Fast and Flexible Example-Based Treebank Search with Vector Symbolic Architectures

Adam Roussel

Ruhr University Bochum

2026-05-11, SLiDE @ LREC

Overview

Motivation

- Exploration of treebanks, looking for similar constructions
  - A degree of fuzziness is advantageous
  - If query too narrow, miss interesting cases

Motivation

- Exploration of treebanks, looking for similar constructions
  - A degree of fuzziness is advantageous
  - If query too narrow, miss interesting cases
- Unknown how given construction annotated in corpus

Motivation

- Exploration of treebanks, looking for similar constructions
  - A degree of fuzziness is advantageous
  - If query too narrow, miss interesting cases
- Unknown how given construction annotated in corpus
- Variety of query languages and search tools

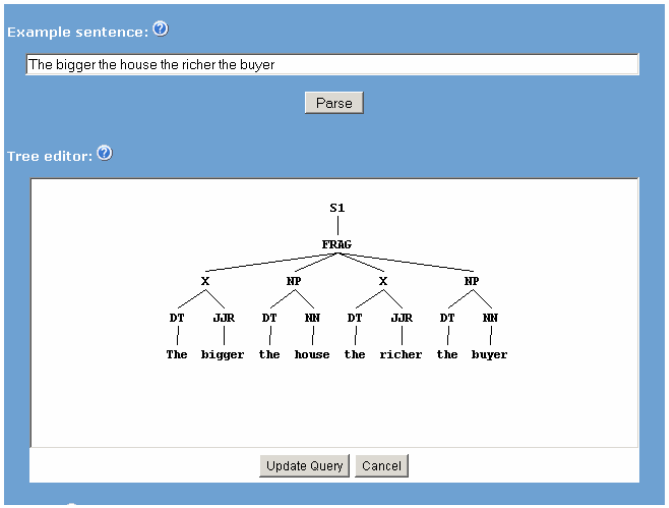
Example-based search: Procedure

Example-based search (Resnik & Elkiss 2005; Augustinus et al. 2012):

Example-based search: Procedure

Example-based search (Resnik & Elkiss 2005; Augustinus et al. 2012):

1. Automatic parser parses the example to provide a preliminary analysis.

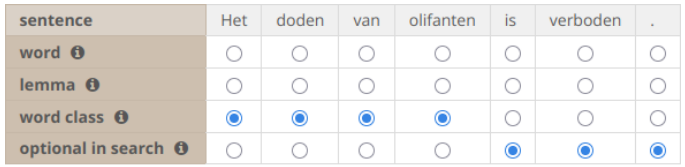


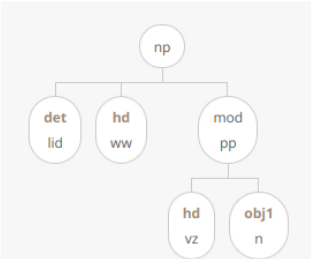
Linguist's Search Engine (LSE)

Example-based search: Procedure

Example-based search (Resnik & Elkiss 2005; Augustinus et al. 2012):

1. Automatic parser parses the example to provide a preliminary analysis.
2. Irrelevant parts of the analysis can be removed from the analysis to broaden the query.



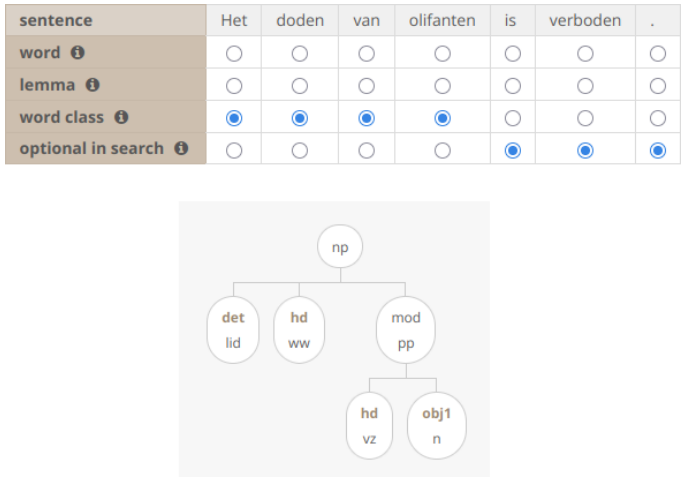


GrETEL

Example-based search: Procedure

Example-based search (Resnik & Elkiss 2005; Augustinus et al. 2012):

1. Automatic parser parses the example to provide a preliminary analysis.
2. Irrelevant parts of the analysis can be removed from the analysis to broaden the query.
3. A query is generated on the basis of the selected subtree

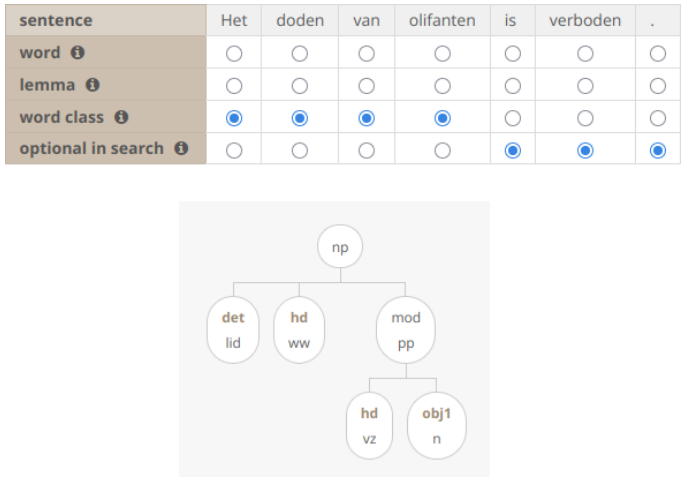


GrETEL

Example-based search: Procedure

Example-based search (Resnik & Elkiss 2005; Augustinus et al. 2012):

1. Automatic parser parses the example to provide a preliminary analysis.
2. Irrelevant parts of the analysis can be removed from the analysis to broaden the query.
3. A query is generated on the basis of the selected subtree
4. A treebank is searched with this query and the results are presented to the user.



GrETEL

Why VSAs?

With vector symbolic architectures (VSA) / hyper-dimensional computing (HDC):

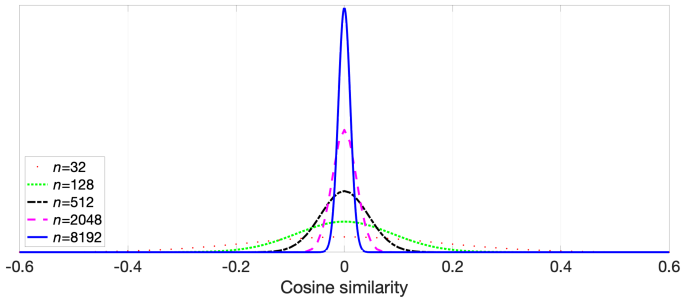
- Possible to create vector representations of complex data structures
  - These representations can be analysed and manipulated systematically.
  - Using structured data in contexts that require vectors
- Computationally lightweight
- Allows for graded similarity and robustness as well as structures that are transparent and linguistically meaningful
  - "One can have both" (Karlgren 2023)

Overview

High dimensionality and randomness

Properties of high-dimensional vector spaces:

- Huge number of nearly orthogonal vectors in high-dimensional spaces

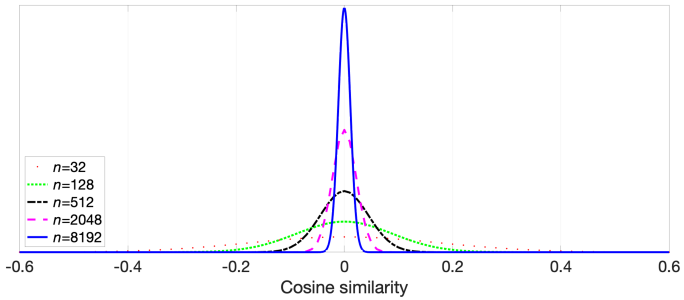


(Figure from Kleyko et al. (2022).)

High dimensionality and randomness

Properties of high-dimensional vector spaces:

- Huge number of nearly orthogonal vectors in high-dimensional spaces
- Any new symbol $\Rightarrow$ New random vector

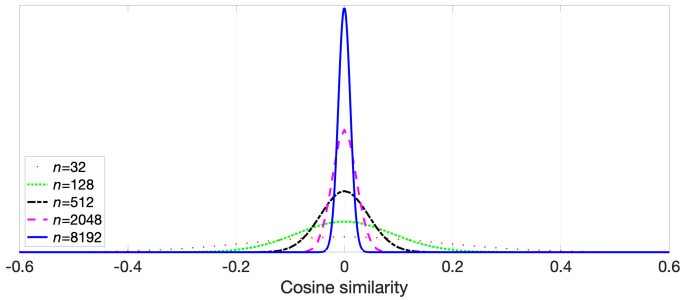


(Figure from Kleyko et al. (2022).)

High dimensionality and randomness

Properties of high-dimensional vector spaces:

- Huge number of nearly orthogonal vectors in high-dimensional spaces
- Any new symbol $\Rightarrow$ New random vector
- Near-orthogonality distinguishes one symbolic vector from another

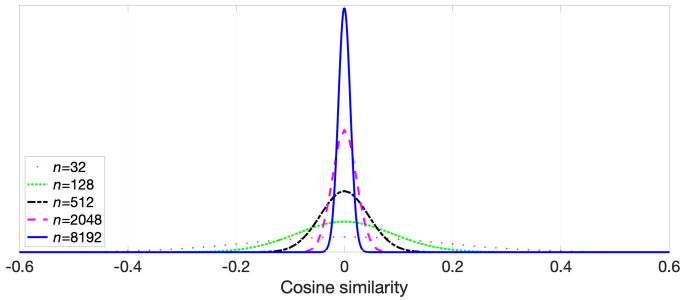


(Figure from Kleyko et al. (2022).)

High dimensionality and randomness

Properties of high-dimensional vector spaces:

- Huge number of nearly orthogonal vectors in high-dimensional spaces
- Any new symbol $\Rightarrow$ New random vector
- Near-orthogonality distinguishes one symbolic vector from another
- Greater similarity than chance $\Rightarrow$ "matching" in some way



(Figure from Kleyko et al. (2022).)

Kinds of VSAs

A VSA consists of:

1. A high-dimensional vector space

Kinds of VSAs

A VSA consists of:

1. A high-dimensional vector space
2. A measure of similarity

Kinds of VSAs

A VSA consists of:

1. A high-dimensional vector space
2. A measure of similarity
3. Appropriate operations on those vectors

Kinds of VSAs

A VSA consists of:

1. A high-dimensional vector space
2. A measure of similarity
3. Appropriate operations on those vectors

The variant here, "binary spatter code" or BSC (Kanerva 1996):

- Binary vectors, roughly equal number of 0s and 1s
  - Bitpacking to save memory
- Similarity measure: Hamming distance (scaled and centered)

Basic operations

- **bundling** $(+)$: Results in a vector that is similar to its inputs, $a + b \approx a \approx b$

Basic operations

- **bundling** $(+)$: Results in a vector that is similar to its inputs, $a + b \approx a \approx b$
  - Here: Majority rule, i.e. keep the bits that are shared by most of the inputs

Basic operations

- **bundling** ($+$): Results in a vector that is similar to its inputs, $a + b \approx a \approx b$
  - Here: Majority rule, i.e. keep the bits that are shared by most of the inputs
- **binding** ($\otimes$): Results in a vector dissimilar to its inputs, $a \otimes b \not\approx a$

Basic operations

- **bundling** ($+$): Results in a vector that is similar to its inputs, $a + b \approx a \approx b$
  - Here: Majority rule, i.e. keep the bits that are shared by most of the inputs
- **binding** ($\otimes$): Results in a vector dissimilar to its inputs, $a \otimes b \not\approx a$
  - Here: XOR, 0 when same and 1 when different

Basic operations

- **bundling** ($+$): Results in a vector that is similar to its inputs, $a + b \approx a \approx b$
  - Here: Majority rule, i.e. keep the bits that are shared by most of the inputs
- **binding** ($\otimes$): Results in a vector dissimilar to its inputs, $a \otimes b \not\approx a$
  - Here: XOR, 0 when same and 1 when different
- **permutation** (ρ^n): Results in a vector dissimilar to its input, $a \not\approx \rho^1 a \not\approx \rho^2 a$

Basic operations

- **bundling** ($+$): Results in a vector that is similar to its inputs, $a + b \approx a \approx b$
  - Here: Majority rule, i.e. keep the bits that are shared by most of the inputs
- **binding** ($\otimes$): Results in a vector dissimilar to its inputs, $a \otimes b \not\approx a$
  - Here: XOR, 0 when same and 1 when different
- **permutation** (ρ^n): Results in a vector dissimilar to its input, $a \not\approx \rho^1 a \not\approx \rho^2 a$
  - Here: Every value with index i shifted to $i + n$

Data structures

Encoding key–value structures with a combination of binding and bundling:

$$\begin{bmatrix} \text{form} : \text{“reads”} \\ \text{upos} : \text{VERB} \end{bmatrix} = v_{\text{form}} \otimes v_{\text{reads}} + v_{\text{upos}} \otimes v_{\text{verb}} \tag{1}$$

Data structures

Encoding key–value structures with a combination of binding and bundling:

$$\begin{bmatrix} \text{form} : \text{“reads”} \\ \text{upos} : \text{VERB} \end{bmatrix} = v_{\text{form}} \otimes v_{\text{reads}} + v_{\text{upos}} \otimes v_{\text{verb}} \tag{1}$$

Unbinding (same operation in BSC) to retrieve the information from such a vector:

$$v_{\text{upos}} \oslash \left(v_{\text{upos}} \otimes v_{\text{verb}} \right) = v_{\text{verb}} \tag{2}$$

$$v_{\text{upos}} \oslash \left(v_{\text{form}} \otimes v_{\text{reads}} + v_{\text{upos}} \otimes v_{\text{verb}} \right) \approx v_{\text{verb}} \tag{3}$$

Overview

An example

(4) The bookworm reads.

$$\begin{bmatrix} \text{form} : \text{“reads”} \\ \text{upos} : \text{VERB} \\ \text{nsubj} : \begin{bmatrix} \text{form} : \text{“bookworm”} \\ \text{upos} : \text{NOUN} \\ \text{det} : \begin{bmatrix} \text{form} : \text{“the”} \\ \text{upos} : \text{DET} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Encoding strategies

Terminal nodes:

$$
\begin{aligned}
R &= (\rho^1 v_{\text{form}} \otimes v_{\text{reads}} + \rho^1 v_{\text{upos}} \otimes v_{\text{VERB}}) \\
W &= (\rho^1 v_{\text{form}} \otimes v_{\text{bookworm}} + \rho^1 v_{\text{upos}} \otimes v_{\text{NOUN}}) \\
T &= (\rho^1 v_{\text{form}} \otimes v_{\text{the}} + \rho^1 v_{\text{upos}} \otimes v_{\text{DET}})
\end{aligned}
\tag{5}
$$

Strategy 1: Thin

$$\begin{bmatrix} \text{form} : \text{“reads”} \\ \text{upos} : \text{VERB} \\ \text{nsubj} : \begin{bmatrix} \text{form} : \text{“bookworm”} \\ \text{upos} : \text{NOUN} \\ \text{det} : \begin{bmatrix} \text{form} : \text{“the”} \\ \text{upos} : \text{DET} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\begin{aligned} & R+ \\ & \rho^2 v_{\text{nsubj}} \otimes W+ \\ & \rho^3 v_{\text{nsubj}} \otimes \rho^2 v_{\text{det}} \otimes T \end{aligned} \tag{6}$$

Each element occurs only once, at its own level.

Strategy 2: Subtree

$$\begin{bmatrix} \text{form} : \text{“reads”} \\ \text{upos} : \text{VERB} \\ \text{nsubj} : \begin{bmatrix} \text{form} : \text{“bookworm”} \\ \text{upos} : \text{NOUN} \\ \text{det} : \begin{bmatrix} \text{form} : \text{“the”} \\ \text{upos} : \text{DET} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\begin{aligned} & R+ \\ & \rho^2 v_{\text{nsubj}} \otimes W + W+ \\ & \rho^3 v_{\text{nsubj}} \otimes \rho^2 v_{\text{det}} \otimes T+ \\ & \boxed{\rho^2 v_{\text{det}} \otimes T}+ \\ & \boxed{T} \end{aligned} \tag{7}$$

Each element occurs at the top level, at its own level, and all intervening levels.

Strategy 3: Flat

$$\begin{bmatrix} \text{form} : \text{“reads”} \\ \text{upos} : \text{VERB} \\ \text{nsubj} : \begin{bmatrix} \text{form} : \text{“bookworm”} \\ \text{upos} : \text{NOUN} \\ \text{det} : \begin{bmatrix} \text{form} : \text{“the”} \\ \text{upos} : \text{DET} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\begin{aligned} & R + \\ & \rho^2 v_{\text{nsubj}} \otimes W + \\ & \boxed{\rho^2 v_{\text{det}} \otimes T} + \\ & \boxed{\rho^3 v_{\text{nsubj}} \otimes T} \end{aligned} \tag{8}$$

Similar to Thin, but the intervening levels are decoupled, so that they may match independently.

Overview

The procedure

1. Encode each sentence in a treebank according to one of these methods and store all of those vectors.

The procedure

1. Encode each sentence in a treebank according to one of these methods and store all of those vectors.
2. Accept a query phrase or sentence from the user.

The procedure

1. Encode each sentence in a treebank according to one of these methods and store all of those vectors.
2. Accept a query phrase or sentence from the user.
3. Parse the query with a parser that matches the treebank.

The procedure

1. Encode each sentence in a treebank according to one of these methods and store all of those vectors.
2. Accept a query phrase or sentence from the user.
3. Parse the query with a parser that matches the treebank.
4. Encode this parse into vector, the same way we did with the sentences.

The procedure

1. Encode each sentence in a treebank according to one of these methods and store all of those vectors.
2. Accept a query phrase or sentence from the user.
3. Parse the query with a parser that matches the treebank.
4. Encode this parse into vector, the same way we did with the sentences.
5. Then compare this query vector with all of the treebank vectors that we have stored. Then we can ask which sentences are nearest to the query vector.

Overview

Example 1: NP with PP

Using Lassy Small corpus and this example ("NP with PP") from Augustinus et al. (2012):

(9) Het doden van olifanten is verboden.
'The killing of elephants is prohibited.'

Example 1: NP with PP

Using Lassy Small corpus and this example ("NP with PP") from Augustinus et al. (2012):

(9) Het doden van olifanten is verboden.
'The killing of elephants is prohibited.'

0.40	Het nalaten van **het identificeren van risicogroepen** kan tot ongewenste gezondheidsrisico's leiden. 'The cessation of the identification of at-risk groups can lead to unwanted health risks.'
0.39	**Het houden van gedomesticeerde dieren** heet veeteelt. 'The keeping of domesticated animals is called husbandry.'
0.39	**Het stellen van confronterende vragen** leidt tot spanningen. 'The posing of confrontational questions leads to tensions.'
0.39	**Het erkennen van fouten** door de NS-directie is geen alledaags verschijnsel. 'The acknowledgment of errors by the NS management is no everyday occurrence.'

Example 2: Shell nouns

A shell noun example:

(10) De hoofdzaak is dat je beter wordt
'the main thing is that you get better'

Example 2: Shell nouns

A shell noun example:

(10) De hoofdzaak is dat je beter wordt
'the main thing is that you get better'

0.17	De enige **waarheid** is dat de journalisten logen. 'The only truth is that the journalists lied.'
0.15	Het is de **bedoeling** dat de maatregel afschrikkend werkt. 'It is the goal that the measure works as a deterrent.'
0.14	Het **voordeel** daarvan is dat je de zaken op een onafhankelijke manier kunt bekijken. 'The advantage of that is that you can view things in an independent manner.'
0.14	De **verwachting** is dat de Italiaanse troepen zeker tot 2005 zullen blijven. 'The expectation is that the Italian troops will stay at least until 2005.'

A more systematic evaluation

Intuition: Exact matches should rank high

But not *highest*, e.g.:

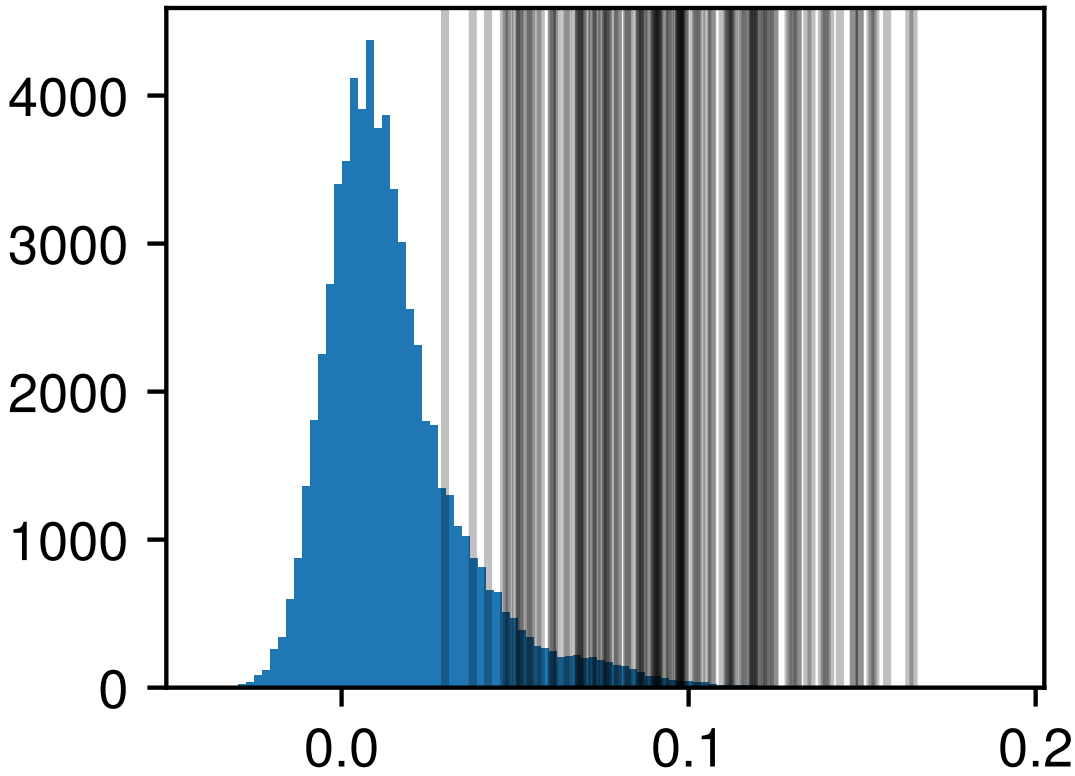
- All features count
- Features can "match" multiple times
- Noise

A more systematic evaluation

Intuition: Exact matches should rank high

But not *highest*, e.g.:

- All features count
- Features can "match" multiple times
- Noise

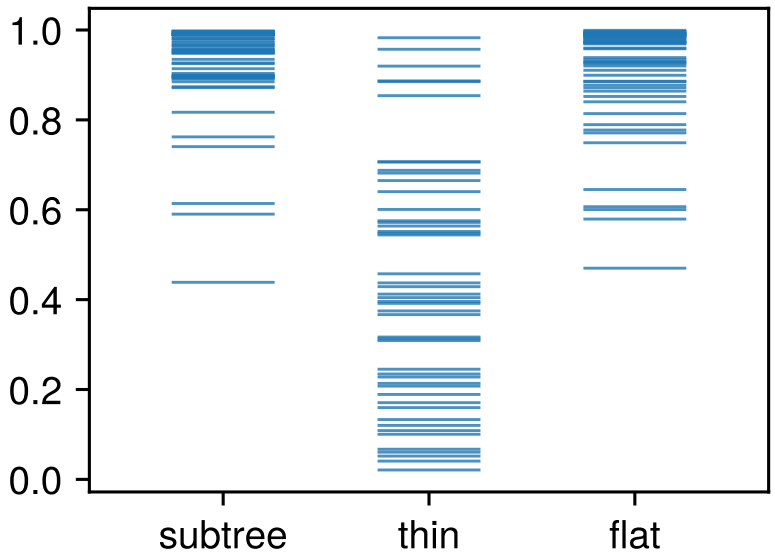


Ranking exact matches

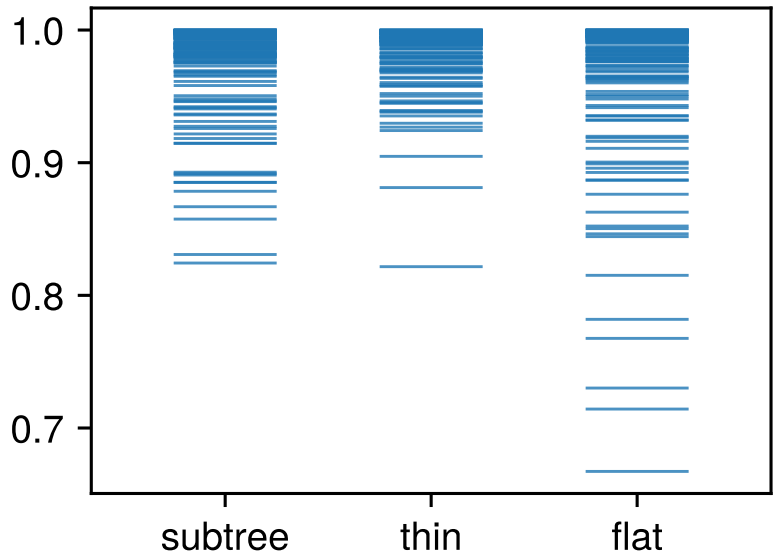
Strategy	NP with PP PR	Shell nouns PR
Subtree	91.4 ± 11.3	97.3 ± 3.7
Thin	42.7 ± 27.1	98.3 ± 2.6
Flat	89.3 ± 12.7	96.1 ± 6.2

Table: Average percentile rank of exact hits returned by GrETEL for both queries, according to each encoding strategy.

Percentile rank of exact matches



NP with PP



Shell noun

Overview

Conclusion

- Recap:
  - Simple and lightweight approach to treebank search
  - Example-based (also possible to specify queries)

Conclusion

- Recap:
  - Simple and lightweight approach to treebank search
  - Example-based (also possible to specify queries)
- Discovering relevant *patterns* in corpora
  - Schmid (2000, p. 56): "However, it is not possible to use the corpus as a heuristic device for finding patterns which do not strictly answer the precise queries."

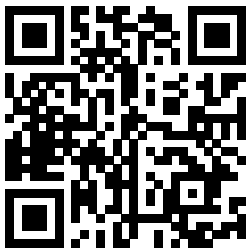
Conclusion

- Recap:
  - Simple and lightweight approach to treebank search
  - Example-based (also possible to specify queries)
- Discovering relevant *patterns* in corpora
  - Schmid (2000, p. 56): "However, it is not possible to use the corpus as a heuristic device for finding patterns which do not strictly answer the precise queries."
- Prospectively: Integrate different kinds of information into search
  - Combining syntactic representations and embeddings, e.g.: Looking for sentences where a particular adjective is used to describe a certain kind of thing

Conclusion

- Recap:
  - Simple and lightweight approach to treebank search
  - Example-based (also possible to specify queries)
- Discovering relevant *patterns* in corpora
  - Schmid (2000, p. 56): "However, it is not possible to use the corpus as a heuristic device for finding patterns which do not strictly answer the precise queries."
- Prospectively: Integrate different kinds of information into search
  - Combining syntactic representations and embeddings, e.g.: Looking for sentences where a particular adjective is used to describe a certain kind of thing

Link to the repository for this paper:



References I


Augustinus, Liesbeth, Vincent Vandeghinste, and Frank Van Eynde (2012). "Example-Based Treebank Querying." In: *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*. Ed. by Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Mehmet Uğur Doğan, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis. Istanbul, Turkey: European Language Resources Association (ELRA), pp. 3161–3167 (cit. on pp. 6–10, 42–43).


Kanerva, Pentti (1996). "Binary spatter-coding of ordered K-tuples." In: *Artificial Neural Networks – ICANN 96*. Springer Berlin Heidelberg, pp. 869–873. DOI: 10.1007/3-540-61510-5_146 (cit. on pp. 17–20).


Karlgren, Jussi (2023). "High-dimensional vector spaces can accommodate constructional features quite conveniently." In: *Proceedings of the First International Workshop on Construction Grammars and NLP (CxGs+NLP, GURT/SyntaxFest 2023)*. Ed. by Claire Bonial and Harish Tayyar Madabushi. Washington, D.C.: Association for Computational Linguistics, pp. 31–35 (cit. on p. 11).

References II


Kleyko, Denis, Dmitri A. Rachkovskij, Evgeny Osipov, and Abbas Rahimi (2022). "A Survey on Hyperdimensional Computing aka. Vector Symbolic Architectures, Part I: Models and Data Transformations." In: *ACM Comput. Surv.* 55.6, pp. 1–40. DOI: 10.1145/3538531 (cit. on pp. 13–16).


Resnik, Philip and Aaron Elkiss (2005). "The Linguist's Search Engine: An Overview." In: *Proceedings of the ACL Interactive Poster and Demonstration Sessions*. Ed. by Masaaki Nagata and Ted Pedersen. Ann Arbor, Michigan: Association for Computational Linguistics, pp. 33–36. DOI: 10.3115/1225753.1225762 (cit. on pp. 6–10).


Schmid, Hans-Jörg (2000). *English Abstract Nouns as Conceptual Shells: From Corpus to Cognition*. De Gruyter. DOI: 10.1515/9783110808704 (cit. on pp. 51–54).