Statistical Natural Language Processing

Part IV: Representation Learning

Henning Wachsmuth

https://ai.uni-hannover.de

Learning Objectives

Concepts

- Main types of instance representations in NLP
- The distinction between features and feature types
- Distributional vector semantics
- The notions of similarity, distance, and relatedness

Methods

- Semi-automatic learning of feature representations
- Distributional representation learning with skip-gram
- Vector-based similarity measures
- Other similarity measures for specific use cases

Outline of the Course

Introduction

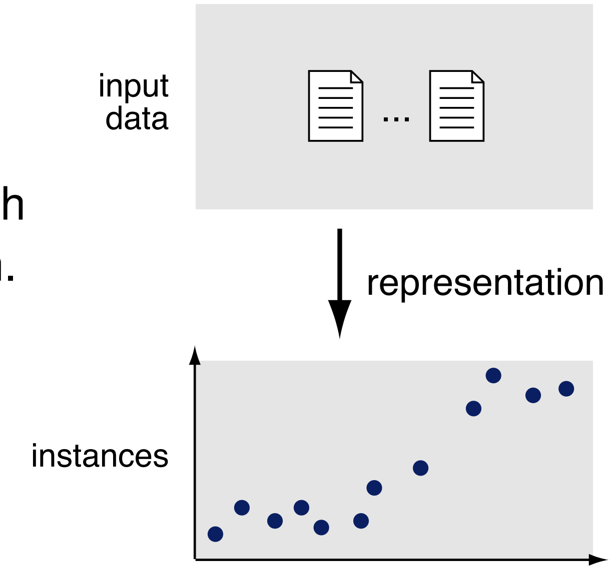
Representation Learning

Representation

- Given a task to predict some type C, each input $o \in O$ is mapped to a common form.

Part-of-speech tagging:	each input is a token
Sentiment analysis:	each input is a text (span)

- Governs what inferences can be drawn



Selected language aspects to capture

- Similar and opposite meaning, such as "car"/"bike" and "hot"/"cold"
- Positive and negative connotations, as in "firm" and "rough"
- Specific types of relatedness, as for "buy", "sell", and "pay"
- Typical structures, such as NP-VP-NP or thesis-antithesis-synthesis

Representation learning

- Learning how to represent inputs (semi-) automatically on input data.
- A good representation enables an effective prediction of C.

Representation Learning

Types of Representations

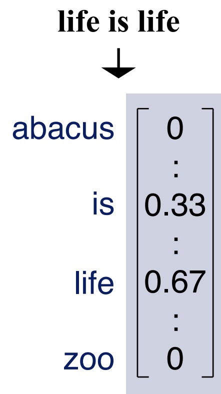
Instance representation in NLP

- Most representations map an input $o \in O$ to a vector of real values.

Feature representations

- Map each o to a (sparse) vector of feature values $\mathbf{x}$.
- Each $x_j \in \mathbf{x}$ captures a measurable property of the input.
- Feature *types* are defined manually, features are learned.
- Features are the basis of feature-based NLP methods.

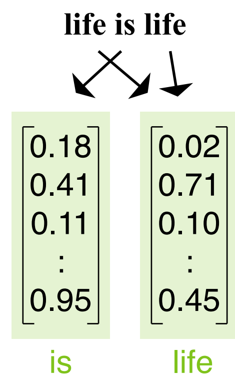
  Neural methods may also include features as part of their input.



Distributional representations

- Map each o to one or more (dense) vectors of values $\mathbf{v}$.
- Each $\mathbf{v}_j$ *embeds* the *distributional semantics* of (parts of) o.
- An embedding model can be learned fully automatically.
- Embeddings are the basis of neural NLP methods.

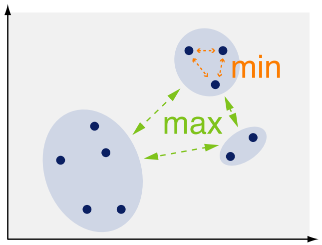
  Embeddings may also be part of a feature vector.

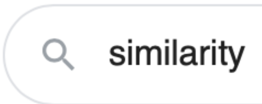


Similarity Measures

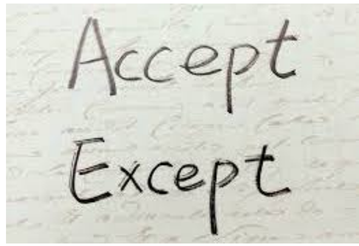
Similarity measure

- A measure that quantifies how similar two instances of a concept are
- Different types of similarity measures exist.
- Many of them build on feature and embedding representations.





"gtp-4" → Did you mean "gpt-4"?



https://piqsels.com

https://commons.wikimedia.org

Selected applications in NLP

- Clustering of related documents
- Retrieval of relevant web pages
- Spelling correction of text
- Near-duplicate or plagiarism detection
- Detection of social bias in text collections

Feature Representation

Feature Representation

Feature

- A feature x denotes any measurable property of an input.
- An input o_i is mapped to one value $x_j^{(i)}$ for each considered feature x_j.
- What features to consider is a design decision.

Example features in NLP

- The relative frequency of a particular word, e.g., "the" $\rightarrow$ [0,1]
- The shape of a word, e.g., Shape $\rightarrow$ {CAPS, CamelCase, ...}
- The existence of an entity type in a sentence, e.g., Organization $\rightarrow$ {0,1}

  ... among zillions of other features

Feature vector

- An ordered set of $m \geq 1$ features of the form $\mathbf{x} = (x_1, \ldots, x_m)$
- Each feature vector $\mathbf{x}^{(i)}$ contains one value $x_j^{(i)}$ for each feature $x_j \in \mathbf{x}$.
- m may vary from a handful to hundreds of thousands.

Feature Representation

Types and Scales

Feature type

- A set of features that conceptually belong together

Bag-of words. Relative frequency of each considered word

POS 3-grams. Relative frequency of each possible part-of-speech 3-gram

- The features of a type can often be found automatically on training data, in order to obtain those that are relevant.

Bag-of words. All words with a training set occurrence $\geq \tau$

Scales of features

- We consider only features here with values from a real-valued scale.
- Nominal, boolean, and similar features can be transformed.

phrase type $\rightarrow$ {“VP”, “NP”, “PP”} $\mapsto$ VP $\rightarrow$ {0,1}, NP $\rightarrow$ {0,1}, PP $\rightarrow$ {0,1}

- Usually, the values of all features are *normalized* to the same interval.

Feature Representation

Normalization

Feature value normalization

- Value ranges of features may vary drastically.
- Normalization scales all values to a uniform range, typically $[0, 1]$ (used here) or $[-1, 1]$.

	# "the"	US?	...
o_1	1	1	
o_2	102	1	
o_3	42	1	
o_4	0	0	
...	...		

Why normalize?

- Machine learning works better with uniform values, due to the interplay with weights, learning rates, and similar.
- At best, the whole (normalized) value range is covered for a feature.

Common ways to normalize

- Divide by the length of the given text (e.g., in # tokens).
- Subtract the mean feature value, divide by the standard deviation.
- Divide by the maximum value found in a training set, and cut at 1.0.
- Divide by a manually-defined maximum and cut at 1.0.

Feature Representation

Learning and Computation of Features

How to learn the set of features in a vector

1. Specify using expert knowledge which feature types to consider.

   (a) Bag-of-words   (b) text length in # tokens

2. Where needed, process training set to get counts of candidate features.

   (a) "the" $\rightarrow$ 4242, "a" $\rightarrow$ 2424, ..., "wooooodchuck" $\rightarrow$ 1   (b) max tokens = 120

3. Keep only features whose counts lie within some defined threshold.

   (a) "the", "a", ..., ~~"wooooodchuck"~~   (b) n/a

How to compute the values for each feature

1. Compute value of each feature in a vector for a given input text.

   (a) "the" $\rightarrow$ 6, "a" $\rightarrow$ 7, ...   (b) # tokens $\rightarrow$ 50

2. Normalize feature values.

   (a) "the" $\rightarrow$ 0.12, "a" $\rightarrow$ 0.14, ...   (b) text length $\rightarrow$ 0.417

Feature Representation

Feature Engineering

Importance of feature engineering

- The features used determine the level of abstraction of o in $\mathbf{x}$.
- Some features generalize worse than others towards unseen data.
- Engineering features that predict a target variable C and generalize well is key to effective feature-based NLP.

Feature engineering in NLP

- Standard features. Some types help in many tasks, e.g., bag-of-words.
- Specific features. Often, the most discriminative types encode expert knowledge about the task and input.

  Also, advanced versions of standard features exist, such as *TF-IDF*.

Feature selection and dimensionality reduction

- Techniques that aim to reduce the set of considered features to improve generalizability and training efficiency

  Not in the focus of this course

Distributional Representation

Distributional Representation

Limitation of feature representations

- While a feature can encode any measurable property, it represents the property's meaning by a single value.
- However, meaning can be expressed in various linguistic ways.

"traveling" vs. "travelling" | "woodchuck" vs. "groundhog" | "Biden" vs. "The President"

"We should ban the death penalty." vs. "Abolish capital punishment."

"You shall know a word by the company it keeps!" (Firth, 1957)

The distributional idea

- Two words are similar, if they occur in similar contexts, i.e., if they have similar words around them.

"Many people like tesgüino."	"A bottle of tesgüino is on the table."
"Tesgüino makes you drunk."	"Tesgüino is brewed from cereal grains."

→ An alcoholic beverage like beer

Distributional Representation

Word Embeddings

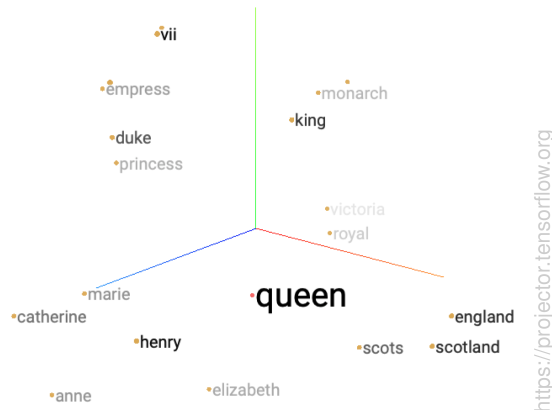
Distributional (vector) semantics

- Represent the meaning of all known words W in an *embedding space* V, where contextually related words are similar.
- Usually, context is modeled by the surrounding words in texts.

Word embedding (aka word vector)

- A real-valued vector $\mathbf{v} \in V$ that represents a specific word $w \in W$ in the modeled space

"queen" $\rightarrow$ $\mathbf{v}_{queen} = (0.13, -0.02, 0.1, 0.4, \ldots, -0.22)$



Properties of word embeddings

- Word embeddings are dense, so the vectors usually have few zeros.
- The m dimensions of embeddings do not have a clear interpretation.
- Mostly, m lies in 100–500 (*static models*) or 768–3072 (*contextualized*).

  All these numbers are small compared to common feature representations.

Distributional Representation

Embedding Models

Word embedding model

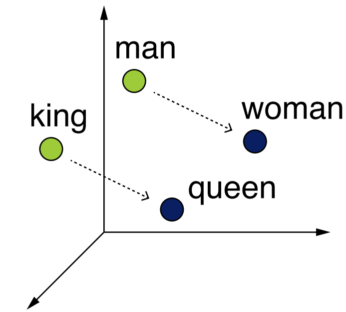
- A function α that maps each word $w \in W$ to an embedding $\mathbf{v} \in V$
- Static. Each w is mapped to a fixed $\mathbf{v}$ ($\alpha_{stat} : W \rightarrow V$).
- Contextualized. The mapping of w also depends on the context $\mathbf{W}$ in which the current instance of w occurs ($\alpha_{ctxt} : W \times \mathbf{W} \rightarrow V$).

  Contextualized embeddings will be discussed in Lecture Part IX.

Some properties of embedding models

projector.tensorflow.org, turbomaze.github.io/word2vecjson

- Similar context results in similar embeddings.
- Analogies are arithmetically represented.



$$\mathbf{v}_{king} - \mathbf{v}_{man} + \mathbf{v}_{woman} \approx \mathbf{v}_{queen}$$

Impact of distributional representations

- Every modern NLP method represents word meaning with embeddings.
- The key is that meaning can be modeled beyond the words used.

  Different strategies exist to deal with unknown words.

Distributional Representation
How to Obtain Embedding Models

Learning of distributed representations

- An embedding model $\alpha : W \rightarrow V$ can be learned from the distribution of words in a (normally huge) text corpus.
- The training process can be realized fully *self-supervised*.

Self-supervised learning

- Supervised learning but without any need for human annotation
- The correct output for an input is available by default or is computable

Learning algorithms for (static) embedding models

- Skip-gram. Learn to predict surrounding words
- CBOW. Learn to predict missing words

______ ______ queen ______ ______	The ______ rules the monarchy

- Both algorithms learn weights that eventually become the embeddings.

Representation Learning with Skip-Gram

Skip-gram (used by *Word2Vec*)

- A self-supervised algorithm that learns an embedding model for any target word w from a vocabulary W
- The embeddings are derived from a binary classification problem.

Skip-gram in a nutshell

1. Treat w with neighboring context words c^+ as positive examples (w, c^+).
2. Sample other words c^- in W to get negative samples (w, c^-).
3. Train a classifier to get the probability $P(+|w, c)$ that a word c is positive.
4. Use the learned classifier weights as the embeddings.

Example

- Let the context of $w =$ "night" be given by a window C of ± 2 words:

...	we	spent	one	**night**	at	that	hotel	...
		c^+_{-2}	c^+_{-1}	w	c^+_1	c^+_2		

- Then ("night", "at") is a positive instance; ("night", "sun") a negative one.

Representation Learning with Skip-Gram

Probabilities

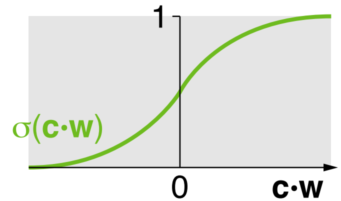
Intuition of skip-gram probabilities

- A word c is likely to occur near w, if its embedding $\mathbf{c}$ is similar to $\mathbf{w}$ of w.
- In this case, skip-gram assigns (w, c) a high probability $P(+|w, c)$.
- Technically, it assign probabilities $P(+|w, C)$ to whole contexts C.

Probability of one instance

- Two vectors $\mathbf{c}, \mathbf{w}$ are more similar, the higher their dot product $\mathbf{c} \cdot \mathbf{w}$ is.
- The sigmoid function maps $\mathbf{c} \cdot \mathbf{w}$ to a probability:

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



Probability of a context

- Skip-gram assumes that all context words in C are independent, so:

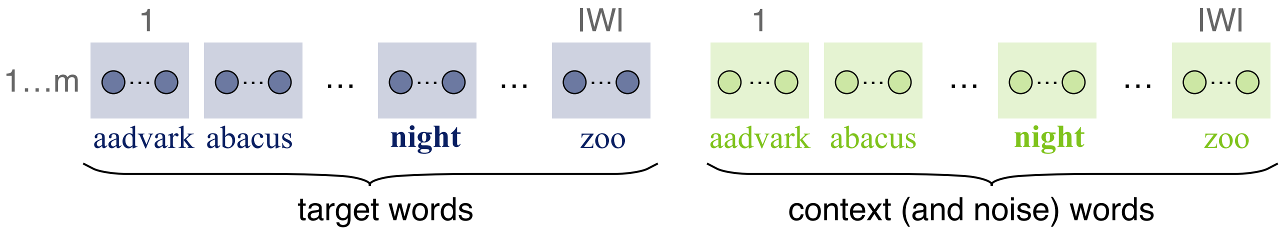
$$P(+|w, C) \ = \prod_{c_i \in C} \sigma(\mathbf{c}_i \cdot \mathbf{w})$$

Representation Learning with Skip-Gram

Embeddings

Learning embeddings with skip-gram

- Two embeddings are learned: $\mathbf{w}$ for w as a target, and $\mathbf{c}_w$ if w is context.
- In total, skip-gram learns $2 \cdot |W|$ weight vectors of length m.



Learning algorithm in a nutshell

- Input. A text corpus D, and a vocabulary W
- Assign random vectors $\mathbf{w}$ and $\mathbf{c}_w$ to each word $w \in W$.
- Iteratively adjust all $\mathbf{w}$ and $\mathbf{c}_w$, such that
  - more similar to embeddings of words that occur nearby
  - less similar to embeddings of words that do not occur nearby
- Output. A model that returns an embedding $\mathbf{w}$ for any word $w \in W$

Representation Learning with Skip-Gram

Classifier

Need for a classifier

- To compute probabilities, embeddings are needed for all words $w \in W$.
- These embeddings are given by the learned weights of the classifier.
- For training, both positive and negative instances are needed.

  The classifier itself is not needed anymore after training.

...	we	spent	one	**night**	at	that	hotel	...
		c^{+}_{-2}	c^{+}_{-1}	w	c^{+}_{1}	c^{+}_{2}		

Training instances

- Positive instances (w, c^{+}) are all tuples derived from the context:

  (night, spent) (night, one) (night, at) (night, that)

- Per (w, c^{+}), k instances (w, c^{-}) are built with *noise words* (e.g., $k = 2$).

  A noise word is a random word $c^{-} \in W$, $c^{-} \neq w$, sampled according to frequency.

  (night, zoo) (night, where) (night, sun) (night, abacus) ... (night, if)

Representation Learning with Skip-Gram

Training

Learning goal

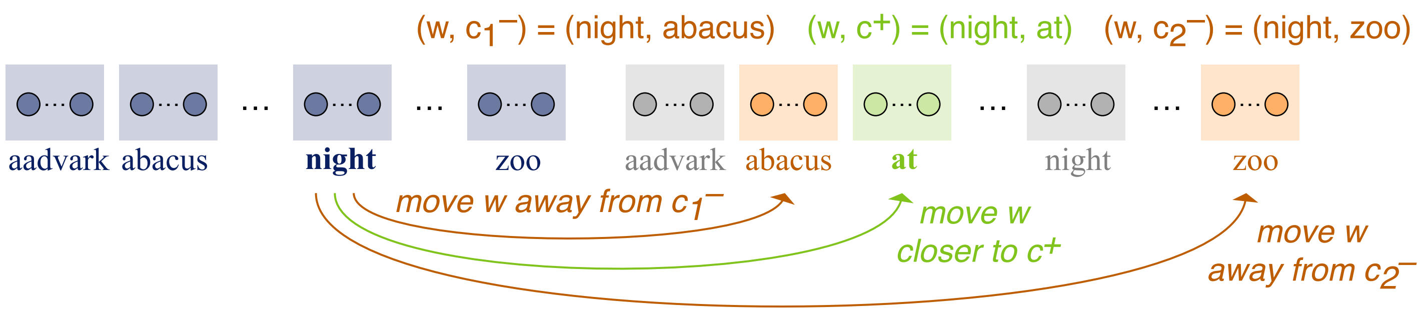
- Given the set of all training instances (w, c^+) and (w, c^-).
- Obtain embeddings that maximize the similarity of w and c^+, and minimize the similarity of w and c^-.

Loss function

- Let $(w, c_1^-), \ldots, (w, c_k^-)$ be the negative instances associated to (w, c^+). Then:

$$\mathcal{L} \;:=\; -\log\left(P(+|w, c^+) \cdot \prod_{i=1}^{k} P(-|w, c_i^-)\right)$$

- This function can be minimized with (stochastic) gradient descent.



Distributional Representation

How to Employ Embedding Models

Selected other learning algorithms

- GloVe. Includes global cooccurrence statistics in the training
- Fasttext. Models subwords to handle unknown words and morphology
- Flair. Learns contextualized character-level embeddings
- BERT. Learns contextualized subword embeddings

Pre-training and fine-tuning

- Training an embedding model is computationally expensive.
- To avoid training models from scratch, *pretrained* models are used.
- Pretrained models can then be *fine-tuned* to a given task.

  Details on such *transfer learning* follow later for neural techniques.

From word embeddings to text embeddings

- Simple. Average the embeddings of each word in a text.
- More sophisticated. Learn embeddings for sentences or similar.

  In general, the longer the text, the harder it is to embed its semantics.

Similarity Measures

Similarity

Similarity

- Two concepts are similar if they overlap somehow in meaning (or form).
- In NLP, concepts may be words, terms, or other text spans.

"car" vs. "bike"

"Biden gave a speech." vs. "The US president spoke."

Synonymy

- Synonyms are substitutable without changing the truth of a proposition

"couch" vs. "sofa" | "big" vs. "large" | "water" vs. "H_20" | "vomit" vs. "throw up"

- Synonymy is a relation between senses rather than words.

"big" vs. "large" $\rightarrow$ "Linda thought, good that I have such a <insert> brother."

Principle of contrast (Clark, 1987)

- Differences in linguistic form always imply some difference in meaning.
- That is, there are hardly any perfect synonyms.

Even seemingly identical terms usually differ in terms of politeness, slang, genre, etc.

Similarity

Related Notions

Similarity vs. relatedness (aka association)

- Words and terms can also be related in other ways than similarity.
- For example, they may be from the same semantic domain.

In certain settings, relatedness adequately reflects similarity.

"car" vs. "gas" | "Joe Biden" vs. "president"

Similarity vs. distance

- Similarity can be seen as the inverse of distance.
- With normalized values, deriving one from the other is straightforward.

Meaning vs. form

- The form of texts may serve as a proxy for similarity, but this has limits.

Similar form, but different meaning: "This is sh*t." vs. "This is *the* sh*t."

Vice versa: "Biden visited the capital of France." vs. "Joe Biden was in Paris."

Similarity Measures

Similarity measure

- A real-valued function sim that quantifies how similar two objects o_1, o_2 of the same concept O are
- Mostly, values of sim range between 0 (no similarity) and 1 (identity).
- In NLP, objects are text spans represented in some way.

Types of similarity measures in NLP

- Vector-based. For feature and embedding vector representations
- String-based. For character sequences
- Concept-based. For taxonomic relatedness of concepts
- Set-based. For sets of words or embeddings

Similarity as a hyperparameter

- There is not one best measure for all tasks.
- One way to deal with this is to evaluate different measures.
- In some tasks, multiple measures can also be used simultaneously.

Vector-based Measures

Vector-based similarity measures

- Quantify similarity of $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}$ based on values $x_j^{(1)}, x_j^{(2)}$ at each position j
- Apply equally to feature vectors and embedding vectors
- Examples. Cosine, Euclidean, and Manhattan similarity

  Various similarity and distance measures exist for vectors (Cha, 2007).

Measuring similarity between vectors

- Compare two vectors of the same representation with each other.

  $\mathbf{x}^{(1)} = (1.0, 0.1, 0.3),\ \mathbf{x}^{(2)} = (0.0, 0.1, 0.6)$ for $\mathbf{x} =$ (red, green, blue)

- Compute similarity individually for values $x_j^{(1)}$ and $x_j^{(2)}$ at each position j.

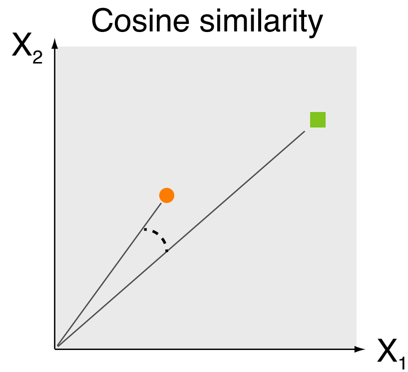
  $sim_1(1.0, 0.0) = 0.0 \quad sim_2(0.1, 0.1) = 1.0 \quad sim_3(0.3, 0.6) = 0.5$

- Aggregate all individual similarities in some way.

  $sim(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) = \frac{0.0+1.0+0.5}{3} \approx 0.5$

Vector-based Measures

Cosine Similarity



Cosine similarity (aka cosine score)

- The cosine of the angle between two vectors
- The smaller the angle, the more similar the vectors.
  Cosine is maximal (1.0) for 0°.

$$sim_{Cosine}(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) \; := \; \frac{\mathbf{x}^{(1)} \cdot \mathbf{x}^{(2)}}{||\mathbf{x}^{(1)}|| \cdot ||\mathbf{x}^{(2)}||} = \frac{\sum_{j=1}^{m} x_j^{(1)} \cdot x_j^{(2)}}{\sqrt{\sum_{j=1}^{m} x_j^{(1)^2}} \cdot \sqrt{\sum_{j=1}^{m} x_j^{(2)^2}}}$$

Observations

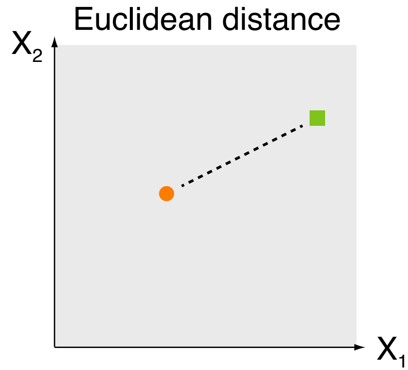
- Cosine focuses on the vector values that occur (i.e., those with $x_j \neq 0$).
- It abstracts from the length of the vectors.
- It targets settings where a vector's direction matters mainly.
  A typical task is matching queries with documents in web search.

Notice

- Angle computation works for any number of dimensions.

Vector-based Measures

Euclidean Similarity



Euclidean distance

- The straight-line distance between two vectors

$$d_{Euclidean}(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) \;:=\; \sqrt{\sum_{j=1}^{m} (x_j^{(1)} - x_j^{(2)})^2}$$

Euclidean similarity

- If all values are normalized to $[0,1]$, the Euclidean similarity is:

$$sim_{Euclidean}(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) \;:=\; 1 - \frac{d_{Euclidean}(\mathbf{x}^{(1)}, \mathbf{x}^{(2)})}{\sqrt{m}}$$

Observations

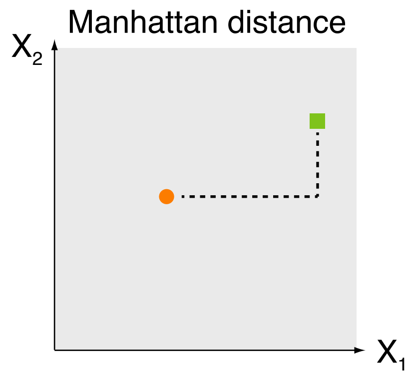
- In Euclidean spaces, a 0 does not mean the absence of a property.
- Euclidean similarity target settings where exact vector values matter.

Notice

- Euclidean spaces generalize to any number of vector dimensions $m \geq 1$.

Vector-based Measures

Manhattan Similarity



Manhattan distance (aka Hemming or city block distance)

- The sum of all differences between two vectors

$$d_{Manhattan}(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) \; := \; \sum_{j=1}^{m} |x_j^{(1)} - x_j^{(2)}|$$

Manhattan similarity

- If all values are normalized to $[0, 1]$, the Manhattan similarity is:

$$sim_{Manhattan}(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) \; := \; 1 - \frac{d_{Manhattan}(\mathbf{x}^{(1)}, \mathbf{x}^{(2)})}{m}$$

Observations

- Manhattan is preferred when outliers in vector positions do not matter.
- Manhattan and Euclidean are special cases of the *Minkowski distance*:

$$d_{Minkowski}(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) := \sqrt[p]{\sum_{j=1}^{m} |x_j^{(1)} - x_j^{(2)}|^p} \quad \text{for any } p \in \mathbb{N}^+$$

Similarity Measures

Other Measures

String-based measures

- Quantify how similar two character sequences are
- Example. *Minimum edit distance* for writing variations

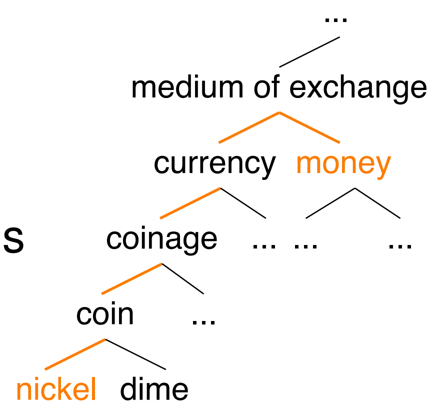
“traveling” vs. “travelling” | “he’s king” vs. “he’s the king”

```
I N T E * N T I O N
| | | | | | | | | |
d s s   i s
| | | | | | | | | |
* E X E C U T I O N
```

Concept-based measures

- Quantify how related two terms are conceptually.
- Example. WordNet for hypernym/hyponym relations

“woodchuck” vs. “groundhog” | “money” vs. “nickel”



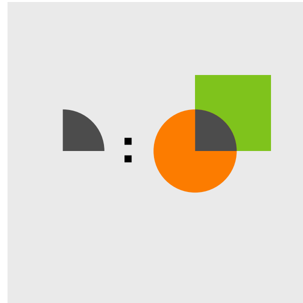
Set-based measures

- Quantify how similar to (ordered) sets of words or terms are
- Examples. *Jaccard similarity* and *word mover’s distance*

“Biden speaks to the media in Illinois” vs. “The press is greeted by the President in Chicago”

Set-based Measures

Jaccard Similarity



Jaccard similarity (aka Jaccard coefficient/index)

- The proportion of the intersection of two (possibly ordered) sets from their union

$$sim_{Jaccard}(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) := \frac{|\mathbf{x}^{(1)} \cap \mathbf{x}^{(2)}|}{|\mathbf{x}^{(1)} \cup \mathbf{x}^{(2)}|} = \frac{|\mathbf{x}^{(1)} \cap \mathbf{x}^{(2)}|}{|\mathbf{x}^{(1)}| + |\mathbf{x}^{(2)}| - |\mathbf{x}^{(1)} \cap \mathbf{x}^{(2)}|}$$

$$= \frac{\sum_{x_j^{(1)} = x_j^{(2)}} 1}{\sum_{x_j^{(1)}} 1 + \sum_{x_j^{(2)}} 1 - \sum_{x_j^{(1)} = x_j^{(2)}} 1}$$

Observations

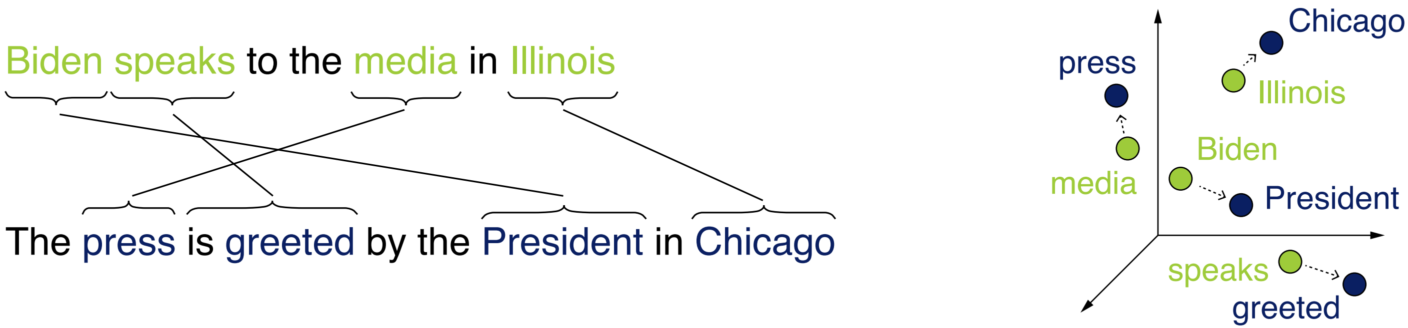
- Jaccard does *not* consider the size of the difference between values.
- This abstraction may benefit robustness (i.e., it “overfits” less).

Notice

- The input may be sets of words in two texts or vectors of boolean values

Word Mover's Distance (Kusner et al., 2015)

- The distance of the optimal alignment of two texts, each represented as a sequence of word embeddings
- The optimal alignment is a matching of words that maximizes the average similarity of the respective embeddings.



Observations

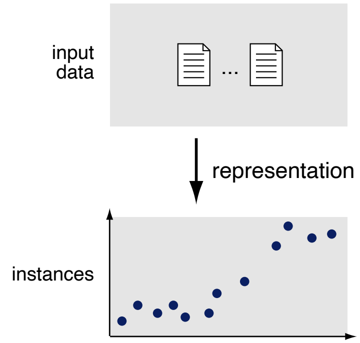
- The ordering of words plays no role in the word mover's distance.
- More sophisticated extensions may be considered for this purpose.

Conclusion

Conclusion

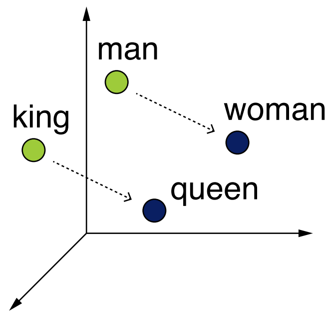
Representation learning

- NLP uses common representations of words and texts
- These capture intrinsic or distributional text properties
- They can be learned on (unannotated) text corpora



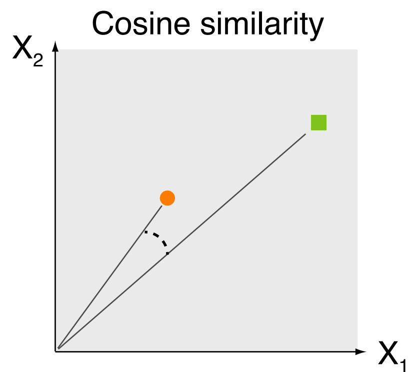
Types of representations

- Mostly, inputs are represented as real-valued vectors
- Features can be learned, their types are hand-defined
- Embeddings can be learned fully self-supervised



Similarity measures

- Quantify how similar concepts are in meaning or form
- Most measures build on feature or embedding vectors
- Different types of measures with different use cases



References

Some content and examples taken from

- **Cha (2007).** Sung-Hyuk Cha. Comprehensive Survey on Distance/Similarity Measures between Probability Density Functions. International Journal of Mathematical Models and Methods in Applied Sciences, 1(4):300–307, 2007.
- **Clark (1987).** Eve V. Clark. The principle of contrast: A constraint on language acquisition. Mechanisms of language acquisition, pages 1–33. LEA, 1987.
- **Firth (1957)**. John R. Firth. Applications of General Linguistics. Transactions of the Philological Society 56(1), pages 1–14, 1957.
- **Jurafsky and Manning (2016).** Daniel Jurafsky and Christopher D. Manning. Natural Language Processing. Lecture slides from the Stanford Coursera course, 2016. `https://web.stanford.edu/~jurafsky/NLPCourseraSlides.html`
- **Jurafsky and Martin (2021).** Daniel Jurafsky and James H. Martin. Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics. Draft or 3rd edition, December 29, 2021. https://web.stanford.edu/ jurafsky/slp3/
- **Kusner et al. (2015).** Matt J. Kusner, Yu Sun, Nicholas I. Kolkin, and Kilian Q. Weinberger. From Word Embeddings to Document Distances. In Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37, pages 957–966, 2015.