) = \frac{2}{2}$$
$$P(\texttt{have} \mid \texttt{we}) = \frac{2}{3}$$
$$P(\texttt{want} \mid \texttt{we}) = \frac{1}{3}$$

Example

Bigrams (2-grams)

Corpus of 3 sentences:

```
<s> we want to eat <e>
<s> we have pizza <e>
<s> today we have pizza for dinner <e>
```

Bigram MLE estimation:

$$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

$$P(\texttt{we} \mid \texttt{<s>}) = \frac{2}{3} \qquad P(\texttt{pizza} \mid \texttt{have}) = \frac{2}{2}$$

$$P(\texttt{<e>} \mid \texttt{pizza}) = \frac{1}{2} \qquad \boxed{P(\texttt{have} \mid \texttt{we}) = \frac{2}{3}}$$

$$P(\texttt{today} \mid \texttt{<s>}) = \frac{1}{3} \qquad P(\texttt{want} \mid \texttt{we}) = \frac{1}{3}$$

Example 2

Estimation for a whole sentence

- Use the Markov assumption and the chain rule
- Multiply the bigrams (N-gram) probabilities

$$P(\texttt{<s>,I, like, pizza, with, mushrooms, <e>}) \approx$$
$$P(\texttt{I} \mid \texttt{<s>}) \cdot P(\texttt{like} \mid \texttt{I}) \cdot P(\texttt{pizza} \mid \texttt{like}) \cdot P(\texttt{with} \mid \texttt{pizza}) \cdot P(\texttt{mushrooms} \mid \texttt{with}) \cdot P(\texttt{<e>} \mid \texttt{mushrooms})$$

N-grams: Effectiveness

Sentences generated by sampling from the distributions of different N-gram models trained on Shakespeare’s sonnets:

- 1 gram: “of hour loved worship sweet metre moving fore rank and the for of fair better a art careful graciously with“
- 2 gram: “thou wilt prove me thus by day by but day by their physicians know not to the bath for blunting the“
- 3 gram: “methinks no face so gracious is as mine importune thee root pity in the face sweet love remembered such wealth brings that“
- 4 gram: “yet this shall I neer know but live in doubt till my bad angel fire my good one out“

Limitation: Sparseness

- Sparseness
- Assume we use sentences based on a vocabulary of 30 000 words.
  - Bigrams: $30000^2 = 900000000$
  - Trigrams: $30000^3 = 27000000000000$
  - 4-grams: $30000^4 = 810000000000000000$

- In a vector model based on N-grams, this is the number of dimensions
- The dataset would contain mostly zeroes

  → this method is non-applicable for classification problems (as-is)

Limitation: Sparseness

- Imagine we split into test and training data
  - Train an N-gram model on the training corpus
  - Test it on sequences of length N-1 in the test corpus

- Likely, there are sequences in the test corpus that are unseen in the training data
  - The model assigns probability 0
  - The model “as is” is overfitting the training data

→ Solution: Smoothing

Solution: Smoothing

- Chip-off some probability from likely sequences
- Distribute it to unseen sequences (→ small but non-zero probability)

- Example: Laplace Smoothing

$$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \Longrightarrow P_{Laplace}(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$$

Ensure non-zero probability

Normalize using size of the vocabulary, to make the sum of all probabilities 1

N-grams on Characters

- The basic version of the N-gram model is often applied to characters instead of words
- The vocabulary is just the alphabet (much smaller than all possible words)

4-grams

Once_upon_a_time: Once
Once_upon_a_time: nce_
Once_upon_a_time: ce_u
Once_upon_a_time: e_up

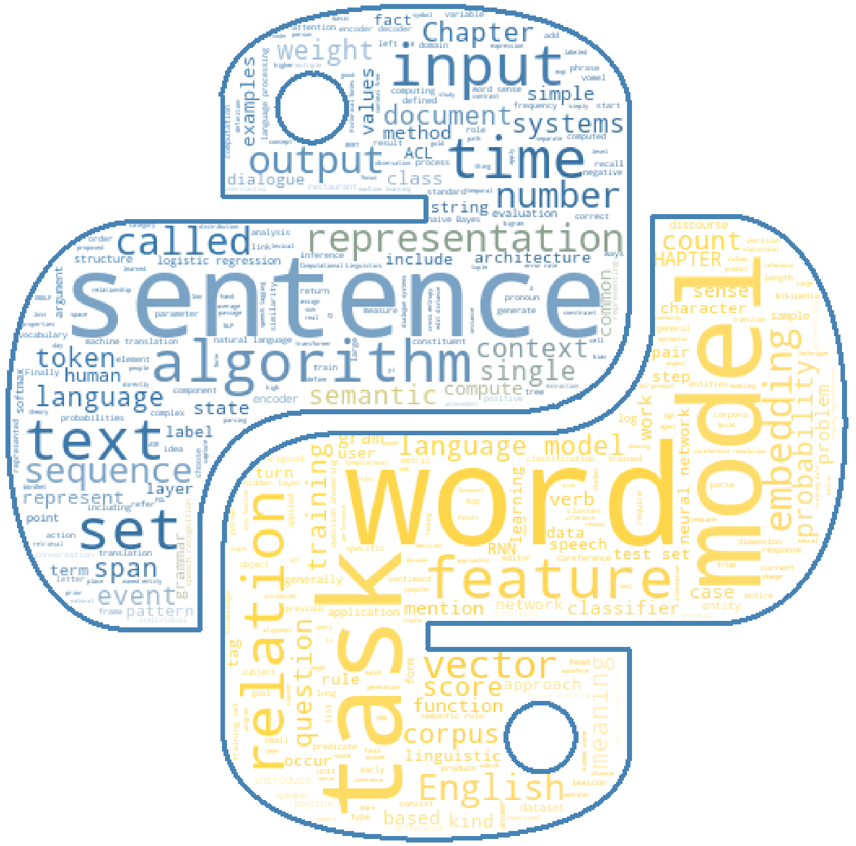
Unknowns

- What if a test corpus contains words that do not occur in the training corpus?
- These are called Out of Vocabulary words (OOV) or unknowns

  → the model cannot predict them
- We can model such unknown words using a specific pseudo-word: <UNK>

Text Mining

1. Introduction to Text Mining
2. Text Preprocessing
3. Modeling
4. Enriching Text: Linked Data
5. N-gram Model
6. **Learning a Representation**
7. Word2vec



Sparseness

- The representations of words are based on the length of the dictionary (BoW, N-grams)

  → tend to be very long

- This is because words behave like the categorical data

  → needs one-hot encoding, one feature per word

Sparseness

- Categorical data needs one-hot representation

  → A vector of length equal to the number of possible values

Rome	= [1, 0, 0, 0, … , 0]
Paris	= [0, 1, 0, 0, … , 0]
Italy	= [0, 0, 1, 0, … , 0]
France	= [0, 0, 0, 0, … , 1]

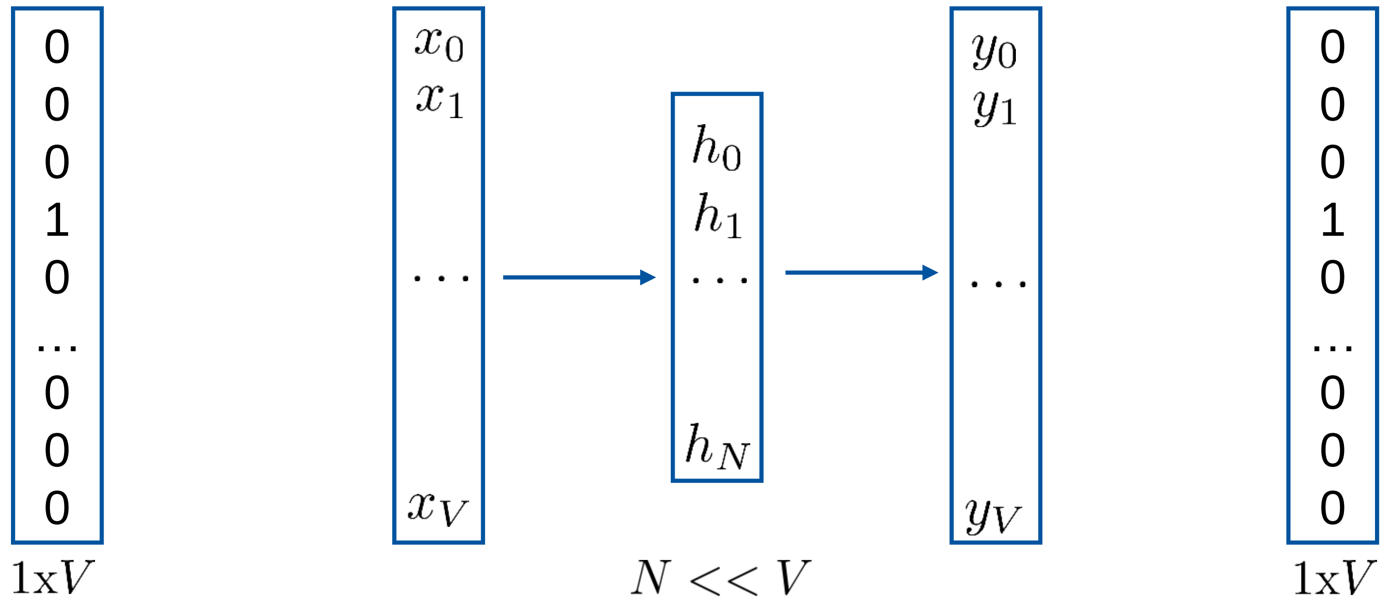
- Text mining: the number of possible values is the size of the vocabulary

Learning a Representation

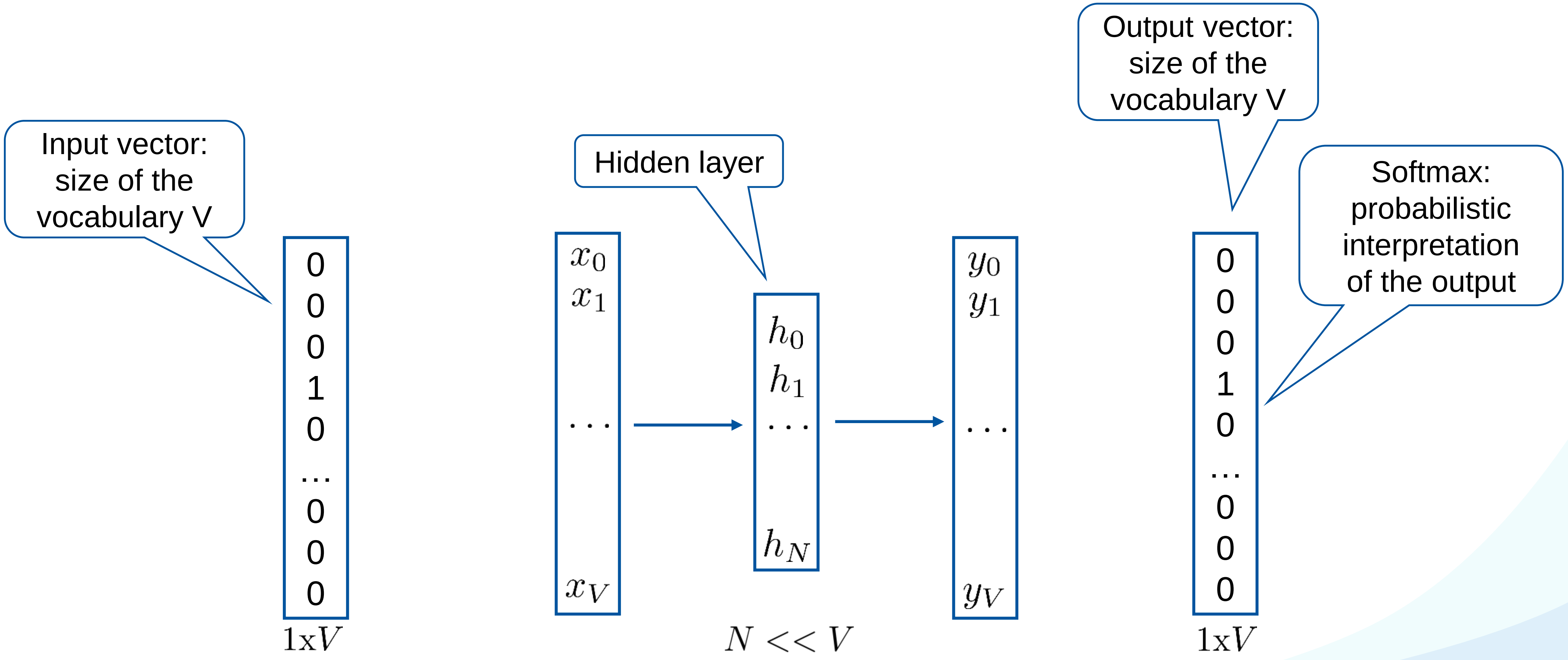
- Solution: use machine learning to automatically identify a smaller data representation
- Autoencoders use neural networks

Autoencoders

- Consider a neural network with a specific structure:
  - Input and output layers of dimension V
  - Hidden layer of dimension N with N << V

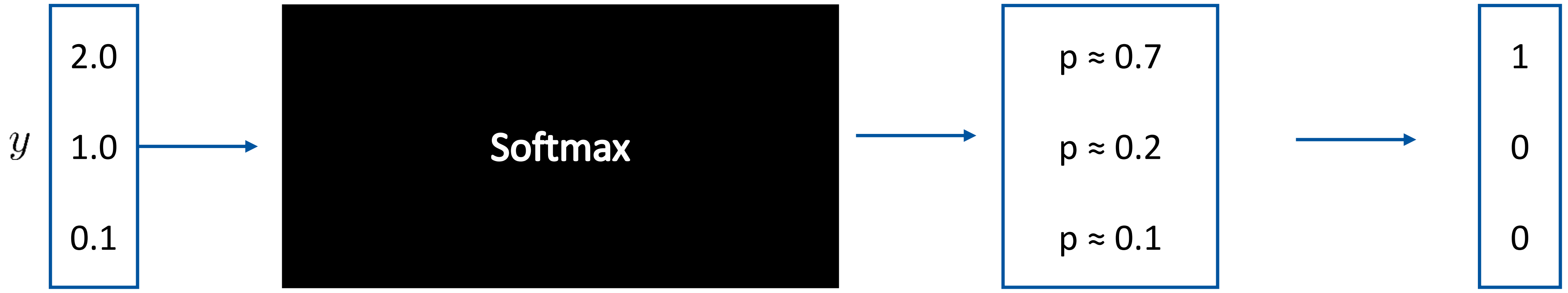


Autoencoders


Input vector: size of the vocabulary V
0 0 0 1 0 … 0 0 0
1xV
x_0 x_1 $\dots$ x_V
Hidden layer
h_0 h_1 $\dots$ h_N
$N << V$
y_0 y_1 $\dots$ y_V
Output vector: size of the vocabulary V
0 0 0 1 0 … 0 0 0
1xV
Softmax: probabilistic interpretation of the output

Autoencoders

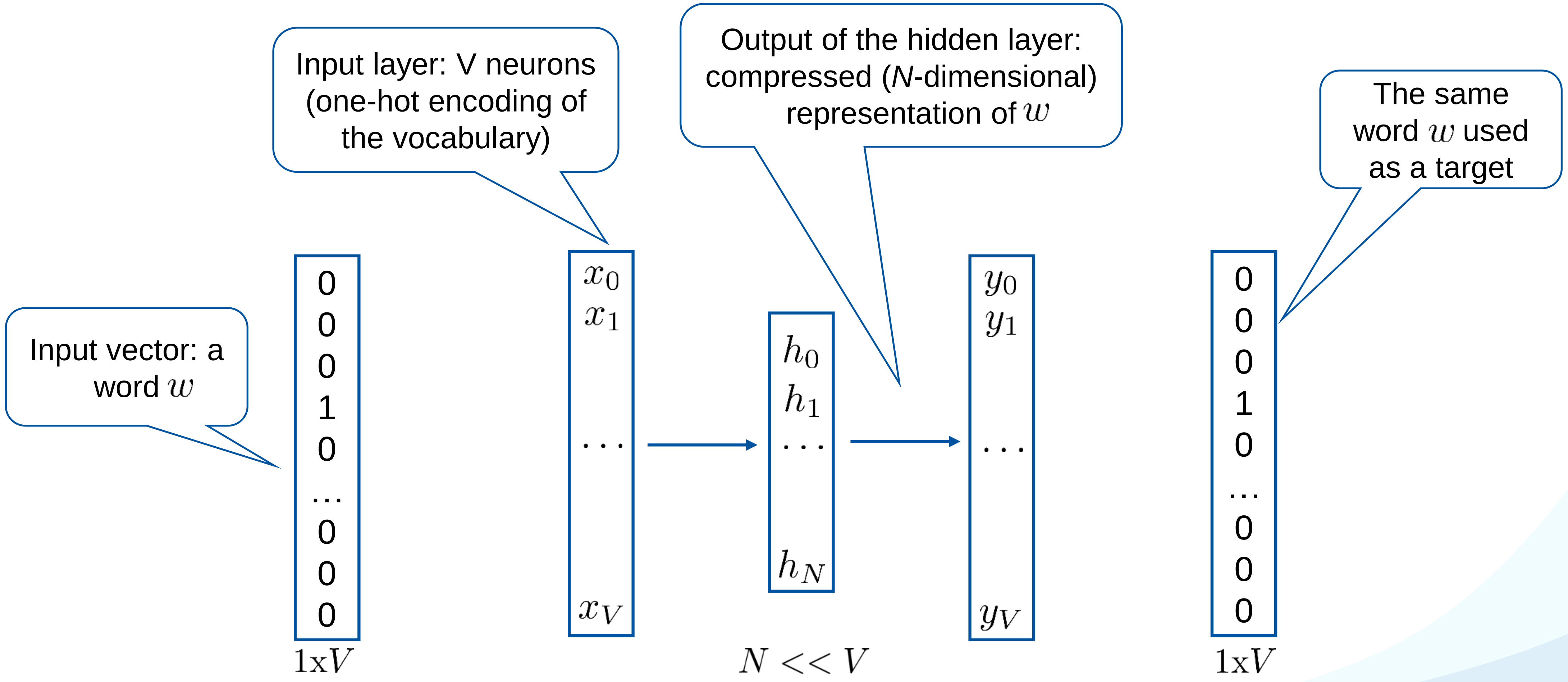
Softmax: transform the output of the NN into interpretable probabilities



Autoencoders - Training

- Train network with the same data in the input and output
  - Convert the corpora into one-hot encoded vectors
  - Feed these one by one into the neural network
  - Perform backpropagation comparing the output with the input
- We obtain a network that outputs (almost) the same one hot encoded vector given as input

Autoencoders


Input vector: a word w
Input layer: V neurons (one-hot encoding of the vocabulary)
Output of the hidden layer: compressed (*N*-dimensional) representation of w
The same word w used as a target
0 0 0 1 0 … 0 0 0
x_0 x_1 $\dots$ x_V
h_0 h_1 $\dots$ h_N
y_0 y_1 $\dots$ y_V
0 0 0 1 0 … 0 0 0
$1\mathrm{x}V$
$N << V$
$1\mathrm{x}V$

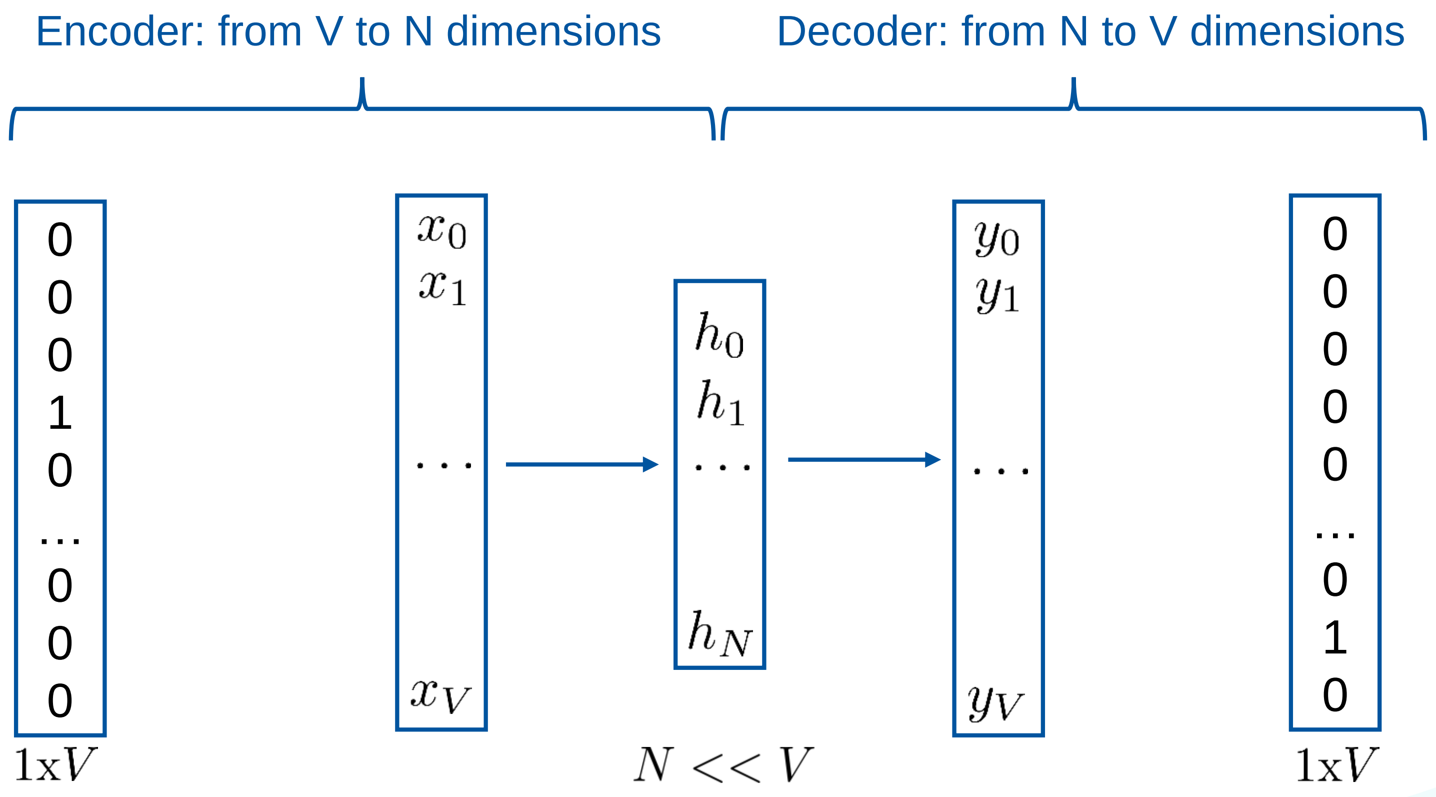
Autoencoders

- The output of the hidden layer gives us a compressed representation of the input word
- The compression ratio is V/N
- These neural networks are called autoencoders

  → one way to automatically learn a representation of data

Autoencoders

It is possible to split an autoencoder in the encoding and decoding part after the training

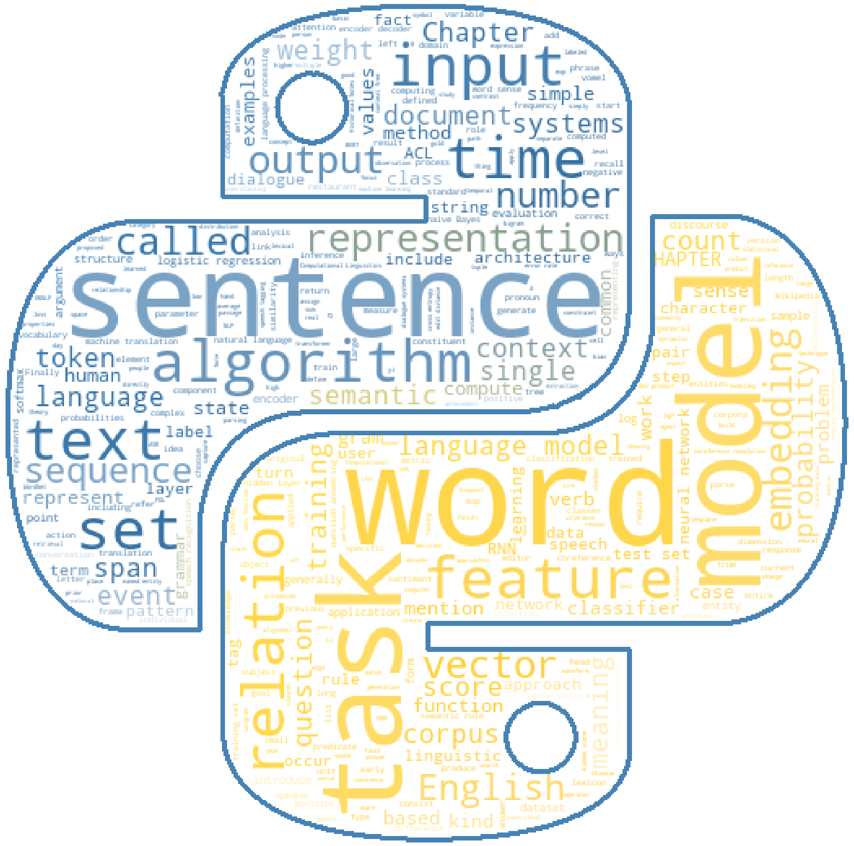

Encoder: from V to N dimensions
Decoder: from N to V dimensions
0 0 0 1 0 … 0 0 0
x_0 x_1 $\dots$ x_V
h_0 h_1 $\dots$ h_N
y_0 y_1 $\dots$ y_V
0 0 0 0 0 … 0 1 0
$1\text{x}V$
$N << V$
$1\text{x}V$

Learning a Representation

- Autoencoders are one way to automatically learn a representation of data
- Learning a smaller data representation is often called embedding
- When applied to a text, it’s referred to as word embeddings

Text Mining

1. Introduction to Text Mining
2. Text Preprocessing
3. Modeling
4. Enriching Text: Linked Data
5. N-gram Model
6. Learning a Representation
7. **Word2vec**



Word Embeddings

- Compression is very useful, but autoencoders do not incorporate context
- N-grams allow to consider the word order and the context
- Can we obtain a word embedding that contains order and context?
- → use an N-grams generalization: skip-grams

Skip-grams

K-skip N-grams

- A skip-gram is an N-gram that allows to skip words
- Skip-grams constructed for a certain skip distance K allow a total of K or less skips for N-grams
- Example:
  - A 3-skip-gram includes: 3 skips, 2 skips, 1 skip or no skip (in total)
  - A 3-skip-trigram (x,y,z) covers “xyz”, “x_yz”, “xy_z”, “x_y_z”, “x_ _yz”, “xy_ _z”, “x_ _y_z”, “x_y_ _z”, “x_ _ _yz”, “xy_ _ _z” in a document

Skip-grams - Example

"Hi Jen did you eat the cake?"

1-skip trigrams: [‘Hi Jen did’, ‘Hi Jen you’, ‘Jen did you’, ‘Jen did eat’, ‘Hi did you’, ‘did you eat’, ‘did you the’, ‘you eat cake’, …]

2-skip trigrams: [‘Hi Jen eat’, ‘did you cake’, ‘Jen you the’, …]

Recall: 2-skip-trigrams also include all 1-skip-trigrams.

Skip-grams - Example

- Skip-grams are can associate a more general notion of a context compared to N-grams
- The surrounding context can be partially skipped:

  this implies less overfitting when used in a learning phase

Word2vec

- Word2vec: one of the most popular techniques to learn word embeddings using shallow neural networks
- Idea: an extension of the autoencoder method

  → Word2vec learns a compressed representation of a word NOT from itself, but from its context

- Step 1: build a training set extracting skip-grams from a text
  - For each word, consider the context in a sliding window around it
  - Create tuples with the word and every skip-gram in the window that does not contain it

Word2vec – Training Set

Example with 2-skip trigrams and a window size of 2 on each side

The little goose swims on a large pond. → (the, [<s>, <s>, little]), (the, [<s>, <s>, goose]), (the, [<s>, little, goose]),

The little goose swims on a large pond. → (little, [the, goose, swims]), (little, [<s>, the, goose]), (little, [<s>, the, swims]), (little, [<s>, goose, swims])

The little goose swims on a large pond. → (goose, [the, little, swims]), (goose, [the, little, on]), (goose, [the, swims, on]), (goose, [little, swims, on])

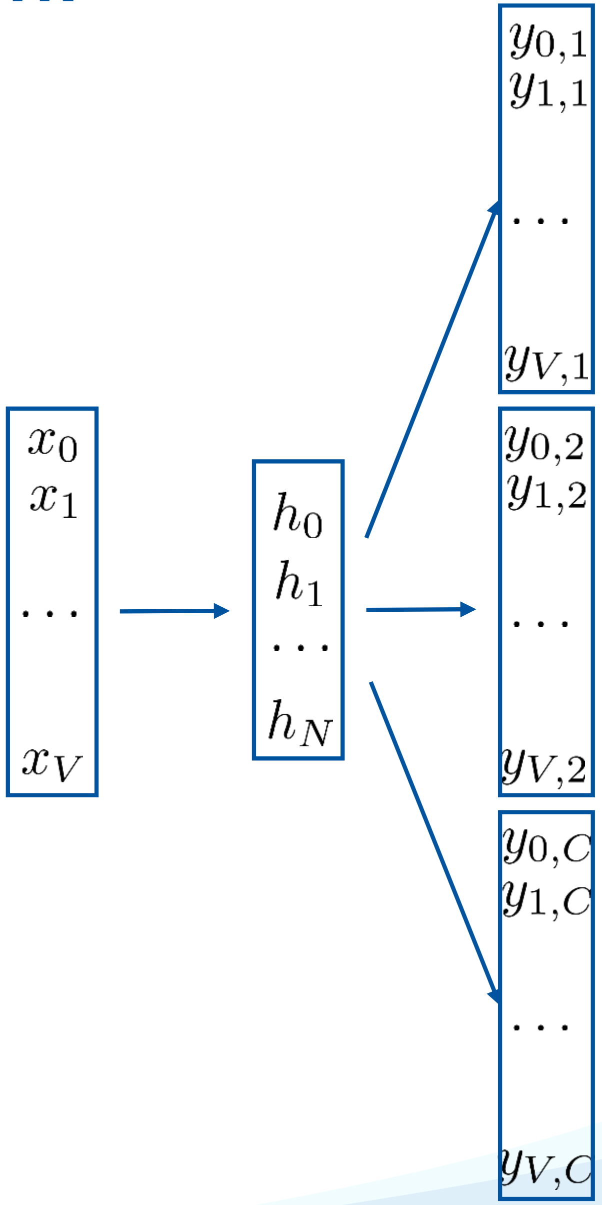
The little goose swims on a large pond. → (swims, [little, goose, on]), (swims, [little, goose, a]), (swims, [little, on, a]), (swims, [goose, on, a])

The context has fixed length representation and its order matters!

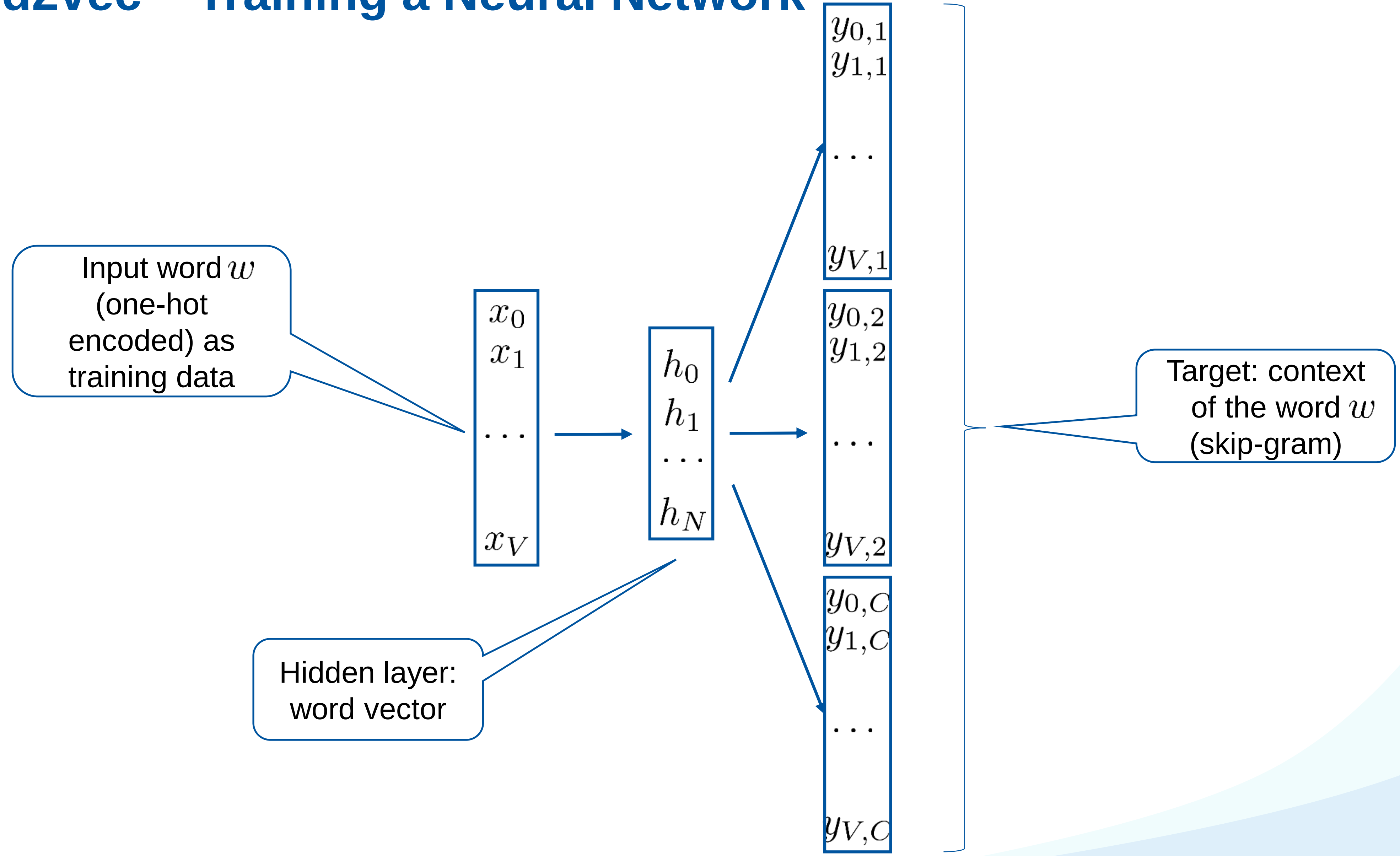
Word2vec – Training a Neural Network

- Step 2: train a neural network with a word as input and its context as output (using the tuples we built in Step 1)

- This is what the network looks like for our example training set (using skip-trigrams)



Word2vec – Training a Neural Network



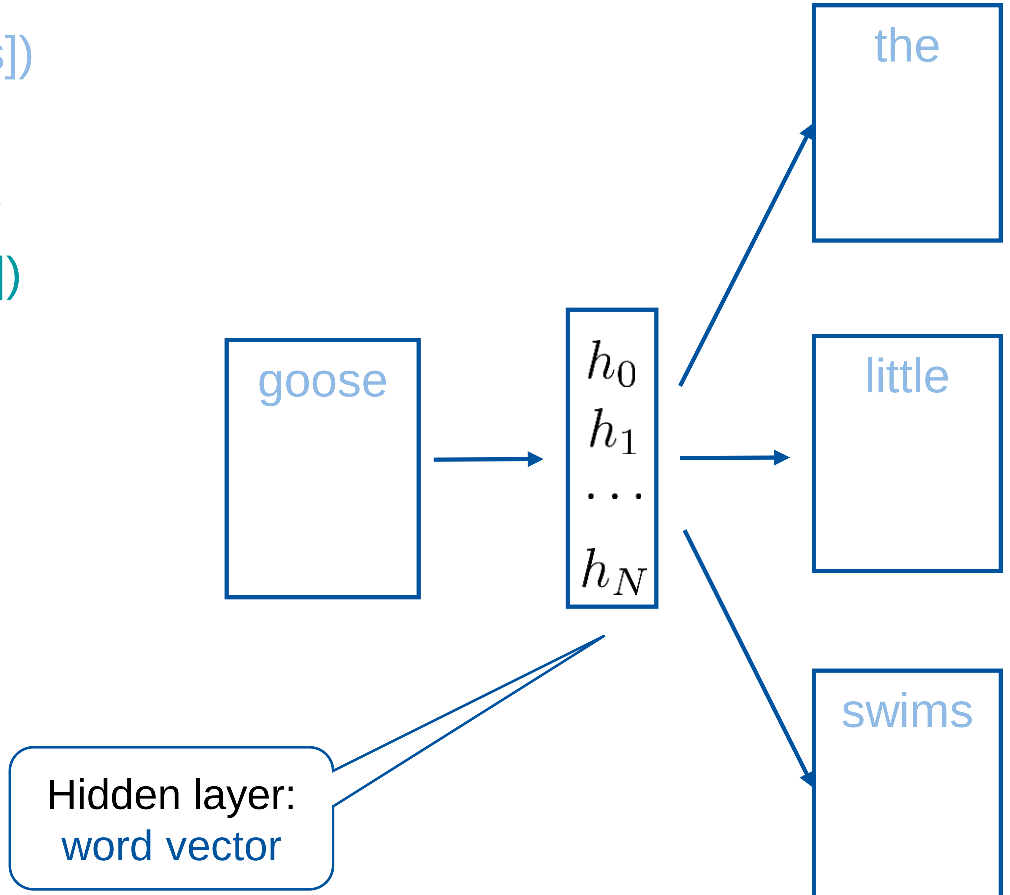
Word2vec – Training Example

(goose, [the, little, swims])

(goose, [the, little, on])

(goose, [the, swims, on])

(goose, [little, swims, on])



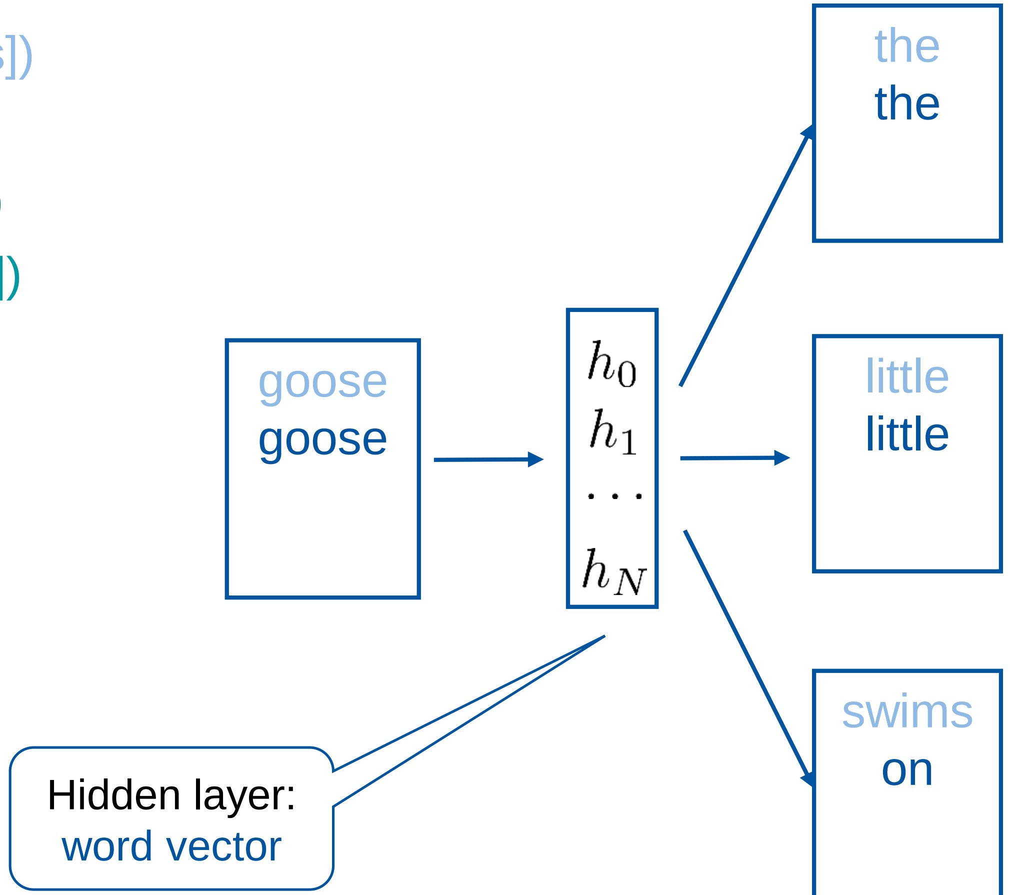
Word2vec – Training Example

(goose, [the, little, swims])

(goose, [the, little, on])

(goose, [the, swims, on])

(goose, [little, swims, on])



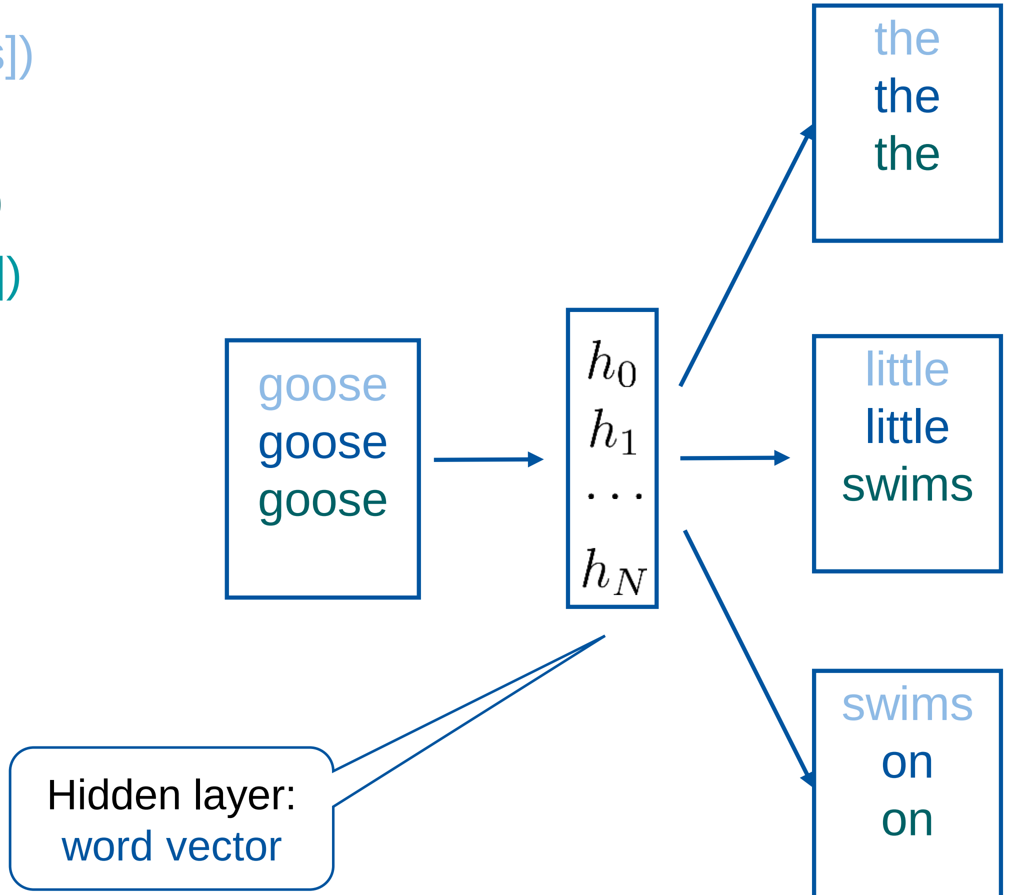
Word2vec – Training Example

(goose, [the, little, swims])

(goose, [the, little, on])

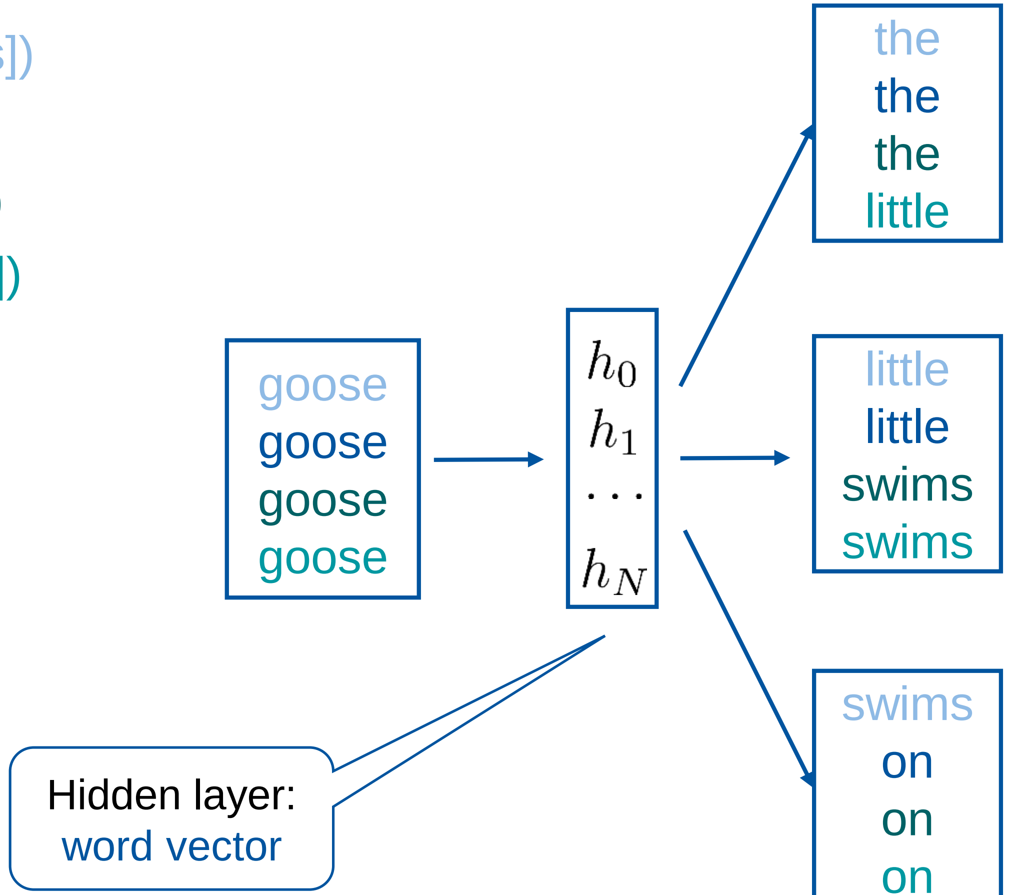
(goose, [the, swims, on])

(goose, [little, swims, on])



Word2vec – Training Example

(goose, [the, little, swims])
(goose, [the, little, on])
(goose, [the, swims, on])
(goose, [little, swims, on])



Word2vec – Training Example

(goose, [the, little, swims])

(goose, [the, little, on])

(goose, [the, swims, on])

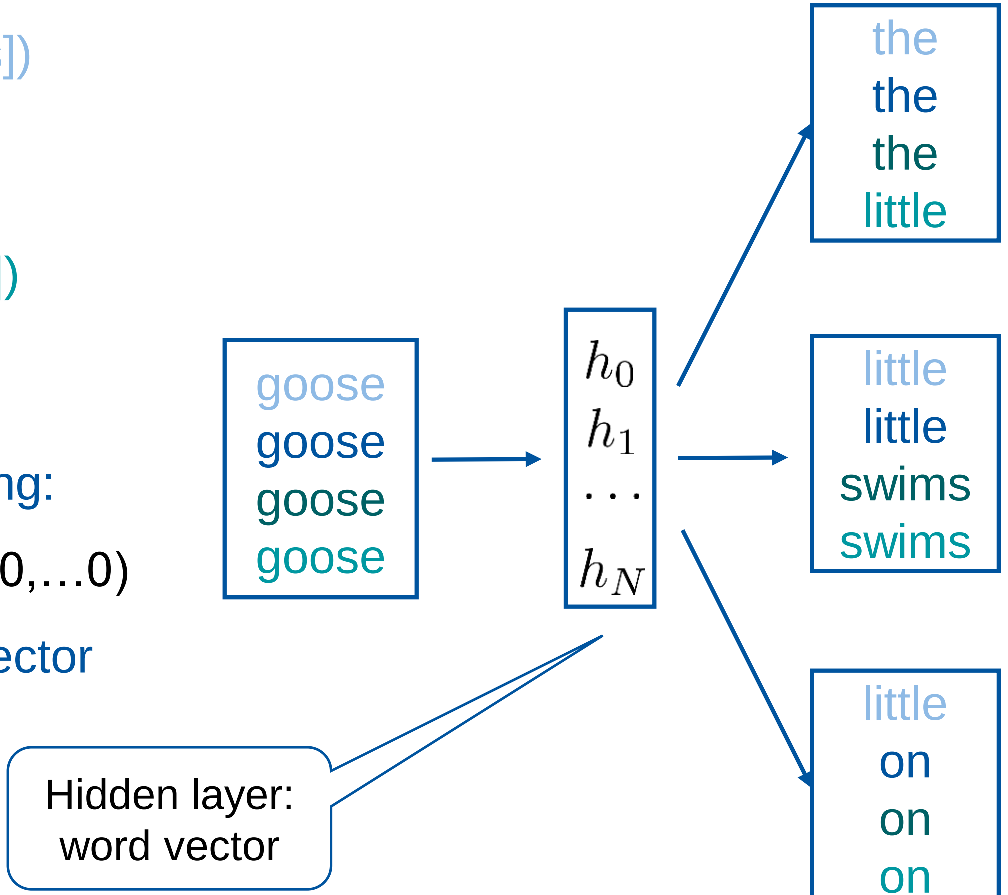
(goose, [little, swims, on])

Hidden Layer after training:

brown = (0,0,0,0,0,0,0,1,0,…0)

is mapped to the word vector

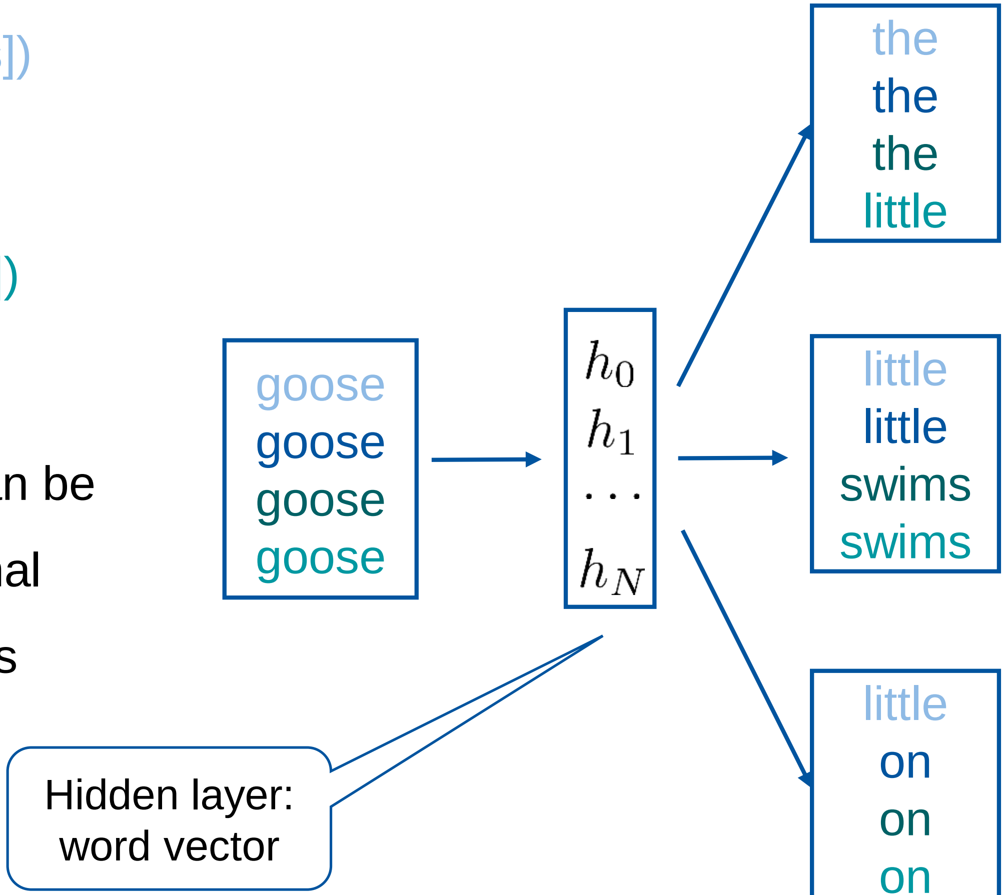
(0.23, 0.74, 0.10)



Word2vec – Training Example

(goose, [the, little, swims])
(goose, [the, little, on])
(goose, [the, swims, on])
(goose, [little, swims, on])

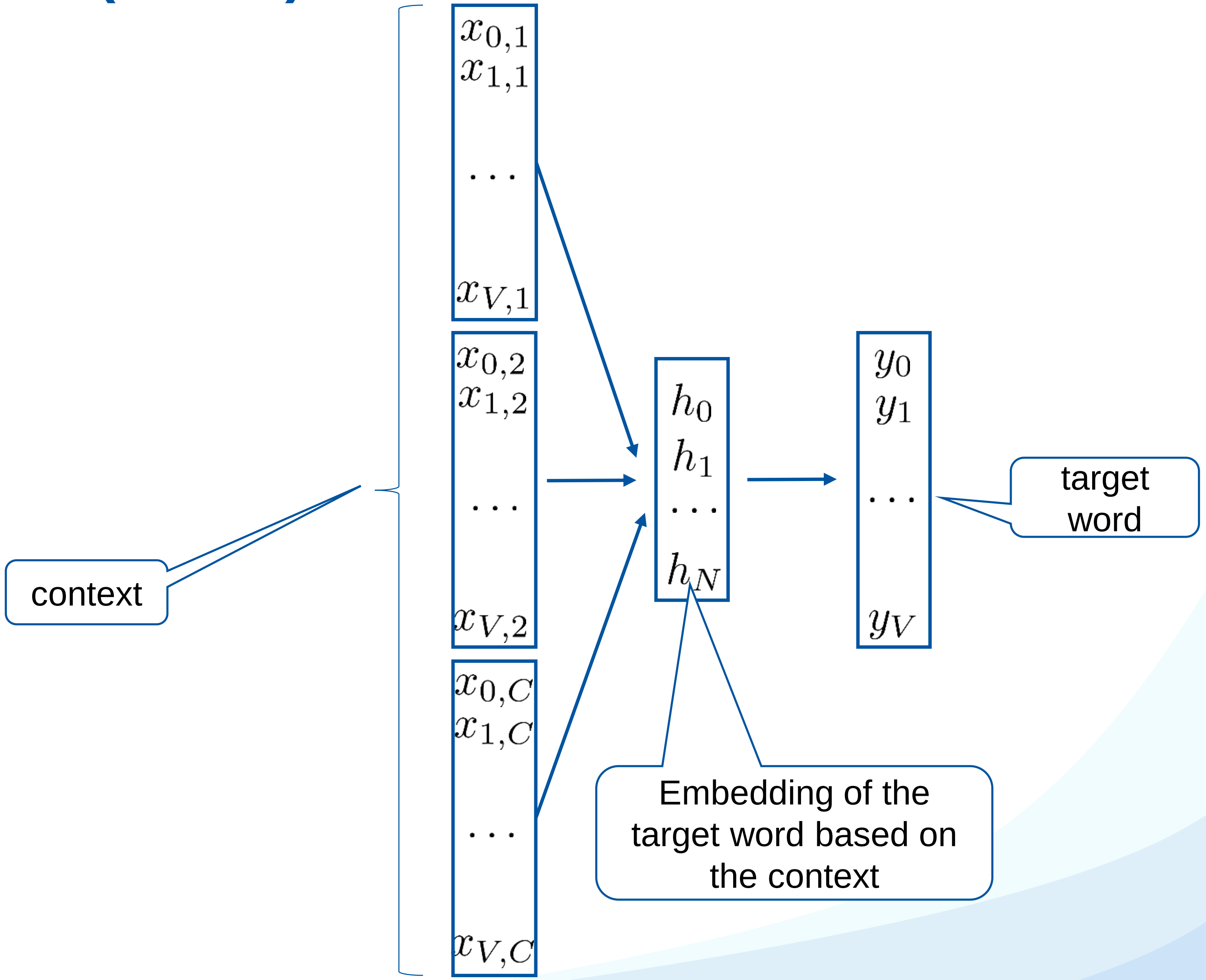
Resulting word vector can be used as lower dimensional input for other techniques



Continuous Bag of Words (CBoW)

Another Variant

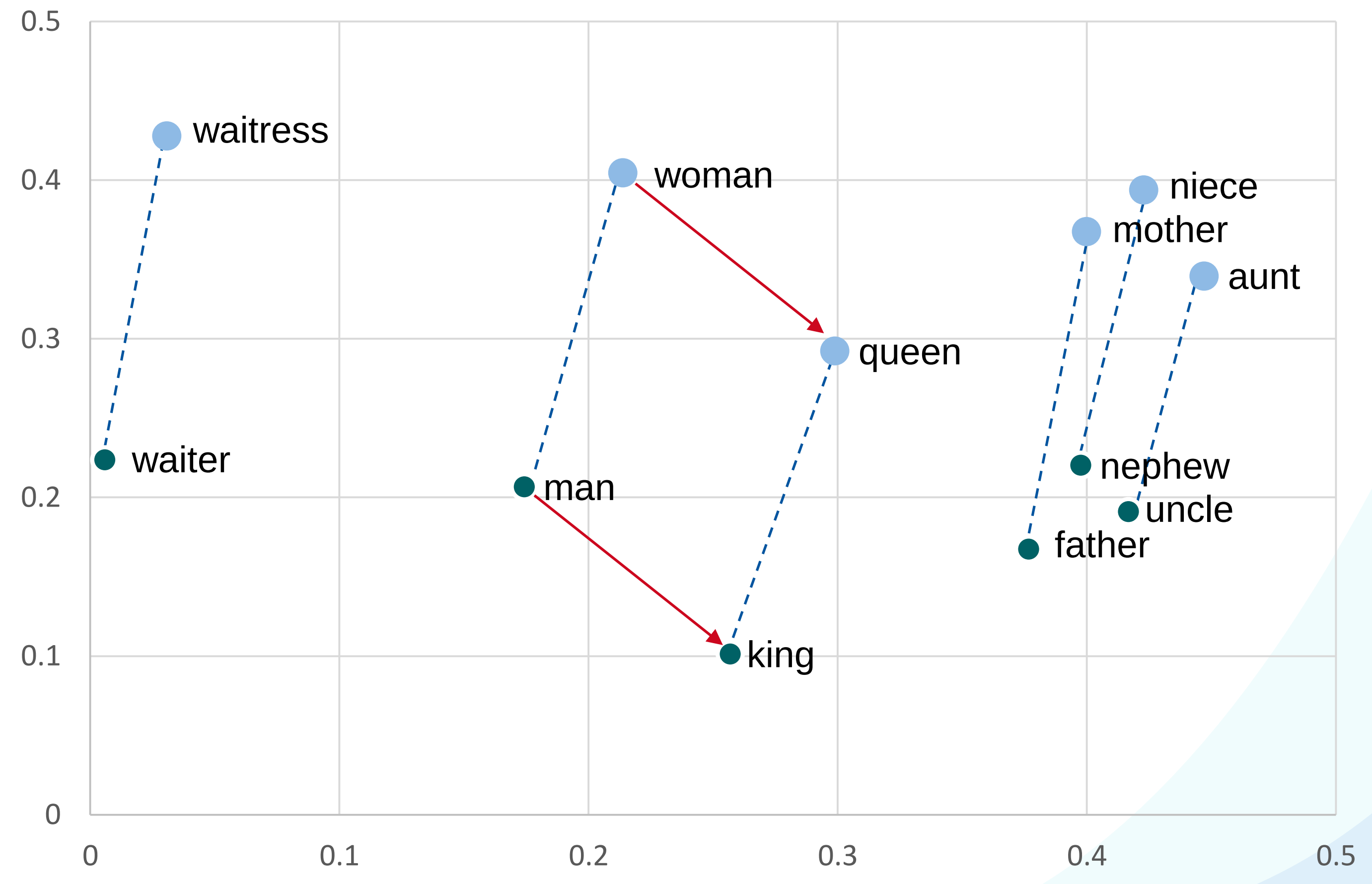
The input is the context and the output is the word


$x_{0,1}$ $x_{1,1}$ $\dots$ $x_{V,1}$
$x_{0,2}$ $x_{1,2}$ $\dots$ $x_{V,2}$
$x_{0,C}$ $x_{1,C}$ $\dots$ $x_{V,C}$
h_0 h_1 $\dots$ h_N
y_0 y_1 $\dots$ y_V
context
target word
Embedding of the target word based on the context

Word2vec – Advantages

- Word2vec captures the meaning of a word using its context
- Returns a powerful and meaningful word representation
  - Vectors for similar words are close to each other
  - It's possible to add and remove context with vector operations

king – man + woman = queen
Father – man + woman = mother
Waitress – woman + man = waiter
…



Doc2vec

- Same principles as Word2vec
- Uses one-hot encoded documents
- Training set of tuples: (one-hot encoded document, skip-gram from that document)

Doc2vec

Example with 2-skip trigrams (but any feature is possible):

0001	‘Cats are the only pet of the felines family, while dogs are canids.’	(0001, [<s>, cats, are]), (0001, [cats, are, the]), (0001, [cats, are, only]), ...
0010	‘Cats are the third-most popular pet in the US.’	(0010, [<s>, cats, are]), (0010, [cats, are, the]), (0010, [cats, are, third]), ...
0100	‘Dogs have been selected for millennia as pet animals.’	(0100, [<s>, dogs, have]), (0100, [dogs, have, been]), (0100, [dogs, have, selected]), ...
1000	‘Normally, dogs are not aggressive towards other dogs outside their territory.’	(1000, [<s>, normally, dogs]), (1000, [normally, dogs, are]), (1000, [normally, dogs, not]), ...

One-hot encodings of four documents

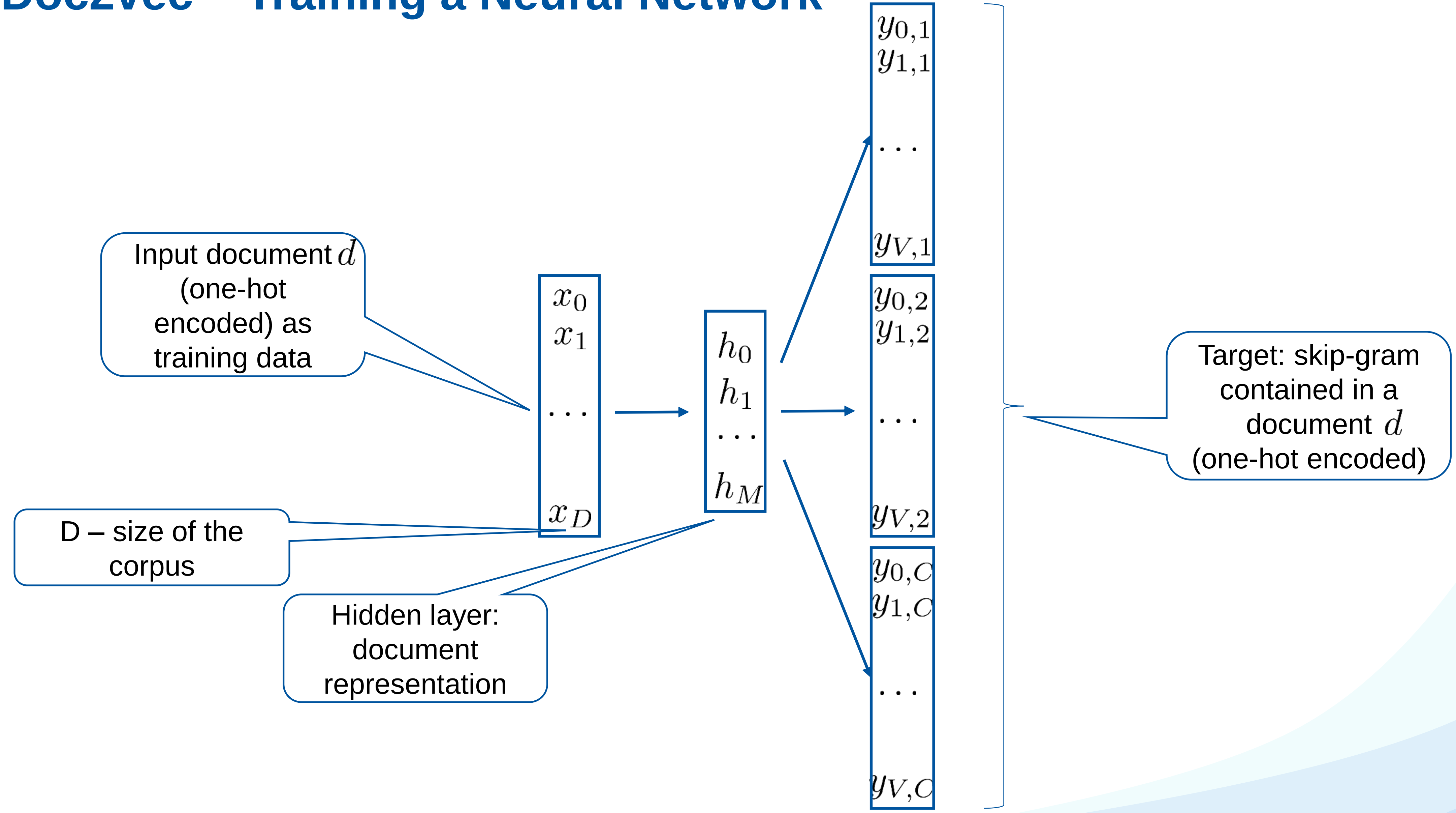
Doc2vec – Training a Neural Network

Learn a representation for a document by training a neural network

- Same architecture as before
- the input is the one-hot encoding of the document
- the output is a skip-gram of the document

→ Hidden layer gives a compact representation of the document

Doc2vec – Training a Neural Network


Input document d (one-hot encoded) as training data
D – size of the corpus
Hidden layer: document representation
Target: skip-gram contained in a document d (one-hot encoded)
x_0 x_1 $\dots$ x_D
h_0 h_1 $\dots$ h_M
$y_{0,1}$ $y_{1,1}$ $\dots$ $y_{V,1}$
$y_{0,2}$ $y_{1,2}$ $\dots$ $y_{V,2}$
$y_{0,C}$ $y_{1,C}$ $\dots$ $y_{V,C}$

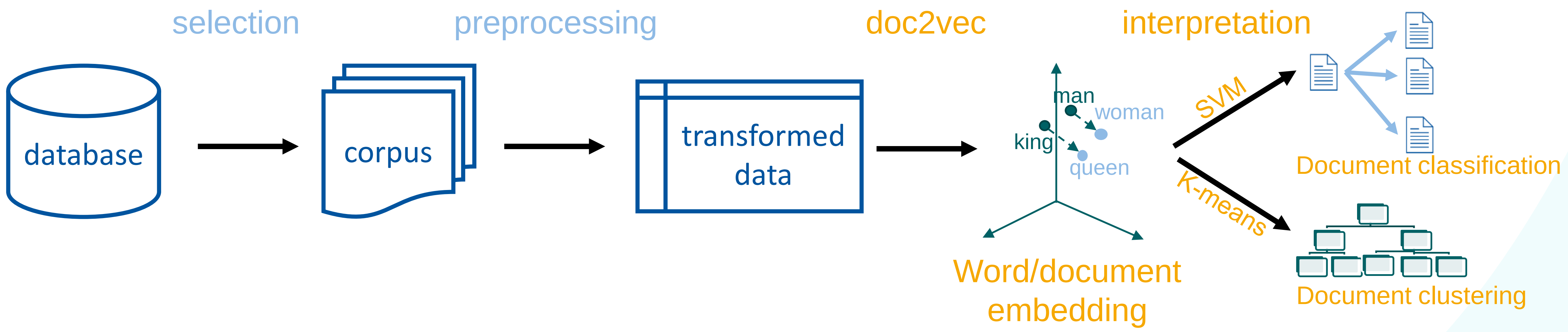
Doc2vec

- Doc2vec: find a low-dimensional, expressive, powerful and context-aware vector representation for documents in the corpus
- Considers the problems we have seen previously:
  - Keeps track of the order of words
  - Avoid the problem of high sparseness of textual data
  - Gives a fixed length representation of the documents

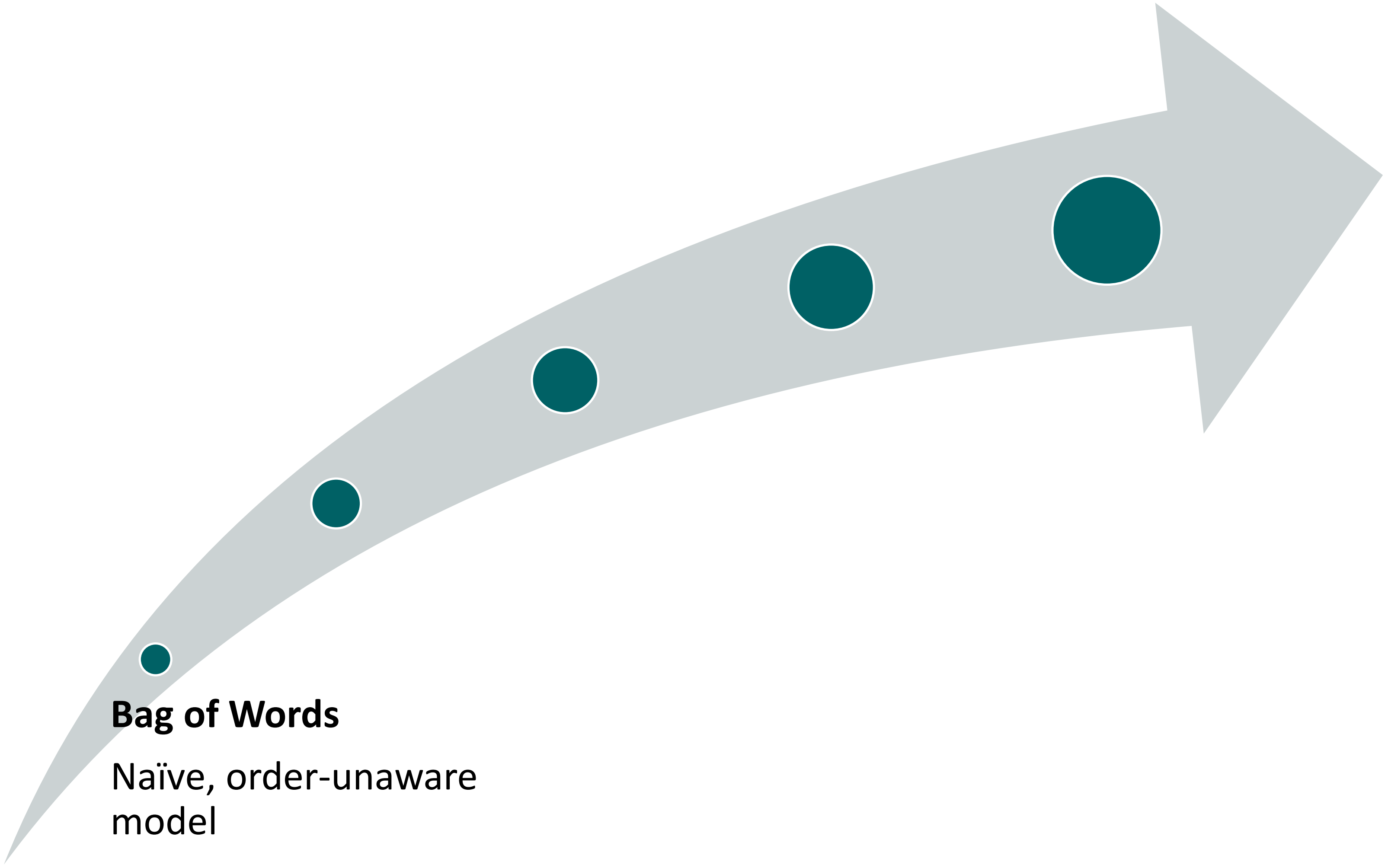
    → suitable as input for other data science applications (e.g. clustering)

- Many other variations possible…

Text Mining Pipeline With Doc2vec



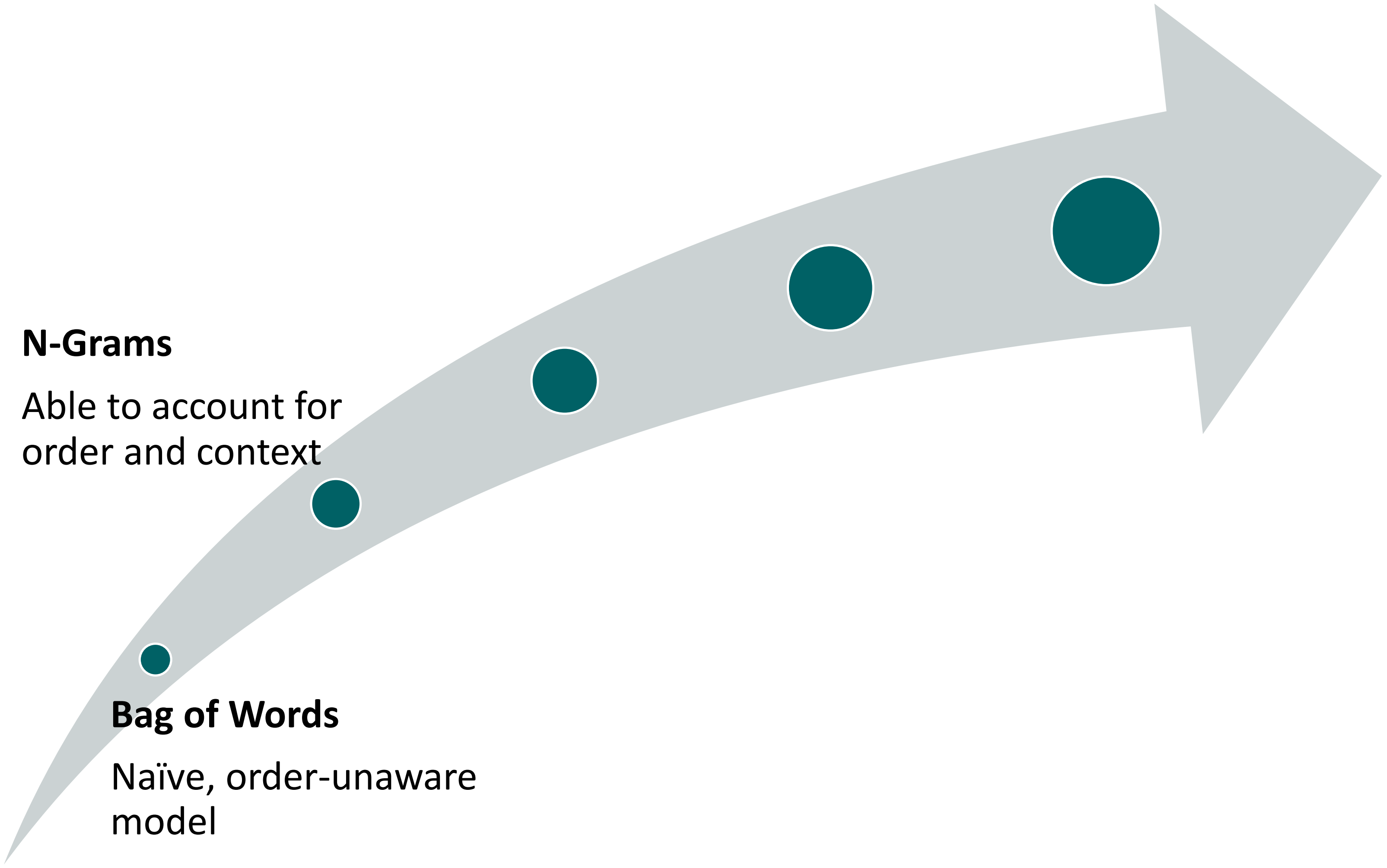
Outlook



Bag of Words

Naïve, order-unaware model

Outlook



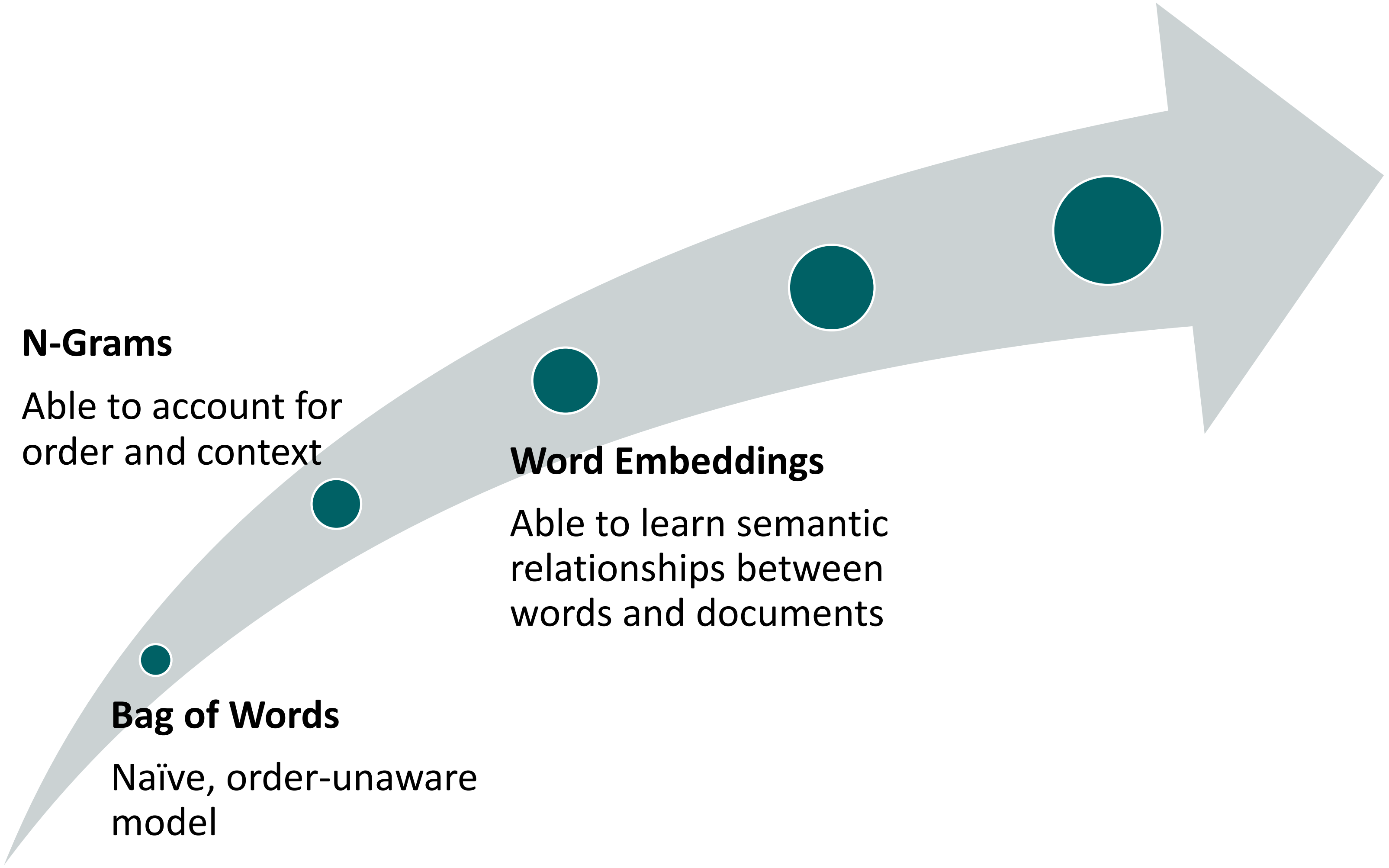
N-Grams

Able to account for order and context

Bag of Words

Naïve, order-unaware model

Outlook



N-Grams

Able to account for order and context

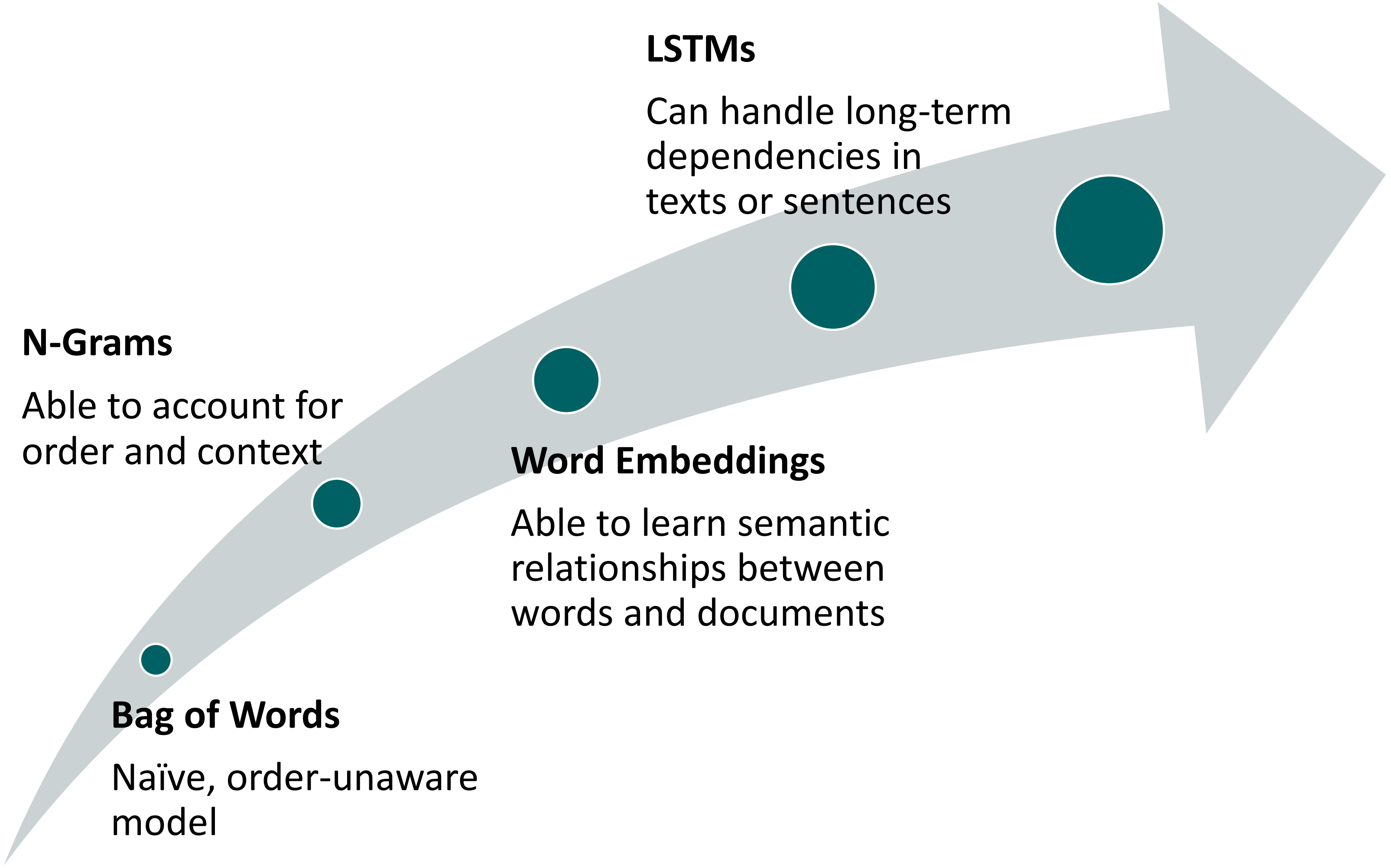
Word Embeddings

Able to learn semantic relationships between words and documents

Bag of Words

Naïve, order-unaware model

Outlook



LSTMs

Can handle long-term dependencies in texts or sentences

N-Grams

Able to account for order and context

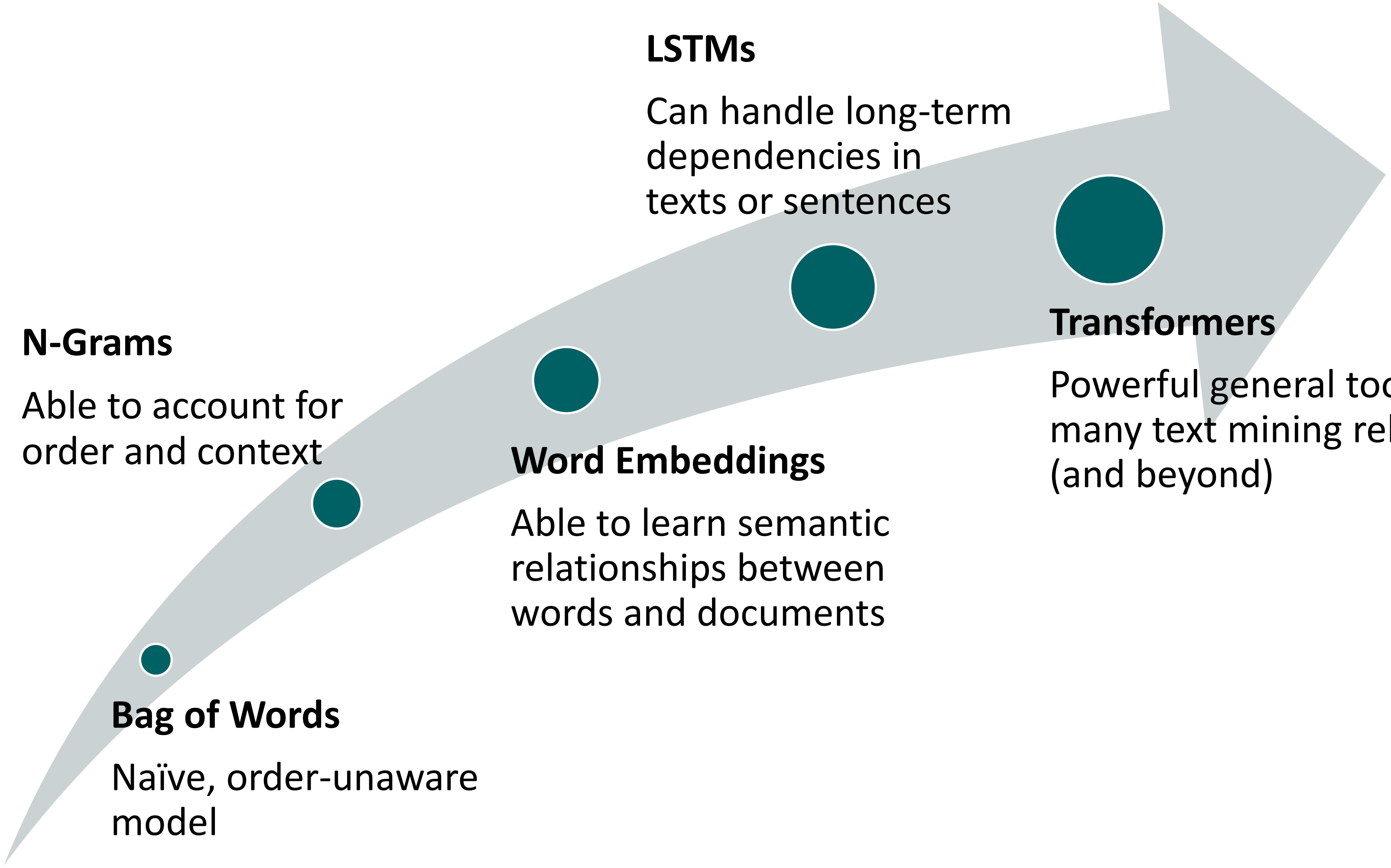
Word Embeddings

Able to learn semantic relationships between words and documents

Bag of Words

Naïve, order-unaware model

Outlook



Bag of Words

Naïve, order-unaware model

N-Grams

Able to account for order and context

Word Embeddings

Able to learn semantic relationships between words and documents

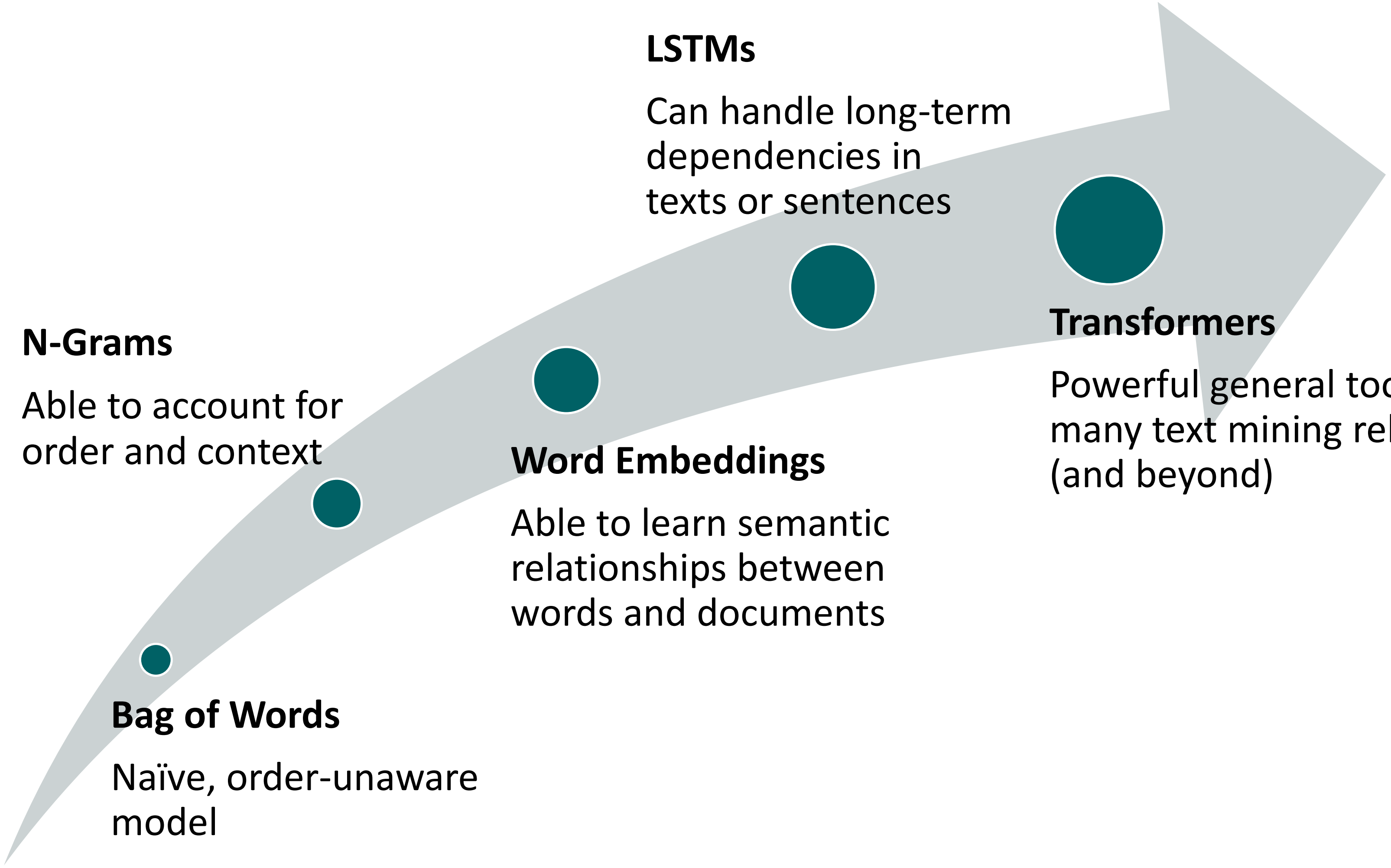
LSTMs

Can handle long-term dependencies in texts or sentences

Transformers

Powerful general tools for many text mining related tasks (and beyond)

Outlook



LSTMs

Can handle long-term dependencies in texts or sentences

N-Grams

Able to account for order and context

Transformers

Powerful general tools for many text mining related tasks (and beyond)

Word Embeddings

Able to learn semantic relationships between words and documents

Bag of Words

Naïve, order-unaware model

Next up: Responsible Data Science