Data Analysis and Machine Learning
Natural Language Processing

Robert Bagheri

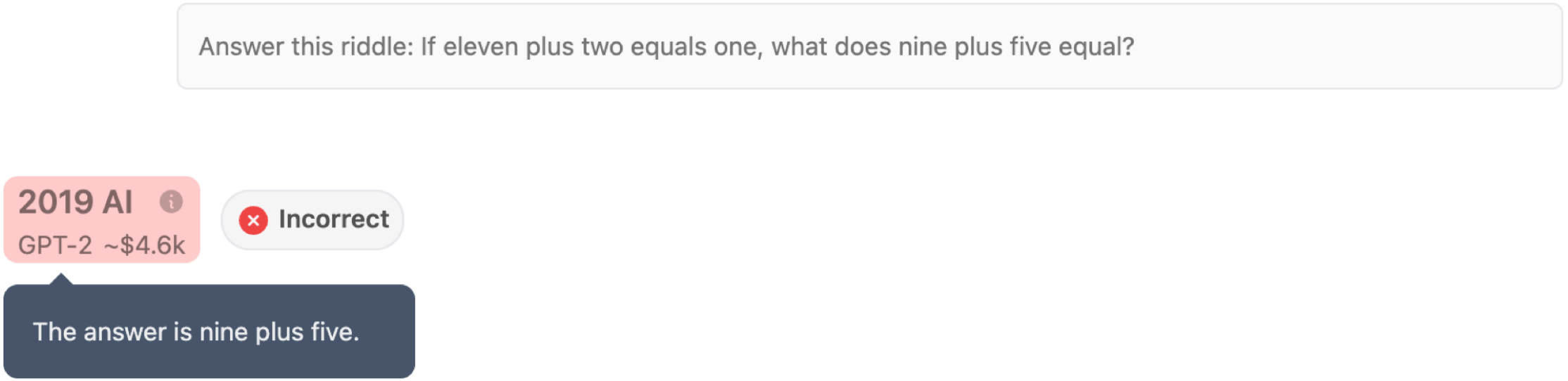
Department of Methodology, Statistics, and Data Science

Utrecht University

Answer this riddle

- If **eleven** plus **two** equals **one**, what does **nine plus five** equal?

NLP & Foundational Models



https://www.reddit.com/r/ChatGPT/comments/17h0hyr/i_made_a_site_where_you_can_ask_the_same_question/

Why NLP & text mining?

- **Text data is everywhere**, websites (e.g., news), social media (e.g., twitter), databases (e.g., doctors' notes), digital scans of printed materials, …
- A lot of world's data is in **unstructured text format**
- **Text is unstructured and high dimensional**
- Applications in industry: search, machine translation, sentiment analysis, question answering, …
- Applications in science: cognitive modeling, understanding bias in language, automated systematic literature reviews, …

Challenges with Text data

High dimensional data

- All possible words & phrases

Complex & subtle relationships in text

- “Instagram merges with Facebook.”
- “Instagram is bought by Facebook.”

Ambiguity & context sensitivity

- car = automobile = vehicle
- kapsalon (hairdresser) or kapsalon (fast food)

Homographs: same words can mean different things

- Apple, Bank, Bat (sports, animal, …)

Synonyms

Misspellings

Abbreviations

Negations

Spelling variations

LANGUAGE!

Regular Expressions

You may design a chatbot at the end of the ADS master, but we need to start with the basics!

Regular expressions (regex)

- **Powerful** and a **very useful** tool for text (pre)processing
- Used in pretty much every pipeline involving text
- Typical applications:
  - Extracting numbers, emails, IP-addresses, etc.
  - Validating text inputs in GUIs
  - Reformatting annoying incorrect dates (everything not yyyy-mm-dd)
  - Scrubbing names and addresses for pseudonimization
- **Cryptic** & takes a lot of practice!

Basic matching using the ‘re’ library

- Find all occurrences of a pattern in the string.

```
import re

text = "cat, dog, bat, elephant, rat, mouse, mat"

matches = re.findall(r"\b[a-z]at\b", text)

print("Matches:", matches)
# Output: ['cat', 'bat', 'rat', 'mat']
```

Basic matching using the ‘re’ library

- Replace a pattern with a new string.

```
import re

text = "I love cats, but my friend loves dogs."

result = re.sub(r"cat", "dog", text)

print("Result:", result)
# Output: I love dogs, but my friend loves dogs.
```

Basic matching using the ‘re’ library

- Replace a pattern with a new string.

```
import re

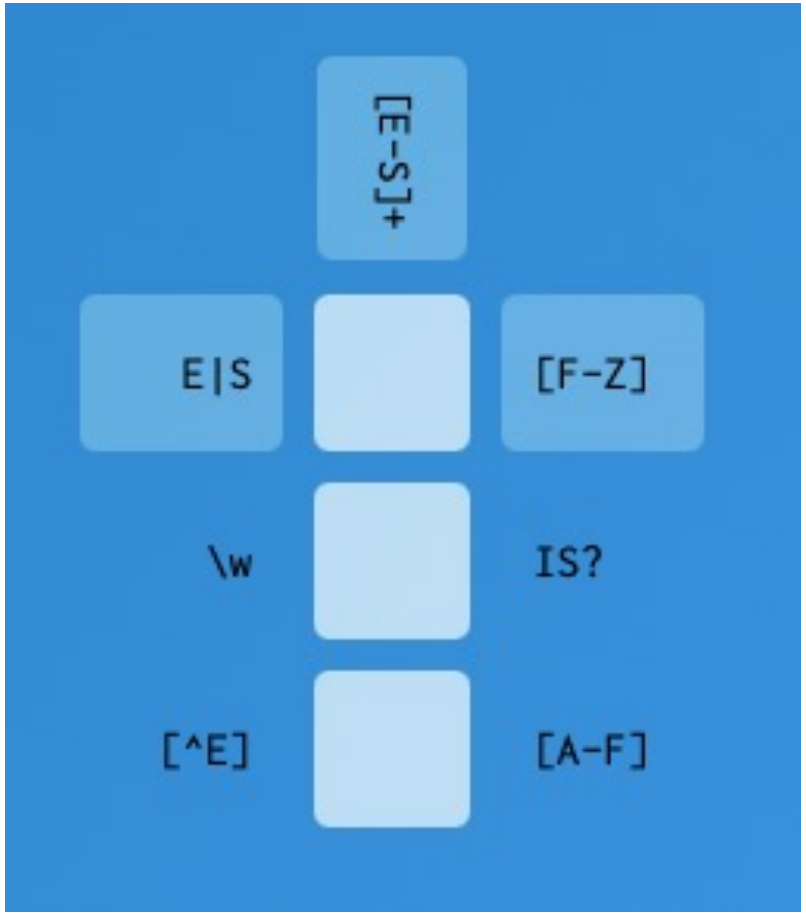
text = "I love cats, but my friend loves dogs."

result = re.sub(
    r"cats|dogs",
    lambda m: "dogs" if m.group() == "cats" else "cats",
    text
)

print("Result:", result)
# Output: I love dogs, but my friend loves cats.
```

Challenge problem: regex crossword

https://regexcrossword.com/



Text Preprocessing

Why preprocessing?

- Remove irrelevant information
- Add relevant information
- Convert your text into a suitable data format
- Allocate less memory to with lower dimensional data
- Improve performance

Typical preprocessing techniques

- Lowercasing (“Ruppert 22”→“ruppert 22”)
- Stopword removal (“text ming is the best!”)
- Punctuation removal (“text ming is the best!”)
- Number removal (“Ruppert 22”)
- Spell correction (“text ming is the best!”)
- Stemming (“running”→“run”) or Lemmatization (“were”→“is”)
- Tokenization (“text”, “ming”, “is”, “the”, “best” , “!”)
- And more..

Not all of these are appropriate at all times!

Tokenization

- Tokenization in Natural Language Processing (NLP) is the process of breaking down a piece of text into smaller units, called **tokens**.

- Tokens are the “units” of a text, words, phrases, sentences, subwords, characters, etc.

Word tokenization

- Splits text into individual words.
- This is the most common type of tokenization.
  - Example: "I love NLP!" → ["I", "love", "NLP", "!"]

- N-grams
  - N-grams are sequences of **n** consecutive tokens extracted from the text. These provide context by considering adjacent words together.
  - Example for bigrams (n=2): "I love NLP" → [("I", "love"), ("love", "NLP")]
  - Example for trigrams (n=3): "I love NLP" → [("I", "love", "NLP")]
  - N-grams are useful for capturing local word dependencies, commonly used in tasks like text classification, machine translation, and language modeling.

Other tokenization

Sentence Tokenization:

- Splits text into sentences
- Example: "I love NLP. It is amazing." → ["I love NLP.", "It is amazing."]

Subword Tokenization:

- Splits text into subword units
- Example: "unbelievable" → ["un", "believable"]

Character Tokenization:

- Splits text into individual characters
- Example: "NLP" → ["N", "L", "P"]

Purpose of tokenization

- Tokenization is a crucial preprocessing step because most NLP models work with tokens instead of raw text.
- Enables easier text manipulation.
- Compatibility with machine learning models.
- Handling of out-of-vocabulary (OOV) words through subword tokenization.
- Efficient encoding of textual data into numerical formats.

Text Representation

Vector Space Model

Basic idea

- Text is “unstructured data”
- How do we get to something structured that we can compute with?
- **Text must be represented somehow**
- Represent the text as something that makes sense to a computer

Vector space model

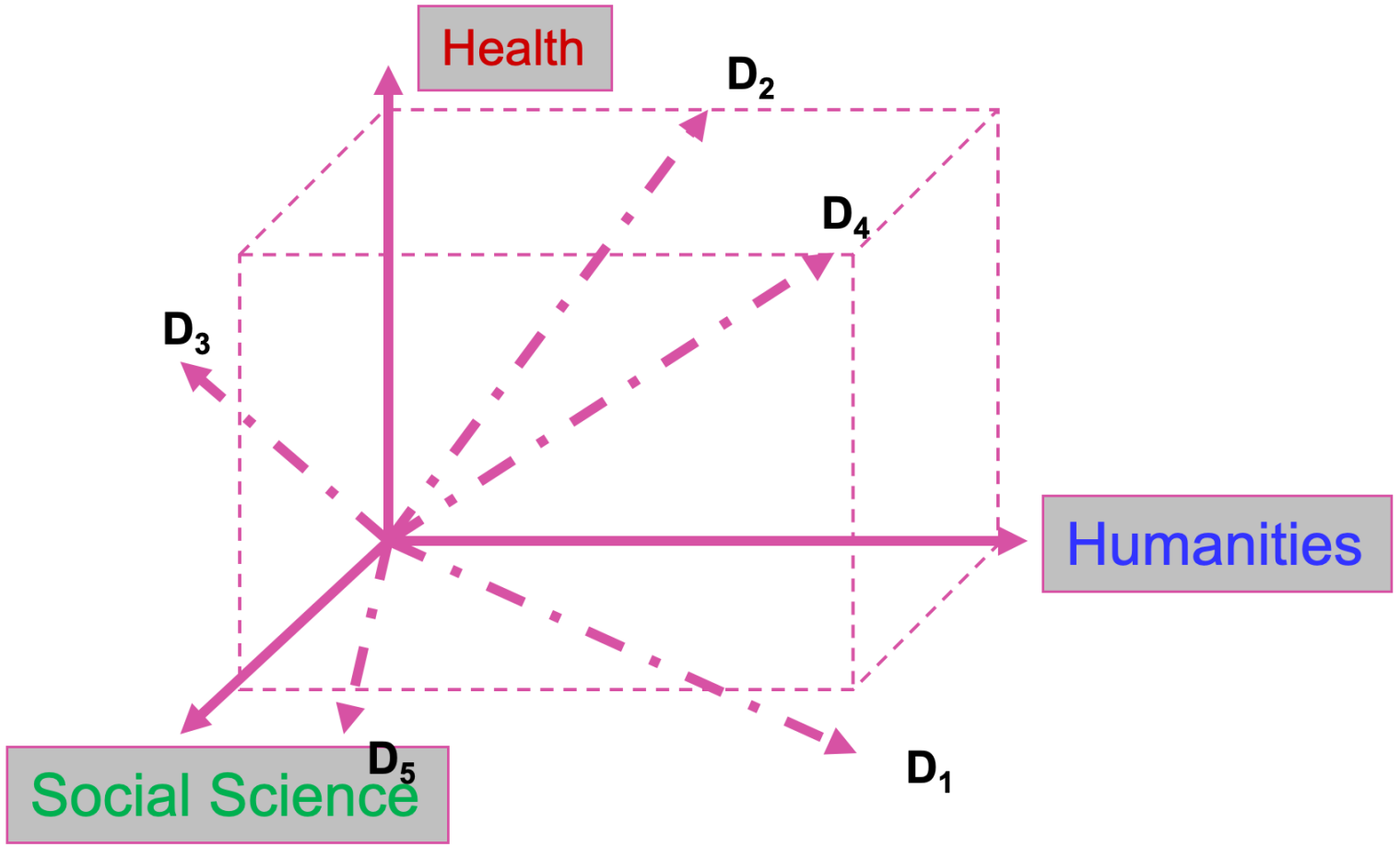
- A vector space is a collection of vectors
- Represent documents by concept vectors
  - Each concept defines one dimension
  - k concepts define a high-dimensional space
  - Element of vector corresponds to concept weight

Vector space model

- Concepts are tokens in text
- Documents are represented as vectors of tokens
- Each dimension (concept) corresponds to a separate token

$$d = (w_1, \ldots, w_n)$$

An illustration

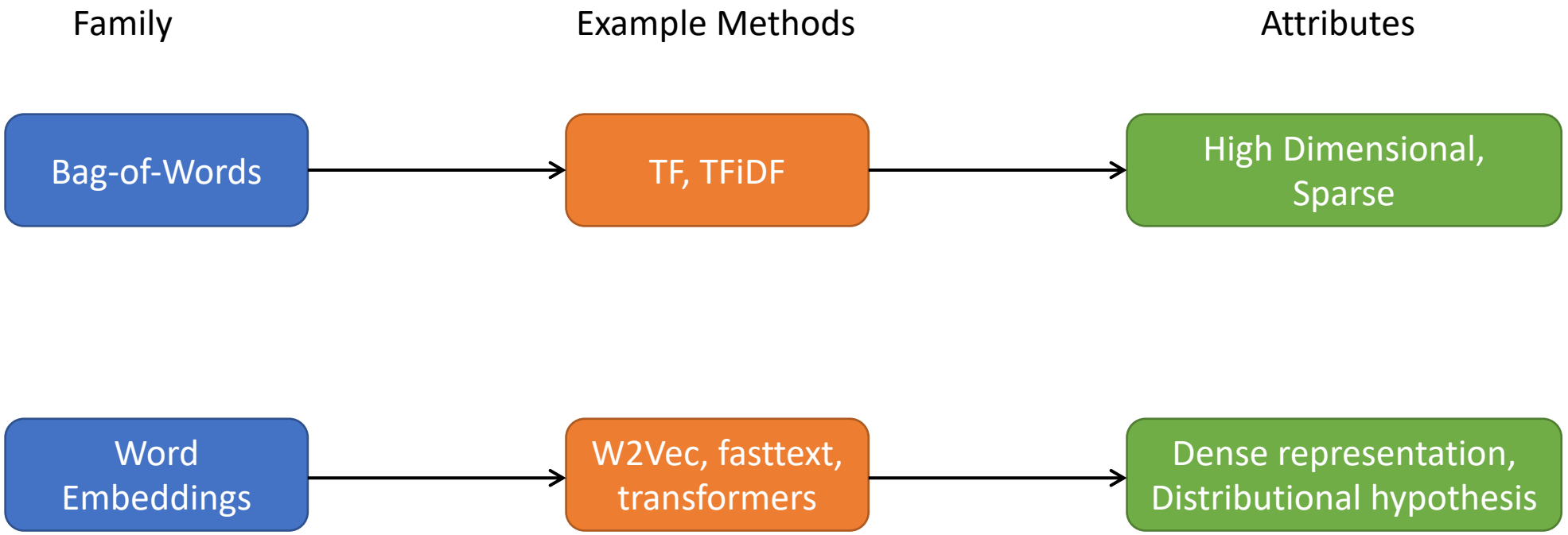


Vectorization

- The process of converting text into numbers is called **Vectorization**

- Distance between the vectors in this concept space
  - Relationship among document vectors

VSM representations



Bag-of-Words

- **Tokens** are words (more generally n-grams)
- **Weights** are number of occurrences of the terms in the document, example methods:
  - Binary
  - Term Frequency (TF)
  - Term Frequency inverse Document Frequency (TFiDF)

Example

Doc1: Text mining is to identify useful information.

Doc2: Useful information is mined from text.

Doc3: Apple is delicious.

Document-Term matrix (DTM):

	text	information	identify	mining	mined	is	useful	to	from	apple	delicious
Doc1	1	1	1	1	0	1	1	1	0	0	0
Doc2	1	1	0	0	1	1	1	0	1	0	0
Doc3	0	0	0	0	0	1	0	0	0	1	1

Term Frequency (TF)

- A measure to evaluate how often a word (or term) appears in a document relative to the total number of words in that document.
- TF helps in ranking terms based on their prominence within a document.
  - **Formula:**

$$TF(t) = \frac{\text{Number of times term } t \text{ appears in a document}}{\text{Total number of terms in the document}}$$

  - **Example:**
    - Document: `"I love NLP. NLP is amazing."`
    - Term: `"NLP"`
      - Frequency of `"NLP"` = 2
      - Total terms = 6
      - $TF(\text{NLP}) = \frac{2}{6} = 0.33$

TF-IDF

TF-IDF combines **TF** with **Inverse Document Frequency (IDF)** to assess how important a term is within a document relative to an entire corpus/dataset.

- **Formula:**

$$TF\text{-}IDF(t) = TF(t) \times IDF(t)$$

$$IDF(t) = \log\left(\frac{\text{Total number of documents}}{\text{Number of documents containing term } t}\right)$$

- **Example:**
  - Corpus: 3 documents:
    - Doc1: `"I love NLP"`
    - Doc2: `"NLP is amazing"`
    - Doc3: `"AI and NLP are evolving"`
  - Term: `"NLP"`
    - $TF(\text{NLP in Doc1}) = \frac{1}{3}$
    - $IDF(\text{NLP}) = \log\left(\frac{3}{3}\right) = 0$ (common in all documents, so it's less unique)
    - $TF\text{-}IDF(\text{NLP in Doc1}) = \frac{1}{3} \times 0 = 0$

What to do with the vectors?

- **We can check documents' similarity!**
  - **Cosine similarity**

$$\text{Cosine Similarity} = \frac{\mathbf{D}_1 \cdot \mathbf{D}_2}{\|\mathbf{D}_1\| \|\mathbf{D}_2\|}$$

Where:

- $\mathbf{D}_1$ and $\mathbf{D}_2$ are the term frequency (TF) or TF-IDF vectors for two documents.
- $\mathbf{D}_1 \cdot \mathbf{D}_2$: The dot product of the vectors.
- $\|\mathbf{D}_1\|$: Magnitude of the first document vector.
- $\|\mathbf{D}_2\|$: Magnitude of the second document vector.

What to do with the vectors?

Documents

- **Doc₁:** "I love NLP"
- **Doc₂:** "NLP is amazing"
- **Doc₃:** "I love programming"

Vocabulary: `[i, love, nlp, is, amazing, programming]`

Term Frequency (TF) Representation

- $D_1 = [1, 1, 1, 0, 0, 0]$
- $D_2 = [0, 0, 1, 1, 1, 0]$
- $D_3 = [1, 1, 0, 0, 0, 1]$

$$\cos(D_1, D_2) = \frac{1}{\sqrt{3} \times \sqrt{3}} = \boxed{0.333}$$

$$\cos(D_1, D_3) = \frac{2}{\sqrt{3} \times \sqrt{3}} = \boxed{0.667}$$

$$\cos(D_2, D_3) = \frac{0}{\sqrt{3} \times \sqrt{3}} = \boxed{0}$$

What to do with the vectors?

- **We can check documents' similarity!**
  - **Document Similarity**:
    - Used in search engines to rank documents based on their relevance to a query.
    - Example: Comparing a user's query with a corpus of documents to find the most relevant ones.
  - **Plagiarism Detection**:
    - Compares textual content to identify overlapping or similar phrases and ideas.
  - **Clustering and Classification**:
    - Groups similar documents or sentences in tasks like topic modeling and text categorization.
  - **Recommendation Systems**:
    - Finds content (e.g., articles, products) similar to a given input based on textual metadata.

Word Embedding

Word representations

How can we represent the meaning of words?

So, we can ask:

- How similar is cat to dog, or Paris to London?
- How similar is document A to document B?

Word as vectors

Can we represent words as vectors?

The vector representations should:

- capture semantics
  - similar words should be close to each other in the vector space
  - relation between two vectors should reflect the relationship between the two words
- be efficient (vectors with fewer dimensions are easier to work with)
- be interpretable

Word as vectors

How similar are the following two words? (not similar 0–10 very similar)

smart and **intelligent:**
easy and **big**:
easy and **difficult**:
hard and **difficult**:

Word as vectors

How similar are the following two words? (not similar 0–10 very similar)

smart and **intelligent:** **9.20**
easy and **big**: **1.12**
easy and **difficult**: **0.58**
hard and **difficult**: **8.77**

(SimLex-999 dataset, https://fh295.github.io/simlex.html)

Words as Vectors

One-hot encoding

Map each word to a unique identifier

e.g. cat (3) and dog (5).

- Vector representation: all zeros, except 1 at the ID

cat	0	0	1	0	0	0	0
dog	0	0	0	0	1	0	0
car	0	0	0	0	0	0	1

One-hot encoding

Map each word to a unique identifier

e.g. cat (3) and dog (5).

- Vector representation: all zeros, except 1 at the ID

cat	0	0	1	0	0	0	0
dog	0	0	0	0	1	0	0
car	0	0	0	0	0	0	1

What are limitations of one-hot encodings?

One-hot encoding

Map each word to a unique identifier

e.g. cat (3) and dog (5).

- Vector representation: all zeros, except 1 at the ID

cat	0	0	1	0	0	0	0
dog	0	0	0	0	1	0	0
car	0	0	0	0	0	0	1

Even related words have distinct vectors!

High number of dimensions

Distributional hypothesis: Words that occur in similar contexts tend to have similar meanings.

You shall know a word by the company it keeps.
(Firth, J. R. 1957:11)

Word vectors based on co-occurrences

documents as context
word-document matrix

	doc_1	doc_2	doc_3	doc_4	doc_5	doc_6	doc_7
cat	5	2	0	1	4	0	0
dog	7	3	1	0	2	0	0
car	0	0	1	3	2	1	1

Word vectors based on co-occurrences

documents as context
word-document matrix

	doc_1	doc_2	doc_3	doc_4	doc_5	doc_6	doc_7
cat	5	2	0	1	4	0	0
dog	7	3	1	0	2	0	0
car	0	0	1	3	2	1	1

neighboring words as context
word-word matrix

	cat	dog	car	bike	book	house	tree
cat	0	3	1	1	1	2	3
dog	3	0	2	1	1	3	1
car	0	0	1	3	2	1	1

Word vectors based on co-occurrences

There are many variants:

- Context (words, documents, which window size, etc.)
- Weighting (raw frequency, etc.)

Vectors are sparse: Many zero entries.

Therefore: Dimensionality reduction is often used (e.g., SVD)

These methods are sometimes called **count-based** methods as they work directly on **co-occurrence** counts.

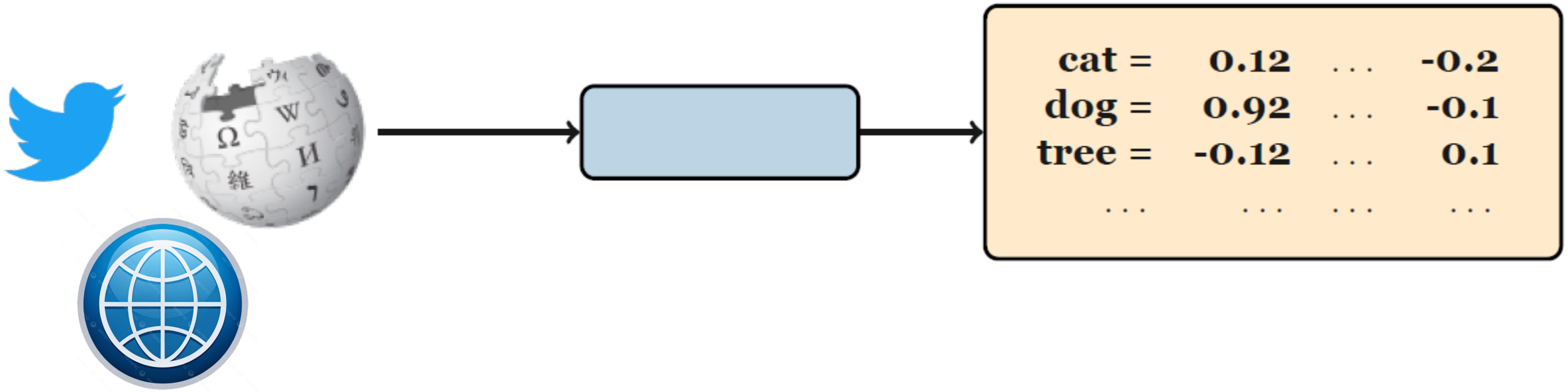
Word embeddings

- Vectors are short; typically 50-1024 dimensions ☺
- Vectors are dense (mostly non-zero values)
- Very effective for many NLP tasks ☺
- Individual dimensions are less interpretable ☹

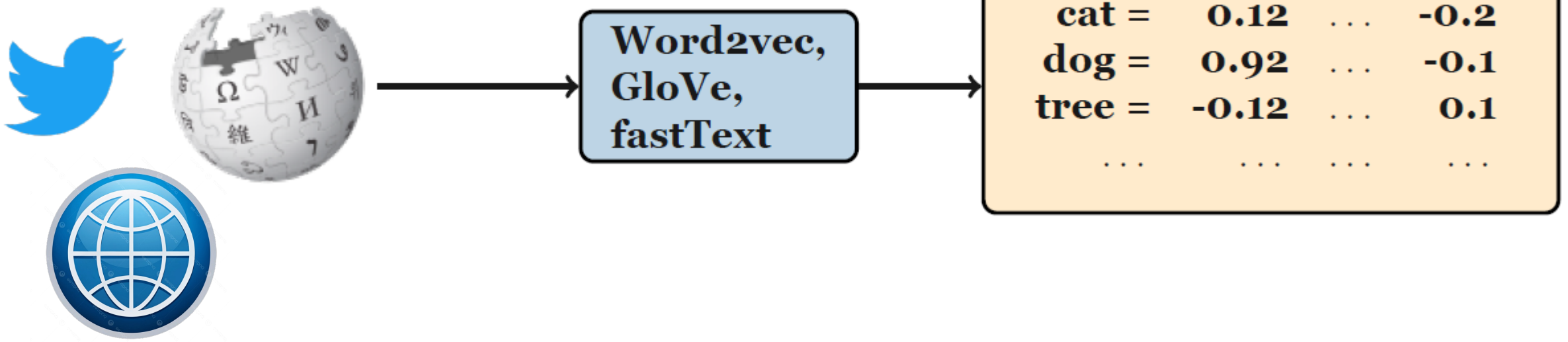
cat	0.52	0.48	-0.01	···	0.28
dog	0.32	0.42	-0.09	···	0.78

How do we learn word embeddings?

Learning word embeddings



Learning word embeddings



Training data for word embeddings

- Use **text itself** as training data for the model!
  - A form of self-supervision.

- Train a **classifier** (neural network, logistic regression, or SVM, etc.) to predict the next word given previous words.

Exercise: Word prediction task

Yesterday I went to the ?

A new study has highlighted the positive ?

Which word comes next?

Word2Vec

- Popular embedding method
- Very fast to train
- Idea: **predict** rather than **count**

- **https://projector.tensorflow.org/**

Word2Vec

The domestic **cat** is a small, typically furry carnivorous mammal

w_{-2} w_{-1} w_0 w_1 w_2 w_3 w_4 w_5

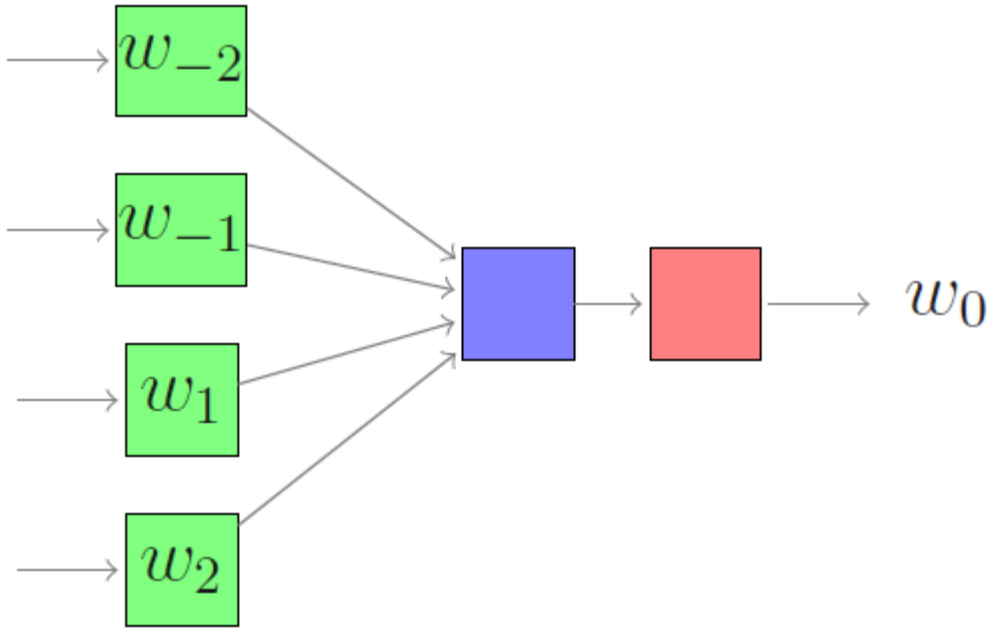
We have **target** words (cat) and **context** words (here: window size = 5).

Word2Vec

- Instead of **counting** how often each word w occurs near a target word
  - Train a classifier on a binary **prediction** task:
    - Is w likely to show up near target?
- We don’t actually care about this task
  - But we'll take the learned classifier weights as the word embeddings
- Big idea: **self-supervision**
  - A word c that occurs near target in the corpus as the gold "correct answer" for supervised learning
  - **No need for human labels**
  - Bengio et al. (2003); Collobert et al. (2011)

Word2Vec algorithms

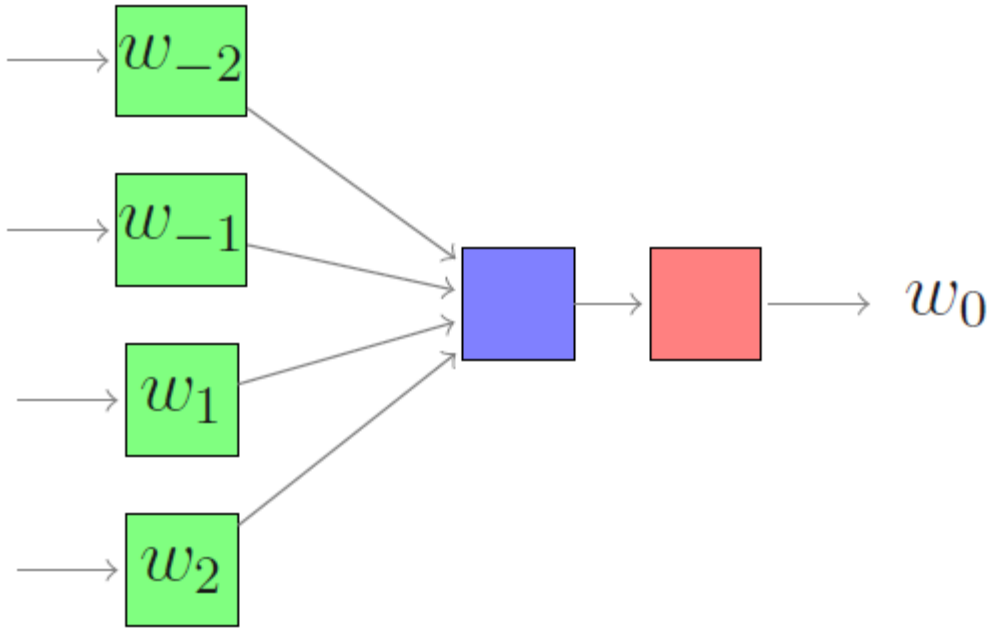
Continuous Bag-Of-Words (CBOW)



one snowy **?** she went

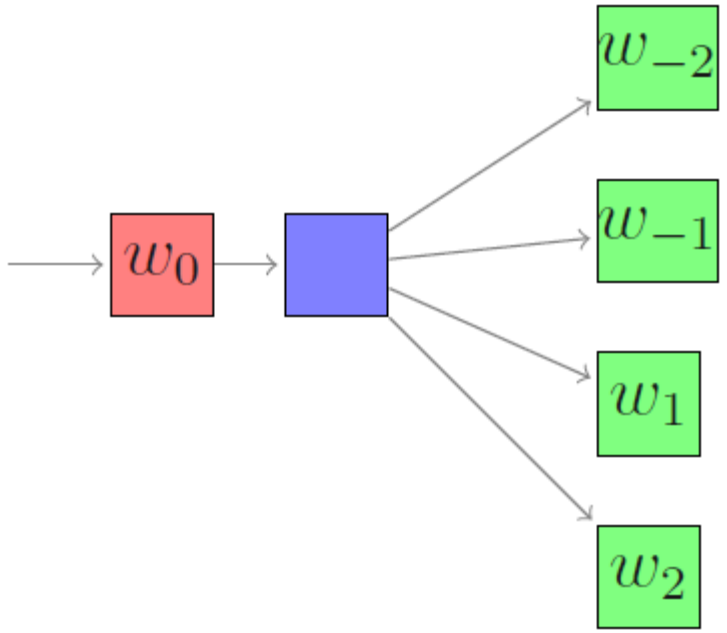
Word2Vec algorithms

Continuous Bag-Of-Words (CBOW)



one snowy ? she went

skipgram



? ? day ? ?

Skipgram overview

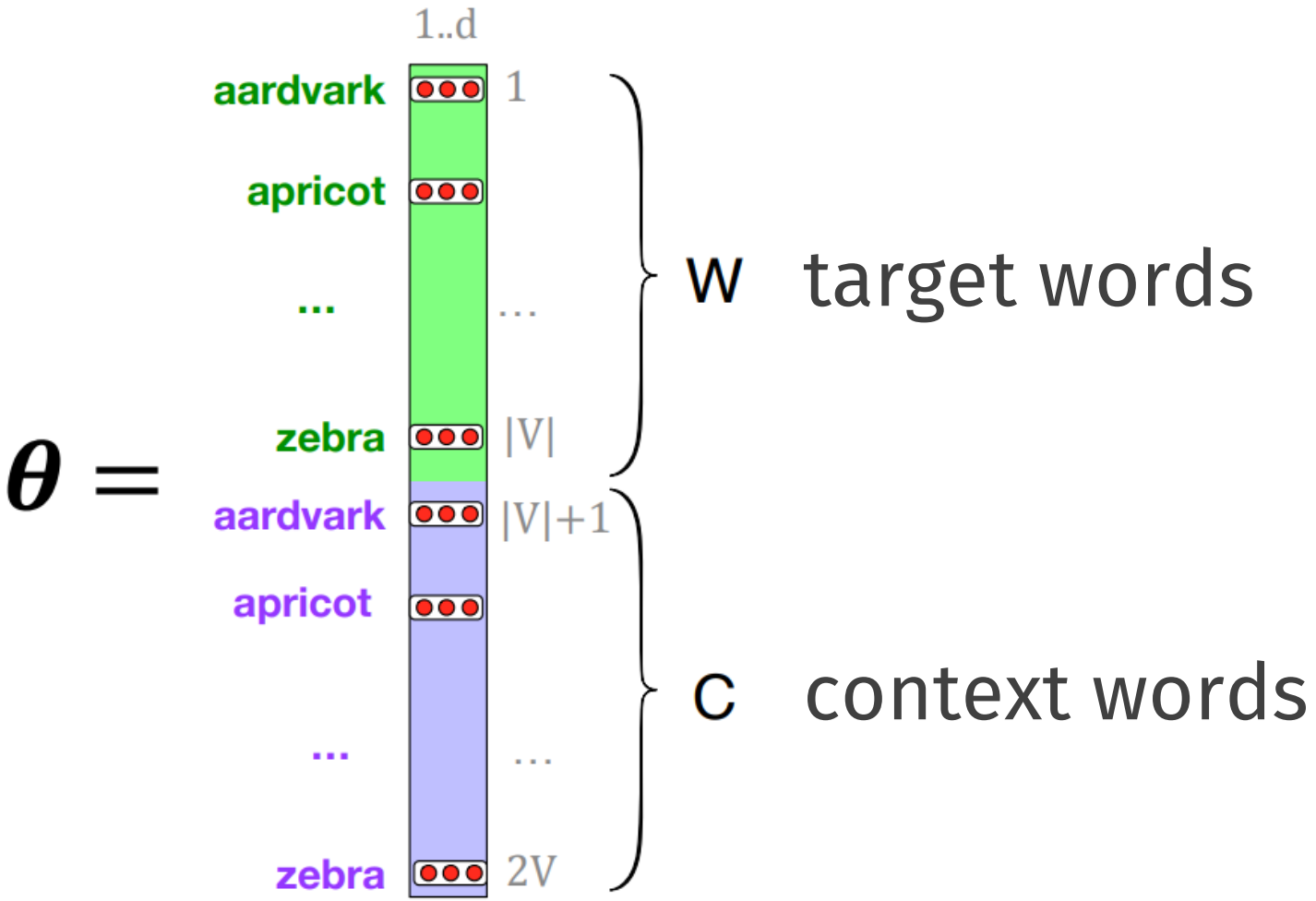
The domestic **cat** is a small, typically furry carnivorous mammal

1. **Create examples**
   - Positive examples: Target word and neighboring context
   - Negative examples: Target word and randomly sampled words from the lexicon (*negative sampling*)
2. Train a **logistic regression** model to distinguish between the positive and negative examples
3. The resulting **weights** are the embeddings!

word (w)	context (c)	label
cat	small	1
cat	furry	1
cat	car	0
...	...	...

Embedding vectors are essentially a byproduct!

Skipgram embeddings



Learning the classifier

- How to learn?
  - **Stochastic gradient descent!**

- SGNS learns two sets of embeddings
  - Target embeddings matrix W
  - Context embedding matrix C
- It's common to just add them together, representing word i as the vector Wi + Ci

Skipgram overview

1. Treat the target word t and a neighboring context word c as positive examples.
2. Randomly sample other words in the lexicon to get negative examples
3. Use logistic regression / NN to train a classifier to distinguish those two cases
4. Use the learned weights as the embeddings

Skipgram classifier

- A probabilistic classifier, given
  - a test target word w
  - its context window of L words $c_{1:L}$
- Estimates probability that w occurs in this window based on similarity of w (embeddings) to $c_{1:L}$ (embeddings).

- To compute this, we just need embeddings for all the words.

Pre-trained embeddings

- We want to build a system to **solve a task** (e.g., sentiment analysis)
  - Use pre-trained embeddings. Should we **fine-tune**?
    - Lots of data: yes
    - Just a small dataset: no

- **Analysis** (e.g., bias, semantic change)
  - Train embeddings from scratch

This is not the end!
The rest in the Transformers course!

Conclusion

- Text is everywhere
- Text Preprocessing: is the process of cleaning, normalizing, and structuring raw text data into a format suitable for analysis or input into NLP models.
- Vector Space Model is the common way for text representation.
- Cosine similarity is a fundamental metric in NLP, enabling the comparison of textual data.
- Word embedding is the connection to the new vsm models.

Next week

Utrecht University

Master Applied Data Science

Student Conference Day

INFOMDW & INFOMDAML

5 November 2026 9:00-17:00

Utrecht University

Coordinators



Hakim Qahtan
Course coordinator
INFOMDW



Robert Bagheri
Course coordinator
INFOMDAML

Keynote speakers



Arthur Hayen
S. Data Scientist
Menzis



Shihan Wang
Assistant professor
ICS – UU



Jelle Teijema
Data Scientist
Ministry of Defense



Emmeke Aarts
Associate professor
Social Sciences – UU

Practical
Representation for Text Classification, Deep Learning & Word Embedding

Questions?

Additional Slides

How to write regular expressions

1. Simple regex searches

RE	Example Patterns Matched
/woodchucks/	"interesting links to woodchucks and lemurs"
/a/	"Mary Ann stopped by Mona's"
/!/	"You've left the burglar behind again!" said Nori

2. Disjunction (OR)

The use of the brackets [] to specify a disjunction of characters:

RE	Match	Example Patterns
/[wW]oodchuck/	Woodchuck or woodchuck	"Woodchuck"
/[abc]/	'a', 'b', *or* 'c'	"In uomini, in soldati"
/[1234567890]/	any digit	"plenty of 7 to 5"

The pipe sympol | is also for disjunction:

Pattern	Matches
groundhog|woodchuck	
yours|mine	yours mine
a|b|c	= [abc]

3. Brackets and dash

The use of the brackets [] plus the dash - to specify a range:

RE	Match	Example Patterns Matched
/[A-Z]/	an upper case letter	"we should call it 'Drenched Blossoms' "
/[a-z]/	a lower case letter	"my beans were impatient to be hoed!"
/[0-9]/	a single digit	"Chapter 1: Down the Rabbit Hole"

4. Negation

The caret ^ for negation or just to mean ^:

RE	Match (single characters)	Example Patterns Matched
/[^A-Z]/	not an upper case letter	"Oyfn pripetchik"
/[^Ss]/	neither 'S' nor 's'	"I have no exquisite reason for't"
/[^.]/	not a period	"our resident Djinn"
/[e^]/	either 'e' or '^'	"look up ^ now"
/a^b/	the pattern 'a^b'	"look up a^b now"

5. Question and period marks

The question mark ? marks optionality of the previous expression:

RE	Match	Example Patterns Matched
/woodchucks?/	woodchuck or woodchucks	"woodchuck"
/colou?r/	color or colour	"color"

The use of the period . to specify any character:

RE	Match	Example Matches
/beg.n/	any character between *beg* and *n*	begin, beg'n, begun

6. Anchors

RE	Match
^	start of line
\$	end of line
\b	word boundary
\B	non-word boundary

7. Common sets

Aliases for common sets of characters:

RE	Expansion	Match	First Matches
\d	[0-9]	any digit	Party␣of␣5
\D	[^0-9]	any non-digit	Blue␣moon
\w	[a-zA-Z0-9_]	any alphanumeric/underscore	Daiyu
\W	[^\w]	a non-alphanumeric	!!!!
\s	[␣\r\t\n\f]	whitespace (space, tab)	
\S	[^\s]	Non-whitespace	in␣Concord

The backslash for escaping!

8. Operators for counting

Aliases for common sets of characters:

RE	Expansion	Match	First Matches
\d	[0-9]	any digit	Party␣of␣5
\D	[^0-9]	any non-digit	Blue␣moon
\w	[a-zA-Z0-9_]	any alphanumeric/underscore	Daiyu
\W	[^\w]	a non-alphanumeric	!!!!
\s	[␣\r\t\n\f]	whitespace (space, tab)	
\S	[^\s]	Non-whitespace	in␣Concord

The backslash for escaping!

9. Other

Some characters that need to be backslashed:

RE	Match	First Patterns Matched
*	an asterisk "*"	"K*A*P*L*A*N"
\.	a period "."	"Dr. Livingston, I presume"
\?	a question mark	"Why don't they come and lend a hand?"
\n	a newline	
\t	a tab	

10. Operator precedence

Parenthesis	()
Counters	* + ? {}
Sequences and anchors	the ^my end$
Disjunction	|

Skipgram training

Skipgram

The domestic **cat** is a small, typically furry carnivorous mammal

w_{-2} w_{-1} w_0 w_1 w_2 w_3 w_4 w_5

We have **target** words (cat) and **context** words (here: window size = 5).

The probability that c is a real context word, and the probability that c is not a real context word:

$$P(+|w, c)$$
$$P(-|w, c) = 1 - P(+|w, c)$$

Skipgram

Similarity is computed from dot product

- **Intuition**: A word *c* is likely to occur near the target *w* if its embedding is similar to the target embedding.

$$\approx w \cdot c$$

  - Two vectors are similar if they have a high dot product
  - Cosine similarity is just a normalized dot product

Turn this into a probability using the sigmoid function:

$$P(+|w,c) = \sigma(c \cdot w) = \frac{1}{1+\exp(-c \cdot w)}$$

$$P(-|w,c) = 1 - P(+|w,c)$$

$$= \sigma(-c \cdot w) = \frac{1}{1+\exp(c \cdot w)}$$

How Skipgram classifier computes P(+|w, c)

$$P(+|w,c) = \sigma(c \cdot w) = \frac{1}{1+\exp(-c \cdot w)}$$

This is for one context word, but we have lots of context words. We'll assume independence and just multiply them:

$$P(+|w,c_{1:L}) = \prod_{i=1}^{L} \sigma(c_i \cdot w)$$

$$\log P(+|w,c_{1:L}) = \sum_{i=1}^{L} \log \sigma(c_i \cdot w)$$

Word2vec: how to learn vectors

- Given the set of positive and negative training instances, and an initial set of embedding vectors
- The goal of learning is to adjust those word vectors such that we:
- **Maximize** the similarity of the **target word**, **context word** pairs (w , c_{pos}) drawn from the positive data
- **Minimize** the similarity of the (w , c_{neg}) pairs drawn from the negative data.

Loss function for one *w* with c_{pos} , c_{neg1} ...c_{negk}

- Maximize the similarity of the target with the actual context words, and minimize the similarity of the target with the k negative sampled non-neighbor words.

$$
\begin{aligned}
L_{CE} &= -\log\left[P(+|w,c_{pos})\prod_{i=1}^{k}P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos}) + \sum_{i=1}^{k}\log P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos}) + \sum_{i=1}^{k}\log\left(1-P(+|w,c_{neg_i})\right)\right] \\
&= -\left[\log \sigma(c_{pos}\cdot w) + \sum_{i=1}^{k}\log\sigma(-c_{neg_i}\cdot w)\right]
\end{aligned}
$$

Learning the classifier

- How to learn?
  - **Stochastic gradient descent!**