Chapter NLP:III

III. Text Models

- □ Text Preprocessing
- Text Representation
- □ Text Similarity
- Text Classification
- □ Sequence Modeling

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 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.

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.

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.

\mathbf{S}_1	r	е	S	t	а	u	r	а	n	t	
\mathbf{S}_2	W	е	S	t	а	u	W	а	n	g	
Distance	1	0	0	0	0	0	1	0	0	1	= 3

Hamming distance with character removal and padding.

\mathbf{S}_1	r	е	S	t	а	u	r	а	n	t	
S_2	r	е	S	t	u	r	а	n	t	[P]	
Distance	0	0	0	0	1	1	1	1	1	1	= 6

String-based Similarity: Levenshtein Distance

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

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

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

 $\text{Levenshtein}(\textbf{s}_1,\textbf{s}_2)$

- 1. IF $|\mathbf{s}_2| = 0$ then *return* $(|\mathbf{s}_1|)$ endif // Do $|\mathbf{s}_1|$ deletions.
- 2. IF $|\mathbf{s}_1| = 0$ THEN *return* $(|\mathbf{s}_2|)$ ENDIF // Do $|\mathbf{s}_2|$ insertions.
- 3. IF $\mathbf{s}_1[0] = \mathbf{s}_2[0]$ THEN *return*(Levenshtein($\mathbf{s}_1[1:], \mathbf{s}_2[1:]$)) ENDIF
- 4. $\mathbf{l}_{del} = 1 + \text{Levenshtein}(\mathbf{s}_1[1:], \mathbf{s}_2)$ // Cost when doing a deletion.
- 5. $\mathbf{l}_{ins} = 1 + \text{Levenshtein}(\mathbf{S}_1, \mathbf{S}_2[1:])$ // Cost when doing an insertion.
- 6. $\mathbf{l}_{sub} = 1 + \text{Levenshtein}(\mathbf{s}_1[1:], \mathbf{s}_2[1:])$ // Cost when doing a substitution.
- 7. $return(min(l_{del}, l_{ins}, l_{sub}))$

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.

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.

 $\texttt{couch} \longleftrightarrow \texttt{sofa} \qquad \texttt{big} \longleftrightarrow \texttt{large}$

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

vehicle $\overrightarrow{Hypernym}$ car vehicle $\overrightarrow{Hypernym}$ ship

WordNet entry for nickel:

<u>S:</u> (n) nickel, <u>Ni</u>, <u>atomic number 28</u> (a hard malleable ductile silvery metallic element that is resistant to corrosion; used in alloys; occurs in pentlandite and smaltite and garnierite and millerite)

<u>S:</u> (n) nickel (a United States coin worth one twentieth of a dollar)

- *direct hypernym* / *inherited hypernym* / *sister term*
 - <u>S:</u> (n) <u>coin</u> (a flat metal piece (usually a disc) used as money)



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.

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 p and q represented as m-length (BoW) vectors.

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

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



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

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

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.

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

- 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.

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.

The smaller the angle, the more similar the vectors. The cosine is maximal for 0°.
 ||x|| denotes the <u>L2 norm</u> of vector x.

Vector Similarity: Cosine Similarity

Geometric interpretation of the Cosine:



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

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.

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



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

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.

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

$$sim_{\text{JSD}}(P(x) \parallel Q(x)) = 1 - \left(\frac{1}{2}D_{\text{KL}}(P(x) \parallel M(x)) + \frac{1}{2}D_{\text{KL}}(Q(x) \parallel M(x))\right)$$
$$M(x) = \frac{1}{2}(P(x) + Q(x))$$

 $\cap (v)$

Vector Similarity

Applications:

- □ Retrieval models in search. (cf. [IR:VI-92 ff.])
- □ Clustering documents. (cf. <u>Data Mining</u>)
- □ 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.

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

Idea: Compute a geometric similarity measure on a sentence embedding.

- □ Vector Average Similarity compares the geometric average of the word vectors.
 - \mathbf{S}^1 : Obama speaks to the media in Illinois

 \mathbf{S}^2 : The press is greeted by the President in Chicago

$$\textit{sim}_{\mathsf{cosine}}(rac{1}{|s^1|} \cdot \sum_{\mathbf{w}_i \in s^1} \mathbf{w}_i, rac{1}{|s^2|} \cdot \sum_{\mathbf{w}_j \in s^2} \mathbf{w}_j)$$



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

Idea: Compute a geometric similarity measure on a sentence embedding.

- □ Vector Average Similarity compares the geometric average of the word vectors.
 - \mathbf{S}^1 : Obama speaks to the media in Illinois
 - \mathbf{S}^2 : The press is greeted by the President in Chicago

$$\textit{sim}_{\mathsf{cosine}}(rac{1}{|s^1|} \cdot \sum_{\mathbf{w}_i \in s^1} \mathbf{w}_i, rac{1}{|s^2|} \cdot \sum_{\mathbf{w}_j \in s^2} \mathbf{w}_j)$$



Use any geometric measure on sentence embeddings. [NLP:III-71 ff.]
 Deep Average Networks, USE, BERT, ...

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.

```
\mathbf{S}^1: The President greets the press in Chicago \mathbf{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<sup>1</sup>: The President greets the press in Chicago
S<sup>2</sup>: 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.





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 $i \in s^1$ to words in $j \in s^2$ in an embedding space.

$$S^1$$
: The President greets the press in Chicago
 S^2 : Obama speaks to the media in Illinois

S³: The band gave a concert in Japan

 $WMD(s^{1}, s^{2}) = 0.45 + 0.24 + 0.20 + 0.18 = 1.07$ $WMD(s^{1}, s^{3}) = 0.49 + 0.42 + 0.44 + 0.28 = 1.63$



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 $i \in s^1$ to words in $j \in s^2$ in an embedding space.

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

□ The cost $c(i, j) = \|\mathbf{x}_i - \mathbf{x}_j\|_2$ is the euclidean distance.

- □ The flow matrix $T_{i,j}$ indicates which capacity of word *i* is moved to word *j*
- □ The outgoing flow of *i* must expend the capacity d_i^1 : $\sum_{i=1}^{n} \mathbf{T}_{i,j} = d_i^1$

The incoming flow into j must fill the capacity d_j^2 : $\sum_{j=1}^{n} \mathbf{T}_{i,j} = d_j^2$

□ The capacity is the term weight, i.e. $d_i^1 = \frac{1}{\|\mathbf{s}_1\|}$

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

Word Vector Similarity

Applications:

- □ Every time semantics is more important than form.
- □ Every time 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.