Chapter NLP:II

II. Text Models

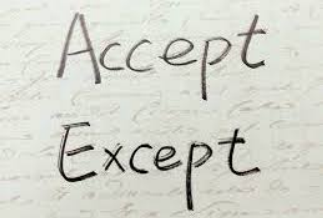
- ❑ Text Preprocessing
- ❑ Text Representation
- ❑ Text Similarity
- ❑ Text Classification
- ❑ Language Modeling
- ❑ Sequence Modeling

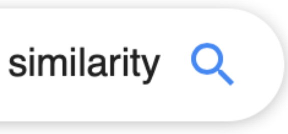
Text Similarity

Text can be similar in different ways:

- Spelling correction
- Retrieval of relevant web pages
- Detection of related documents
- Paraphrase recognition
- (Near-) Duplicate or text reuse detection
- Identification of counterarguments
- Clustering
- Evaluation of machine translation and summarization

. . . and many more





Text Similarity

Text can be similar in different ways:

1. Lexical similarity describes the similarity of form, e.g.:
   - Language variation.

     `color` *vs.* `colour`
   - Additional words.

     `This is shit.` *vs.* `This is` *the* `shit.`
   - Spelling errors.

     `restaurant` *vs.* `westauwang`
   - Similar spelling.

     `content` *vs.* `contempt`

2. Semantic similarity describes the similarity of meaning, e.g.:
   - Synonymy.

     `content` *vs.* `satisfied`
   - Paraphrase.

     `Biden visited the capital of France.`

     vs. `The president was in Paris.`

Text Similarity

Similarity Measures

Depending on the kind of similarity to be measured, we must determine the

- level of text, token, sentence, document, ...
- text representation, strings, count vectors, word vectors, ...
- and similarity function.

Families of similarity functions:

- String-based: Lexical similarity of the form of words.
- Resource-based: Relations between tokens in knowledge graphs.
- Vector distance: Geometric similarity of vector representations in their space.
- Divergence: Similarity between word distributions of documents/corpora.
- Word vectors: Semantic similarity of sentences via word vectors.

Text Similarity

String-based Similarity: Hamming Distance

Idea: Count the number of character positions where two strings s_1 and s_2 differ.

- The Hamming distance measures the number of substitutions required to transform a (bit) sequence into another.
- Can be applied to character sequences of equal length. Pad the shorter token.
- Vulnerable to character addition/removal. i.e. program vs programme

Hamming distance with aligned tokens.

s_1	r	e	s	t	a	u	r	a	n	t	
s_2	w	e	s	t	a	u	w	a	n	g	
Distance	1	0	0	0	0	0	1	0	0	1	= 3

Hamming distance with character removal and padding.

s_1	r	e	s	t	a	u	r	a	n	t	
s_2	r	e	s	t	u	r	a	n	t	[P]	
Distance	0	0	0	0	1	1	1	1	1	1	= 6

Text Similarity

String-based Similarity: Levenshtein Distance

Idea: Count the minimum number of edit operations needed to transform s_1 into s_2.

- Insertion: Add a character to s_1.
- Deletion: Remove a character from s_1.
- Substitution: Replace a character in s_1 with a different one.

Find the minimum cost by trying every combination of operations ($O(n^2)$):

Levenshtein(s_1, s_2)

IF $|s_2| = 0$ THEN *return*$(|s_1|)$ ENDIF // Do $|s_1|$ deletions.
IF $|s_1| = 0$ THEN *return*$(|s_2|)$ ENDIF // Do $|s_2|$ insertions.
IF $s_1[0] = s_2[0]$ THEN *return*(Levenshtein$(s_1[1:], s_2[1:])$) ENDIF
$\mathbf{l}_{del} = 1 +$ Levenshtein$(s_1[1:], s_2)$ // Cost when doing a deletion.
$\mathbf{l}_{ins} = 1 +$ Levenshtein$(s_1, s_2[1:])$ // Cost when doing an insertion.
$\mathbf{l}_{sub} = 1 +$ Levenshtein$(s_1[1:], s_2[1:])$ // Cost when doing a substitution.
return(*min*$(\mathbf{l}_{del}, \mathbf{l}_{ins}, \mathbf{l}_{sub})$)

Text Similarity

String-based Similarity

Applications:

- Spelling correction. Find the most similar word in the vocabulary.
- Query processing. Replace spelling variants, . . .
- Plagiarism detection. Find near-verbatim copies or longest common subsequences.
- De-noising. Correct corrputed texts.
- Evaluate translation or summarization. Compare to a ground truth.

Limitations:

- Semantically agnostic. Use resource-based methods for word-level semantic similarity.
- Very sensitive to word order or inserted text, more suited to short texts or word-by-word comparisons. Use vector distance methods for long sequences.

Remarks:

- There are several variations of Levenshtein distance that modify its behavior, commonly called edit distance.
  - Damerau-Levenshtein distance also allows transposition (swap two adjacent characters at the cost of one operation). This is desirable for spellcheckers.
  - Longest Common Subsequence allows only insertion and deletion, which is faster and can better utilize hashing.
  - Hamming distance allows only substitution.
  - Jaro distance allows only transposition.
- It is common to assign a cost to each operation and instead of counting the number of operations.
- Edit distance can be computed in $O(n+m)$ with the Wagner-Fischer dynamic programming algorithm.

Text Similarity

Resource-based Similarity: Thesaurus relations [NLP:VI-2 ff.]

Idea: Calculate the shortest path through the graph of the semantic word relations found in a thesaurus like WordNet.

Common thesaurus relations used:

- Synonymy: Two words that (in some context) have the same meaning.

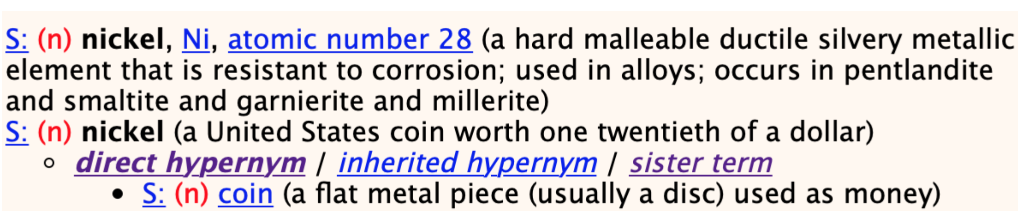
  `couch` $\longleftrightarrow$ `sofa`   `big` $\longleftrightarrow$ `large`

- Hypernymy: A word is included in the meaning of another. IS-A relation.

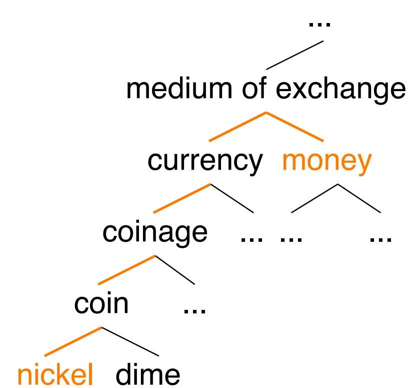
  `vehicle` $\overrightarrow{\text{Hypernym}}$ `car`

  `vehicle` $\overrightarrow{\text{Hypernym}}$ `ship`

WordNet entry for `nickel`:



S: (n) **nickel**, Ni, atomic number 28 (a hard malleable ductile silvery metallic element that is resistant to corrosion; used in alloys; occurs in pentlandite and smaltite and garnierite and millerite)
S: (n) **nickel** (a United States coin worth one twentieth of a dollar)
- *direct hypernym* / *inherited hypernym* / *sister term*
  - S: (n) coin (a flat metal piece (usually a disc) used as money)



Text Similarity

Resource-based Similarity

Applications:

- Synonym search and lexical substitution. (cf. [NLP:VI-2 ff.])

Limitations:

- Thesauri do not cover phrases or sentences. Use word vector-based methods for semantic similarity of longer sequences.
- Thesauri cover only some word classes (i.e. nouns, verbs, adjectives, adverbs) and only some of the words in them.
- Verbs and adjectives are not as hierarchically structured as nouns.
- Thesauri are not available for all languages.

Text Similarity

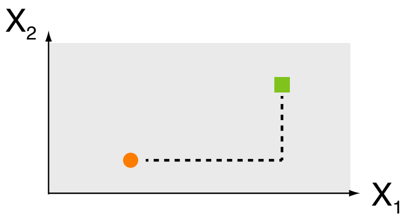
Vector Distance

Idea: If the text are in a numeric vector representation with ordinal dimensions (BoW or feature vectors), then a geometric similarity measure can be used.

Distance *d* of two texts $\mathbf{p}$ and $\mathbf{q}$ represented as m-length (BoW) vectors.

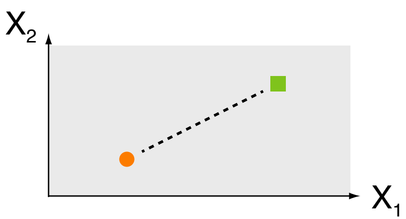
- Manhattan distance The Manhattan distance is the sum of all absolute differences between two feature vectors.

$$d_{\text{Manhattan}}(\mathbf{p}, \mathbf{q}) = \sum_{i=1}^{m} |p_i - q_i|$$



- Euclidean distance The Euclidean distance captures the absolute straight-line distance between two feature vectors.

$$d_{\text{Euclid}}(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{m} |p_i - q_i|^2}$$



Note that the indices i correspond to the same token or feature in both documents.

Text Similarity

Vector Distance

Idea: If the text are in a numeric vector representation with ordinal dimensions (BoW or feature vectors), then a geometric similarity measure can be used.

Both distances can be transformed into a similarity metric:

1. Normalize all dimensions to $[0, 1]$
2. Normalize the distance for the vector size m
3. Invert the measure so larger is more similar.

$$\mathit{sim}_{\text{Manhattan}}(\mathbf{p}, \mathbf{q}) = 1 - \frac{d_{\text{Manhattan}}(\mathbf{p}, \mathbf{q})}{m}$$

- Vector distances (based on count vectors) are vulnerable to document length.
- A long document will be distant from a short one even if it just contains the same content multiple times.

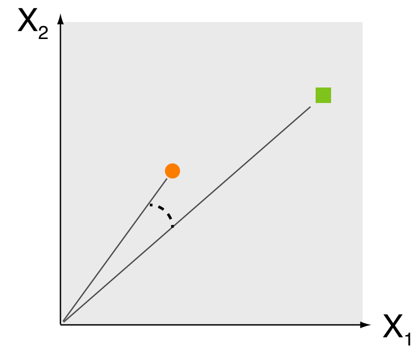
Text Similarity

Vector Similarity: Cosine Similarity

Idea: Measure the difference of the vector's direction and ignore its length.

- Cosine similarity captures the cosine of the angle between two feature vectors.

$$\mathit{sim}_{\text{Cosine}}(\mathbf{p}, \mathbf{q}) = \frac{\mathbf{p} \cdot \mathbf{q}}{||\mathbf{p}|| \cdot ||\mathbf{q}||} = \frac{\sum_{i=1}^{m} p_i \cdot q_i}{\sqrt{\sum_{i=1}^{m} p_i^2} \cdot \sqrt{\sum_{i=1}^{m} q_i^2}}$$



- The smaller the angle, the more similar the vectors. The cosine is maximal for 0°.
- $||\mathbf{x}||$ denotes the L2 norm of vector $\mathbf{x}$.

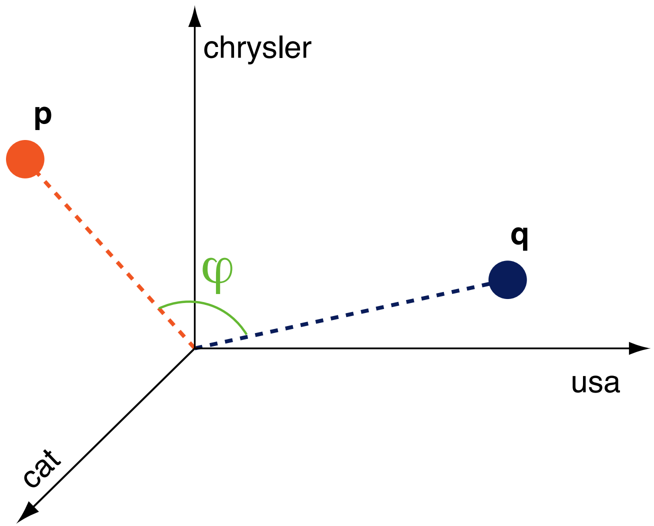
Text Similarity

Vector Similarity: Cosine Similarity

Geometric interpretation of the Cosine:

$$\mathbf{p} = \begin{pmatrix} \texttt{chrysler} & 0.1 \\ \texttt{usa} & 0.4 \\ \texttt{cat} & 0.3 \\ \texttt{dog} & 0.7 \\ \texttt{mouse} & 0.5 \end{pmatrix}$$

$$\mathbf{q} = \begin{pmatrix} \texttt{chrysler} & 0.2 \\ \texttt{usa} & 0.1 \\ \texttt{cat} & 0.5 \\ \texttt{ostrich} & 0.1 \\ \texttt{elephant} & 0.1 \end{pmatrix}$$



The angle φ between $\mathbf{p}$ and $\mathbf{q}$ is about $51°$, $\cos(\varphi) \approx 0.63$.

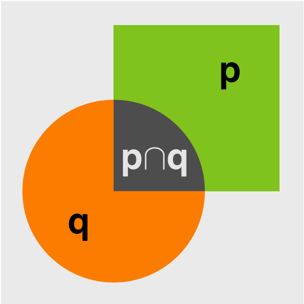
Text Similarity

Vector Similarity: Jaccard Similarity

Idea: If the size of the (element-wise) difference is not relevant, the set overlap can be used to measure similarity.

- Jaccard similarity compares the size of the intersection of two sets to their union.

$$\begin{aligned} sim_{\text{Jaccard}}(\mathbf{p}, \mathbf{q}) &= \frac{|\mathbf{p} \cap \mathbf{q}|}{|\mathbf{p} \cup \mathbf{q}|} \\ &= \frac{|\mathbf{p} \cap \mathbf{q}|}{|\mathbf{p}| + |\mathbf{q}| - |\mathbf{p} \cap \mathbf{q}|} \\ &= \frac{\sum_{p_i = q_i} 1}{m + m - \sum_{p_i = q_i} 1} \end{aligned}$$



- The term $p_i = q_i$ assumes binary features.

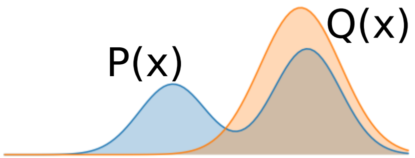
Text Similarity

Vector Similarity: Divergence

Idea: If the texts are represented as probability distributions (over a shared vocabulary), their divergence can be used to measure similarity.

- Kullback–Leibler–Divergence (KLD) measures the distribution divergence with information gain. KLD is asymmetric and not a metric.

$$d_{\mathsf{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log \left(\frac{P(x)}{Q(x)} \right)$$



- Jenson-Shannon-Divergence (JSD) is a symmetric adaptation of the KLD.

$$\begin{aligned} sim_{\mathsf{JSD}}(P(x) \parallel Q(x)) &= 1 - \left(\frac{1}{2} D_{\mathsf{KL}}(P(x) \parallel M(x)) + \frac{1}{2} D_{\mathsf{KL}}(Q(x) \parallel M(x)) \right) \\ M(x) &= \frac{1}{2}(P(x) + Q(x)) \end{aligned}$$

Text Similarity

Vector Similarity

Applications:

- Retrieval models in search. (cf. [IR:VI-92 ff.])
- Clustering documents. (cf. Data Mining)
- Language detection. Similarity between a document and corpus vectors.
- Plagiarism detection. Find candidate documents. Find passages more robustly.
- Authorship analysis. Are two documents written by the same author?

Limitations:

- Agnostic to word order, token semantics, and sentence semantics.
- Poor performance on short texts (sentences); they do not share many words.
- Poor performance on long texts (corpora); they approximate the language.
- Similarity scores and vector differences have no linguistic interpretation.

Remarks:

- Count vectors can be transformed into probability distributions (cf. Probability Mass Functions and [ML:VII-4 ff.])
- JSD and Cosine similarity are (roughly) equivalent for word distributions over a shared vocabulary (i.e. count vectors). However, using a geometric interpretation (cosine) makes little sense for distributions.
- Vector-based methods can be used to compare both document vectors and word vectors.
- If a vector encodes semantic information, then vector-based similarity functions measure semantic similarity.

Similarity Measures

Word Vector Similarity: Sentence Embeddings [Iyyer et al. 2015]

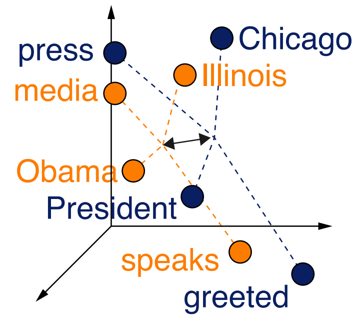
Idea: Compute a sentence embedding based on a sequence of word vectors, compare the embeddings with a geometric measure (i.e. cosine similarity).

- Vector Average Similarity compares the geometric average of the word vectors.

  s^1: `Obama speaks to the media in Illinois`

  s^2: `The press is greeted by the President in Chicago`

$$sim_{\text{cosine}}\left(\sum_{i=1}^{|s^1|} \frac{s_i^1}{|s^1|}, \sum_{i=1}^{|s^2|} \frac{s_i^2}{|s^2|}\right)$$



Similarity Measures

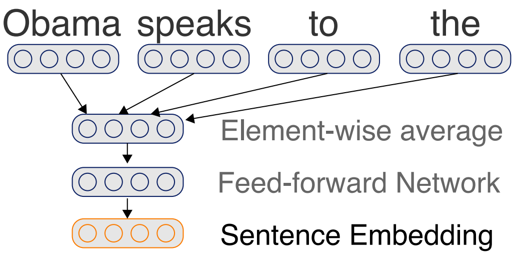
Word Vector Similarity: Sentence Embeddings

[Iyyer et al. 2015, Cer et al. 2018, Reimers et al. 2019]

Idea: Compute a sentence embedding based on a sequence of word vectors, compare the embeddings with a geometric measure (i.e. cosine similarity).

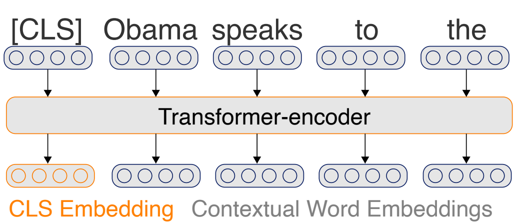
- Deep Average Networks

  Train a feed-forward neural network to predict the sentence embedding given the geometric average of the word vectors as input. Often trained on classification tasks (i.e. sentiment detection).



- Universal Sentence Encoder and Sentence-BERT

  Transformer-encoder can be trained so that their CLS-token embedding equivalates a sentence embedding, which can then be compared through i.e cosine similarity.



Similarity Measures

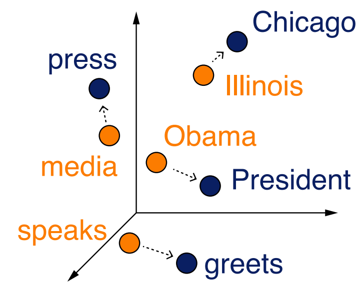
Word Vector Similarity: Word Mover Distance [Kusner et al. 2015]

Idea: For each (non-stop) word $i \in$ s^1, find the closest word $j \in$ s^2 in the word vector space. Sum their distances.

Simple case. There is an even number of (non-stop) word in both sentences. Find the best pairing.

s^1: `The President greets the press in Chicago`

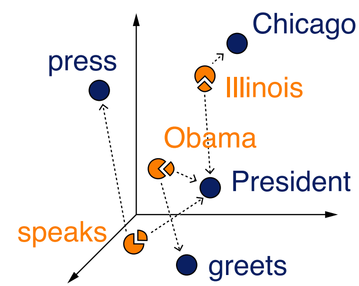
s^2: `Obama speaks to the media in Illinois`



Difficult case. There is an uneven number of word in both sentences. We assume that s^1 should be evenly distributed over s^2, so words must be split and combined.

s^1: `The President greets the press in Chicago`

s^2: `Obama speaks in Illinois`



The Word Mover Distance (WMD) optimally (with minimal transportation cost) distributes the words of the source to fill the capacity of the sink.

Similarity Measures

Word Vector Similarity: Word Mover Distance [Kusner et al. 2015]

The Word Mover Distance finds the minimum cumulative transportation cost to move all words from $i \in \mathsf{s}^1$ to words in $j \in \mathsf{s}^2$ in an embedding space.

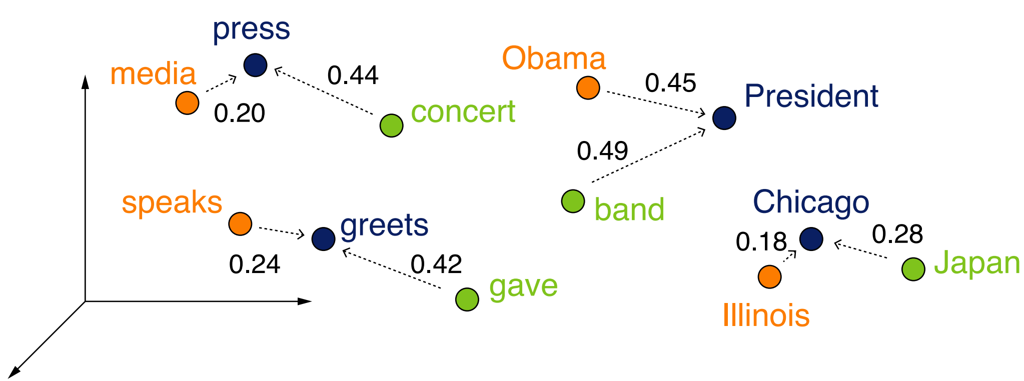
s^1: `The President greets the press in Chicago`

s^2: `Obama speaks to the media in Illinois`

s^3: `The band gave a concert in Japan`

$$\mathrm{WMD}(\mathsf{s}^1, \mathsf{s}^2) = 0.45 + 0.24 + 0.20 + 0.18 = 1.07$$

$$\mathrm{WMD}(\mathsf{s}^1, \mathsf{s}^3) = 0.49 + 0.42 + 0.44 + 0.28 = 1.63$$



Similarity Measures

Word Vector Similarity: Word Mover Distance [Kusner et al. 2015]

The Word Mover Distance finds the minimum cumulative transportation cost to move all words from $i \in \mathsf{s}^1$ to words in $j \in \mathsf{s}^2$ in an embedding space.

$$\text{WMD} = \min_{\mathbf{T} \geq 0} \sum_{i,j=1}^{n} \mathbf{T}_{i,j} \cdot c(i,j)$$

- The cost $c(i,j) = \|\mathbf{x}_i - \mathbf{x}_j\|_2$ is the euclidean distance.
- The flow matrix $\mathbf{T}_{i,j}$ indicates which capacity of word i is moved to word j
- The outgoing flow of i must expend the capacity d_i^1: $\sum_{j=1}^{n} \mathbf{T}_{i,j} = d_i^1$

  The incoming flow into j must fill the capacity d_j^2: $\sum_{i=1}^{n} \mathbf{T}_{i,j} = d_j^2$
- The capacity is the term weight, i.e. $d_i^1 = \dfrac{1}{\|\mathsf{s}_1\|}$

This problem definition is equivalent to the Earth Mover Distance, a common problem from transportation with has efficient solvers. [Pele and Werman, 2009]

Text Similarity

Word Vector Similarity

Applications:

- Every time semantics is more important than form and sequences are (relatively) short.

Limitations:

- Often slower than count-vector based methods if sentence embeddings can not be pre-computed (ie. at index time).
- Often ineffective for document-length texts. Works best on sequences of similar length.
- Ignores all lexical aspects.