

Big Data & Automated Content Analysis

Week 12 – Wednesday: »From word embeddings to deep learning«

Damian Trilling

d.c.trilling@uva.nl

@damian0604

www.damiantrilling.net

28 April 2021

Afdeling Communicatiewetenschap

Universiteit van Amsterdam

Today

From word counts to word vectors

- Training word embeddings

- Using word embeddings to improve models

- (Ab-)using word embeddings to detect biases

- AEM: An application from our own research

Downstream tasks

- Document comparison

- Supervised Machine Learning

Word vectors

Our BOW approach until now

Representing a document by word frequency counts

Result of preprocessing and vectorizing:

0. He took the dog for a walk to the dog playground

⇒ took dog walk dog playground

⇒ 'took':1, 'dog': 2, walk: 1, playground: 1

Consider these other sentences

Our BOW approach until now

Representing a document by word frequency counts

Result of preprocessing and vectorizing:

0. He took the dog for a walk to the dog playground

⇒ took dog walk dog playground

⇒ 'took':1, 'dog': 2, walk: 1, playground: 1

Consider these other sentences

1. He took the doberman for a walk to the dog playground
2. He took the cat for a walk to the dog playground
3. He killed the dog on his walk to the dog playground

Our BOW approach until now

Representing a document by word frequency counts

Result of preprocessing and vectorizing:

0. He took the dog for a walk to the dog playground

⇒ took dog walk dog playground

⇒ 'took':1, 'dog': 2, walk: 1, playground: 1

Consider these other sentences

1. He took the doberman for a walk to the dog playground
2. He took the cat for a walk to the dog playground
3. He killed the dog on his walk to the dog playground

Our BOW approach until now

Representing a document by word frequency counts

Result of preprocessing and vectorizing:

0. He took the dog for a walk to the dog playground

⇒ took dog walk dog playground

⇒ 'took':1, 'dog': 2, walk: 1, playground: 1

Consider these other sentences

1. He took the doberman for a walk to the dog playground
2. He took the cat for a walk to the dog playground
3. He killed the dog on his walk to the dog playground

Our BOW approach until now

Representing a document by word frequency counts

Result of preprocessing and vectorizing:

0. He took the dog for a walk to the dog playground

⇒ took dog walk dog playground

⇒ 'took':1, 'dog': 2, walk: 1, playground: 1

Consider these other sentences

1. He took the doberman for a walk to the dog playground
2. He took the cat for a walk to the dog playground
3. He killed the dog on his walk to the dog playground

The vectorized representations of these sentences have a “distance” (dissimilarity) of 1 each, but arguably, sentences 0 and 1 should be “closer” than others

- Our vectorizers gave a random ID to each word
- What if we instead would represent each word by another vector representing its meaning?
- For, instance, 'doberman' and 'dog' should be represented by vectors that are close to each other in space, while 'kill' and 'walk' should be far from each other.

⇒ That's the idea behind word embeddings!

Or, more broadly: Can computers understand meanings, semantic relationships, different types of contexts?

Word vectors

Training word embeddings

There are two popular approaches to training word embeddings: GloVe and word2vec.

- GloVe is count-based: dimensionality reduction on the co-occurrence counts matrix.
- Word2Vec is a predictive model: neural network to predict words/contexts
- That means that GloVe takes global context into account, word2vec local context
- Some technical implications for how training can be implemented
- **However, only subtle differences in final result.**

Word2Vec: Continuous Bag of Words (CBOW) vs skipgram

Example sentence: "the quick brown fox jumped over the lazy dog"

CBOW: Predict a word given its context

Dataset:

```
([the, brown], quick), ([quick, fox], brown),  
([brown, jumped], fox), ...
```

skipgram: Predict the context given the word

(quick, the), (quick, brown), (brown, quick), (brown, fox), ...

Example taken from here: <https://medium.com/explore-artificial-intelligence/word2vec-a-baby-step-in-deep-learning-but-a-giant-leap-towards-natural-language-processing-40fe4e86>

Continuous Bag of Words (CBOW) vs skipgram

- CBOW is faster
- skipgram works better for infrequent words
- Both are often used
- Usually, we use larger window sizes (e.g, 5)
- We need to specify the number of dimensions (typically 100–300)

- CBOW is faster
- skipgram works better for infrequent words
- Both are often used
- Usually, we use larger window sizes (e.g, 5)
- We need to specify the number of dimensions (typically 100–300)

*In any event, as a result of the prediction task, we end up with a {100/200/300}-dimensional vector representation of each word.**

* If that makes you think of PCA/SVD, that's not completely crazy, see Levy, O., Goldberg, Y., & Dagan, I. (2018). Improving Distributional Similarity with Lessons Learned from Word Embeddings. *Transactions of the Association for Computational Linguistics*, 3, 211–225. doi:10.1162/tacl_a_00134

“...a word is characterized by the company it keeps...” (Firth, 1957)

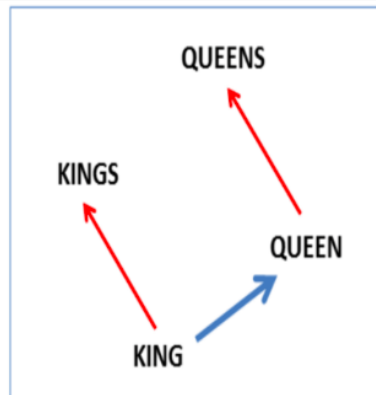
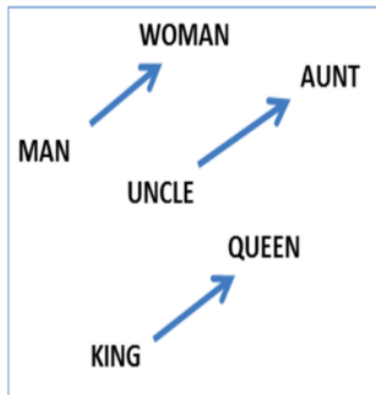
Word embeddings ...

- help capturing the meaning of text
- are low-dimensional vector representations that capture semantic meaning
- are state-of-the-art in NLP...

Firth, J. R. (1957). A synopsis of linguistic theory, 1930-1955. *Studies in linguistic analysis*.

You can literally calculate with words!

And answer questions such as “Man is to woman as king is to
?”



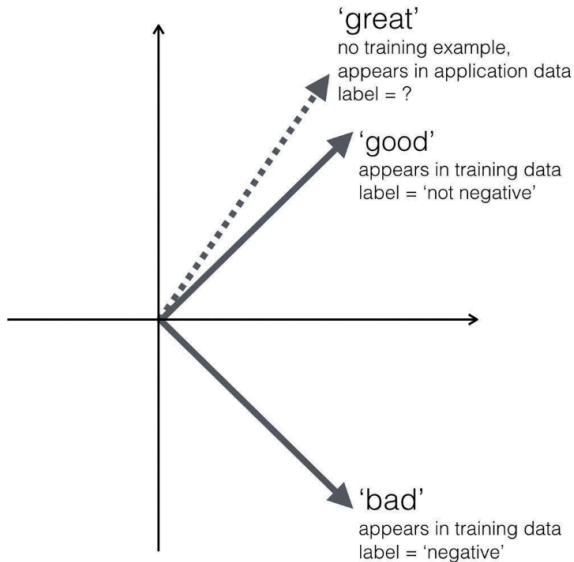
semantic relationships vs. syntactic relationships

Word vectors

Improving models

- Modify CountVectorizer or TfidfVectorizer such that for each term, you do not only count how often it occurs, but also multiply with its embedding vector
- Often, pre-trained embeddings (e.g., trained on the whole wikipedia) are used
- Thus, our supervised model will be able to deal with synonyms and related words!

- Modify CountVectorizer or TfidfVectorizer such that for each term, you do not only count how often it occurs, but also multiply with its embedding vector
- Often, pre-trained embeddings (e.g., trained on the whole wikipedia) are used
- Thus, our supervised model will be able to deal with synonyms and related words!



Rudkowsky, E., Haselmayer, M., Wastian, M., Jenny, M., Emrich, Š., & Sedlmair, M. (2018). More than Bags of Words: Sentiment Analysis with Word Embeddings. *Communication Methods and Measures*, 12(2-3), 140-157. doi:10.1080/19312458.2018.1455817

It's not always black/white...

Sometimes, BOW may be just fine (for very negative sentences, it doesn't matter). But especially in less clear cases ('slightly negative'), embeddings increased performance.

Table 1. Precision, recall, and F1 score for the bag of words approach.

	Actual	Predicted	Precision	Recall	F1 Score
not/slightly negative	524.3	205.6	0.33	0.83	0.47
negative	805.7	1188.7	0.71	0.48	0.57
very negative	730	665.7	0.53	0.58	0.56

Table 2. Precision, recall, and F1 score for the Word Embeddings approach.

	Actual	Predicted	Precision	Recall	F1 Score
not/slightly negative	522.4	575	0.65	0.59	0.61
negative	799.2	771.6	0.52	0.53	0.53
very negative	739.4	714.4	0.55	0.57	0.56

Rudkowsky, E., Haselmayer, M., Wastian, M., Jenny, M., Emrich, Š., & Sedlmair, M. (2018). More than Bags of Words: Sentiment Analysis with Word Embeddings. *Communication Methods and Measures*, 12(2–3), 140–157. doi:10.1080/19312458.2018.1455817

In document similarity calculation

Use cases

- plagiarism detection
- Are press releases/news agency copy/... taken over?
- Event detection

In document similarity calculation

Use cases

- plagiarism detection
- Are press releases/news agency copy/... taken over?
- Event detection

Traditional measures

- Levenshtein distance (How many characters|words do I need to change to transform string A into string B?)
- Cosine similarity (“correlation” between the BOW-representations of string A and string B)

BUT: This only works for literal overlap. What if the writer chooses synonyms?

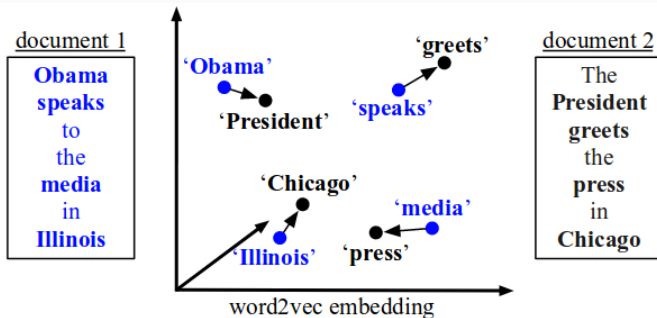


Figure 1. An illustration of the *word mover's distance*. All non-stop words (**bold**) of both documents are embedded into a *word2vec* space. The distance between the two documents is the minimum cumulative distance that all words in document 1 need to travel to exactly match document 2. (Best viewed in color.)

BUT: This only works for literal overlap. What if the writer chooses synonyms?

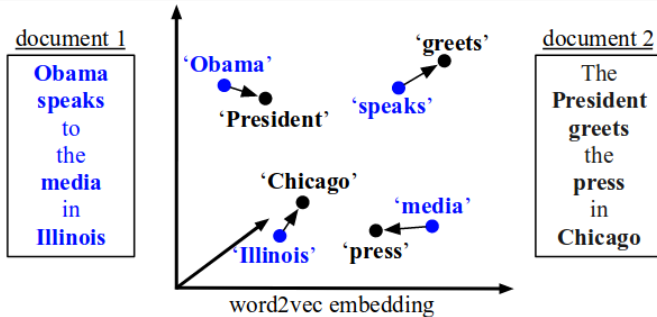


Figure 1. An illustration of the *word mover's distance*. All non-stop words (**bold**) of both documents are embedded into a *word2vec* space. The distance between the two documents is the minimum cumulative distance that all words in document 1 need to travel to exactly match document 2. (Best viewed in color.)

There are several approaches

- word mover's distance
- soft cosine similarity

In common: we use pre-trained embeddings to replace words (that otherwise would just have a random identifier and be unrelated) with vectors representing their meaning, when calculating our measure of interest

Word vectors

Detecting biases

Biased embeddings

- word embeddings are trained on large corpora
- As the task is to learn how to predict a word from its context (CBOW) or vice versa (skip-gram), biased texts produce biased embeddings
- If in the training corpus, the words “man” and “computer programmer” are used in the same context, then we will learn such a gender bias

Bolukbasi, T., Chang, K.-W., Zou, J., Saligrama, V., & Kalai, A. (2016). Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings, 1–25. Retrieved from <http://arxiv.org/abs/1607.06520>

Biased embeddings

Usually, we do not want that (and it has a huge potential for a shitstorm)

unless...

we actually want to chart such biases.

Biased embeddings

Usually, we do not want that (and it has a huge potential for a shitstorm)

unless...

we actually want to chart such biases.

Biased embeddings

Usually, we do not want that (and it has a huge potential for a shitstorm)

unless...

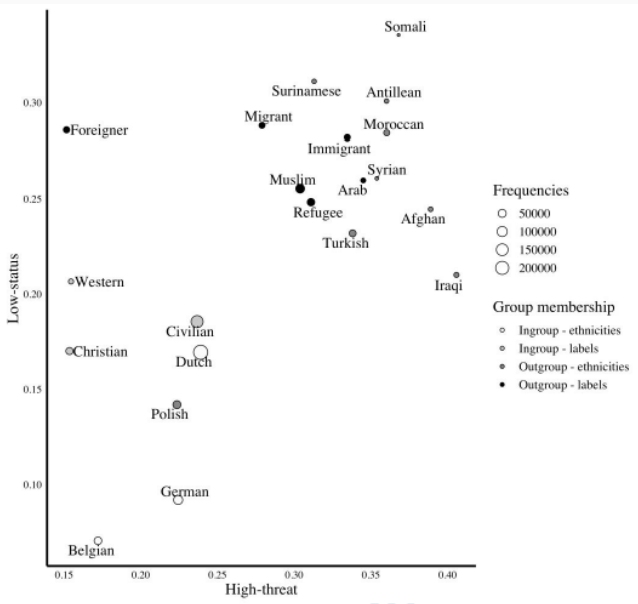
we actually want to chart such biases.

An example from our research

We trained word embeddings on 3.3 million Dutch news articles.

Are vector representations of outgroups (Maroccans, Muslims) closer to representations of negative stereotype words than ingroups?

Kroon, A.C., Van der Meer, G.L.A., Jonkman, J.G.F., & Trilling, D. (in press): Guilty by Association: Using Word Embeddings to Measure Ethnic Stereotypes in News Coverage. *Journalism & Mass Communication Quarterly*



Word vectors

AEM

We can use pre-trained embeddings – but can we make even better ones? **The Amsterdam Embedding Model (AEM)**

Anne Kroon, Antske Fokkens, Damian Trilling, Felicia Loecherbach, Judith Moeller, Mariken A. C. G. van der Velden, Wouter van Atteveldt

Why do this?

- Embedding models are of great interest to communication scholars
- yet... Most publicly available models represent **English** language
- The preparation of good-performing embedding models require a significant amount of **time** and **access to a large amount of data sets**
- Few Dutch embedding models are available, but trained on ordinary human language from the World Wide Web.
- These models do not capture the specifics of news article data and are therefore less suitable to study and understand dynamics of this domain
- ⇒ No model is available trained on Dutch news data

Project's Aim

Aim of the current project

1. Develop and evaluate a high-quality embedding model
2. Assess performance in downstream tasks of interest to Communication Science (such as topic classification of newspaper data).
3. Facilitate distribution and use of the model
4. Offer clear methodological recommendations for researchers interested using our Dutch embedding model

Training data

Training data set

- Dataset: diverse print and online news sources
- Preprocessing: duplicate sentences were removed
- Telegraaf (print & online), NRC Handelsblad (print & online), Volkskrant (print & online), Algemeen Dabldad (print & online), Trouw (print & online), nu.nl , nos.nl
- # words: 1.18b (1181701742)
- # sentences: 77.1M (77151321)

Training model

Training model

- We trained the model using Gensim's Word2Vec package in Python
- Skip-gram with negative sampling, window size of 5, 300-dimensional word vectors

Evaluation

Evaluation of the Amsterdam Embedding Model

Evaluation

Evaluation methods

- To evaluate the model, we compare it to two other publicly available embedding models
 - ⇒ '**Wiki**': Embedding model trained on Wikipedia data (FastText)
 - ⇒ '**Cow**': Embedding model trained on diverse .nl and .be sites (Schafer & Bildhauer, 2012; Tulkens et al., 2016)
 - ⇒ '**AEM**': Amsterdam Embedding Model

Evaluation data

Evaluation data

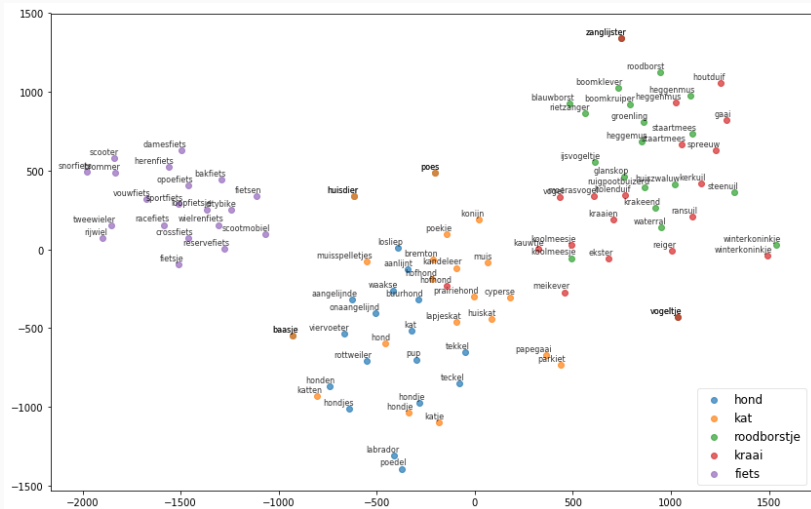
1. 'relationship' / analogy-task (Tulkens et al., 2016)
 - **syntactic relationships**: dans dansen loop [*lopen*]
 - **semantic relationships**: denemarken kopenhagen noorwegen [*oslo*]
2. 5806 relationship tasks

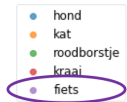


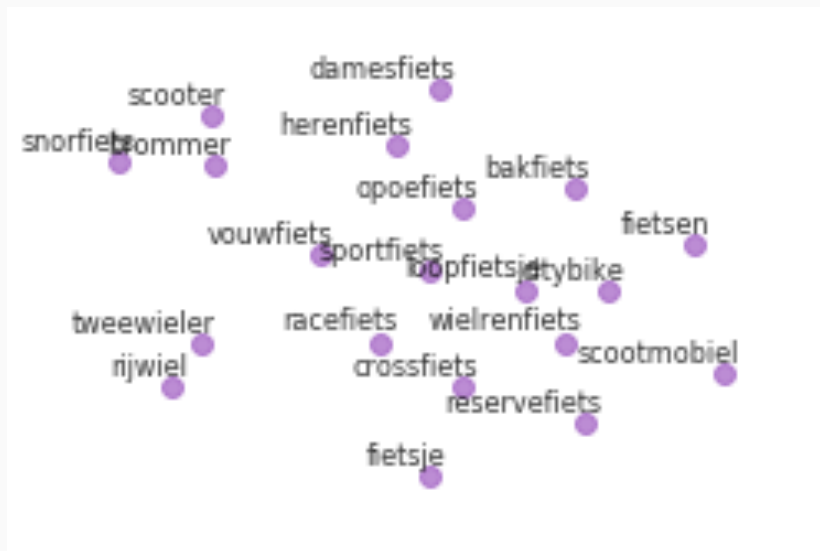
Illustration

Illustration - Using the Amsterdam Embedding Model

Word vectors











Re-usability

Re-usability of the Amsterdam Embedding Model

Re-usability

Reusing model and data

1. See <https://github.com/annekroon/amsterdam-embedding-model>
2. Open access to all the code

Downstream tasks

Downstream tasks

Document comparison

Let's say we have a large corpus of news articles and what to find those that are about the same events.

- 45K articles
- 6 months
- volkskrant.nl, ad.nl, nu.nl

Step 1: Get candidate articles

Comparing everything with everything is

- computationally infeasible
- theoretical nonsensical

Our solution

- Three-day moving window (but “chaining” possible)
- Saturday/Sunday merged into one day

Step 2: Get similarity scores

How to determine similarity between articles?

Our solution

Compare combinations of

- different measures (in particular, $tf \cdot idf$ cosine similarity vs. softcosine similarity)
- different thresholds (to get rid of the overwhelming majority of close-to-zero edges)

Step 3: Network clustering

How to determine events?

After experimenting *a lot*:

Our final solution

- One network for all (instead of one per window)
- Articles are nodes, similarity scores = edge weights
- all edges with weight $<$ threshold removed
- Leiden algorithm (Traag et al., 2019) with Surprise method (Traag et al., 2015) (very suitable for smaller, but more clusters)

Number of articles per event

Table 1: Descriptives for different threshold/similarity combinations

	cosine					soft cosine				
	0.2	0.3	0.4	0.5	0.6	0.2	0.3	0.4	0.5	0.6
mean	2.03	1.58	1.35	1.21	1.12	6.78	2.89	1.88	1.51	1.27
std	3.48	2.00	1.22	0.71	0.45	30.41	10.04	4.27	2.27	1.07
max	88	53	41	21	15	551	367	161	70	30
single-art. events	15626	21854	27135	32232	36348	4262	11043	18305	24337	30700
multi-art. events	6685	6777	6241	5165	3899	2460	4736	5961	5940	5257

What does that mean?

- Use a high threshold!
- Soft-cosine finds some more events, leaves less articles unassigned (good), but that comes at the expense of slightly lower precision
- Example from our data: Because soft-cosine “understands” that Nike and Puma are both sports brands, it incorrectly assigned economic coverage about the two to one event.

How correct are the events?

We manually checked 6×100 events, qualitatively (not shown) and quantitatively:

Similarity	Threshold	Prec. 1 (%)	Prec. 2 (%)	TP/max. TP
cosine	0.4	74	88.52	223/268
cosine	0.5	78	89.02	217/253
cosine	0.6	89	94.39	204/225
softcosine	0.4	56	76.20	234/521
softcosine	0.5	65	81.77	236/379
softcosine	0.6	75	86.92	222/289

Note. Precision 1: The percentage of news events that are entirely clustered correctly. *Precision 2:* The percentage of news articles that are correctly clustered. *max. TP* is the number of articles that are assigned to an event in the sample; hence, the maximum number of true positives that can be achieved.

Cosine vs Softcosine

Also a matter of computational costs

- the document needs to be converted into embeddings
- but once that is done, our document vectors only have 300 instead of thousands of dimensions!

[let's try this out in a notebook (on Friday)]

Downstream tasks

Supervised Machine Learning

In classical SML:

- we represent each document by a vector of word frequencies (or tf·idf scores)
- use these vectors to predict the label

In classical SML:

- we represent each document by a vector of word frequencies (or tf·idf scores)
- use these vectors to predict the label

	ability	about	above	...	zippered	zealotry	→	topic
text1	■	■	■	...	■	■	→	■
text2	■	■	■	...	■	■	→	■
text3	■	■	■	...	■	■	→	■
text4	■	■	■	...	■	■	→	■
new case	■	■	■	...	■	■	→	?

For instance, topic can be ["sports", "economy", "politics"] and the other entries are word frequencies

Our BOW approach until now

Representing a document by word frequency counts

Result of preprocessing and vectorizing:

0. He took the dog for a walk to the dog playground

⇒ took dog walk dog playground

⇒ 'took':1, 'dog': 2, walk: 1, playground: 1

Consider these other sentences

1. He took the doberman for a walk to the dog playground
2. He took the cat for a walk to the dog playground
3. He killed the dog on his walk to the dog playground

The vectorized representations of these sentences have a “distance” (dissimilarity) of 1 each, but arguably, sentences 0 and 1 should be “closer” than others

We modify our vectorizer such that

- for each word in the document, we look up its embedding
- we then aggregate these embeddings (e.g., mean, max, or sum)
- For each document, we now have a 300-dimensional instead of a 10,000-dimensional vector¹

The idea

We modify our vectorizer such that

- for each word in the document, we look up its embedding
- we then aggregate these embeddings (e.g., mean, max, or sum)
- For each document, we now have a 300-dimensional instead of a 10,000-dimensional vector¹

¹in the case of a 300-dimensional embedding model and a vocabulary size of 10,000 of the traditional CountVectorizer)

The idea

We modify our vectorizer such that

- for each word in the document, we look up its embedding
- we then aggregate these embeddings (e.g., mean, max, or sum)
- For each document, we now have a 300-dimensional instead of a 10,000-dimensional vector¹

¹in the case of a 300-dimensional embedding model and a vocabulary size of 10,000 of the traditional CountVectorizer)

What does that mean?

A couple of things:

- Our model is smaller
- We can use words in the prediction dataset *even if it's not in the training dataset*²
- We can learn from similar training samples even if they do not use the same words
- But we also may lose some nuance

²as long as it's in the embedding model, of course

What does that mean?

A couple of things:

- Our model is smaller
- We can use words in the prediction dataset *even if it's not in the training dataset*²
- We can learn from similar training samples even if they do not use the same words
- But we also may lose some nuance

²as long as it's in the embedding model, of course

What does that mean?

A couple of things:

- Our model is smaller
- We can use words in the prediction dataset *even if it's not in the training dataset*²
- We can learn from similar training samples even if they do not use the same words
- But we also may lose some nuance

²as long as it's in the embedding model, of course

What does that mean?

A couple of things:

- Our model is smaller
- We can use words in the prediction dataset *even if it's not in the training dataset*²
- We can learn from similar training samples even if they do not use the same words
- But we also may lose some nuance

²as long as it's in the embedding model, of course

As we see, not *all* embedding models are improving downstream tasks – but good ones can:

<https://github.com/annekroon/amsterdam-embedding-model>

[explain results in README.md]

[Let's do this ourselves in a notebook (on Friday)]

next steps

Try out word embeddings and keras on Friday.

No teaching next week.

Final project

If you haven't done so yet, talk with me about your topic

References



Traag, V. A., Aldecoa, R., & Delvenne, J.-C. (2015). Detecting communities using asymptotical surprise. *Physical Review E*, 92(2). <https://doi.org/10.1103/physreve.92.022816>



Traag, V. A., Waltman, L., & van Eck, N. J. (2019). From Louvain to Leiden: guaranteeing well-connected communities. *Scientific Reports*, 9(1), 1–12. <https://doi.org/10.1038/s41598-019-41695-z>



Trilling, D., & van Hoof, M. (2020). Between article and topic: News events as level of analysis and their computational identification. *Digital Journalism*, 8, 1317–1337. <https://doi.org/10.1080/21670811.2020.1839352>