Word and String similarity

From Minimum Edit Distance to Similarity Learning

How can string similarity be measured?

- Intuitively, there are 2 variants:
  1. **<u>Orthographic:</u>** How similar are sequences of letters?

     e.g. “Hand” vs. “Land” ➔ 3 identical letters and 1 different

     ➔ Based on the spelling, these words would be very similar to each other

  2. **<u>Semantic:</u>** *How similar are the meanings of two words?*

     e.g. “Dog” vs. “Cat” ➔ no identical letters, but both describe a similar concept

For what purpose can it be used?

- Intuitively, there are 2 variants:
  1. **Orthographic:** How similar are sequences of letters?
     - Spelling verification
     - Tracing of old spellings ("thou" → "you")
     - Alignment of gene sequences

```
AGGCTATCACCTGACCTCCAGGCCGATGCCC          -AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---
TAGCTATCACGACCGCGGTCGATTTGCCCGAC    ➔    TAG-CTATCAC--GACCGC--GGTCGATTTGCCCGAC
```

  2. **Semantic:** *How similar are the meanings of two words?*
     - As features for supervised learning methods (e.g. Brown Cluster)
     - Correction of the content of free text answers

Word and String similarity

Orthographic Similarity

Word and string similarity: orthography

- Distinction between
  1. **Distance**
     Similar inputs have as small a distance as possible, dissimilar inputs have as large a distance as possible

  2. **Similarity**
     Similar inputs have as large a similarity as possible, dissimilar ones have as small a similarity as possible

Word and string similarity: edit distance

- *The edit distance calculates the distance between 2 input strings* s_1, s_2 over a set of operations, e.g.:
  - Number of removed letters (Operation: DELETE)
  - Number of newly inserted letters (Operation: INSERT)
  - Number of replaced letters (Operation: REPLACE)

- To get from s_1 to s_2
- The "Minimum Edit Distance" looks for the "best" sequence of operations to transform s_1 into s_2

Word and string similarity: edit distance

- How to find the "best sequence"?
  - For each operation op (DELETE, INSERT, REPLACE) a cost c_{op} is introduced
  - Looking for:

$$\min_{s \in valid\ Seq.} \sum_{op \in S} c_{op}(s_1, s_2)$$

  - Problem:
    - Number of valid sequences is exponential
    - ➔ Solution by means of dynamic programming

Definition of the Minimum Edit Distance

- For 2 strings
  - X of length *n*
  - Y of length *m*

- Define $D(i, j)$ as edit distance between X[1..*i*] and Y[1..*j*]
  - i.e. the first i letters of X and the first j letters of Y
  - The edit distance between X and Y is then: $D(n, m)$

Definition of the Minimum Edit Distance

- Define costs for each operation
  - INSERT → 1
  - DELETE → 1
  - REPLACE → 2

Levenshtein-Distance

- Other sets of costs lead to other edit distances
  - e.g. Damerau-Levenshtein introduces the operation SWAP

Algorithm: Levenshtein-distance

- Initialization

```
D(i,0) = i
D(0,j) = j
```

- Recurrence Relation:

```
For each  i = 1…m
     For each  j = 1…n
```

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 & \text{Insert} \\ D(i,j-1) + 1 & \text{Delete} \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } X(i) \neq Y(j) \\ 0; & \text{if } X(i) = Y(j) \end{cases} & \text{REPLACE} \end{cases}$$

- Termination:

```
D(n,m) is distance
```

Levenshtein distance - Table

j →

i ↓

	#	E	X	E	C	U	T	I	O	N
#	0	1	2	3	4	5	6	7	8	9
I	1									
N	2									
T	3									
E	4									
N	5									
T	6									
I	7									
O	8									
N	9									

- Initialization

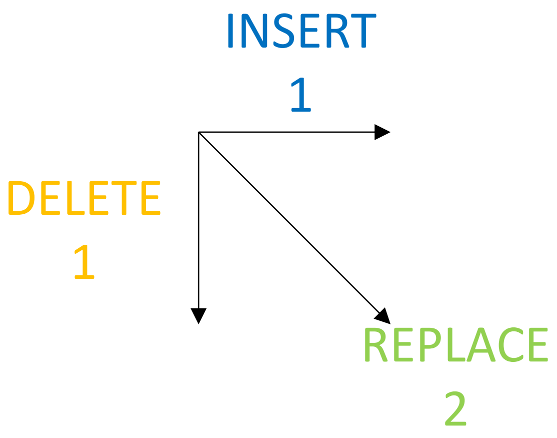
  $D(i,0) = i$

  $D(0,j) = j$

Levenshtein distance - Table

	#	E	X	E	C	U	T	I	O	N
#	0	1	2	3	4	5	6	7	8	9
I	1	2								
N	2									
T	3									
E	4									
N	5									
T	6									
I	7									
O	8									
N	9									

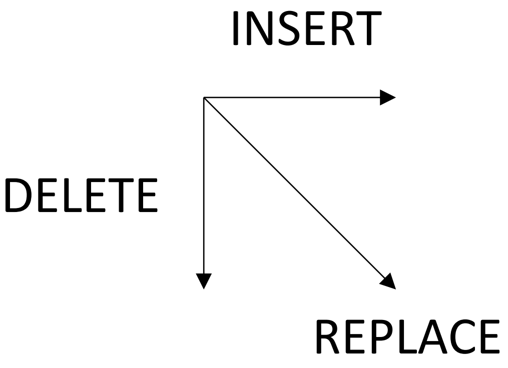
$$D(i,j) = min \begin{cases} D(i-1,j)+1 \\ D(i,j-1)+1 \\ D(i-1,j-1), if X[i]=Y[j] \\ D(i-1,j-1)+2, if X[i] \neq Y[j] \end{cases}$$



Levenshtein distance - Table

	#	E	X	E	C	U	T	I	O	N
#	0	1	2	3	4	5	6	7	8	9
I	1	2								
N	2	3								
T	3									
E	4									
N	5									
T	6									
I	7									
O	8									
N	9									

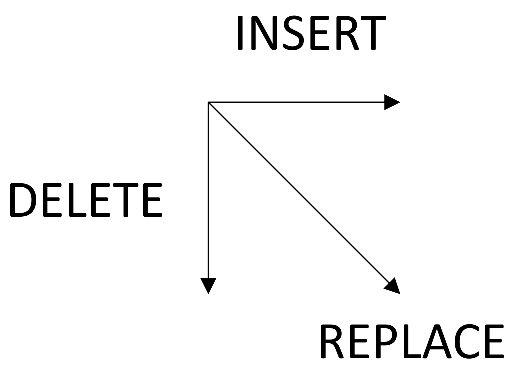
$$D(i,j) = min\begin{cases} D(i-1,j)+1 \\ D(i,j-1)+1 \\ D(i-1,j-1), if X[i] = Y[j] \\ D(i-1,j-1)+2, if X[i] \neq Y[j] \end{cases}$$



Levenshtein distance - Table

	#	E	X	E	C	U	T	I	O	N
#	0	1	2	3	4	5	6	7	8	9
I	1	2								
N	2	3								
T	3	4								
E	4	3								
N	5									
T	6									
I	7									
O	8									
N	9									

$$D(i,j) = min \begin{cases} D(i-1,j)+1 \\ D(i,j-1)+1 \\ D(i-1,j-1), if X[i]=Y[j] \\ D(i-1,j-1)+2\ , if X[i] \neq Y[j] \end{cases}$$

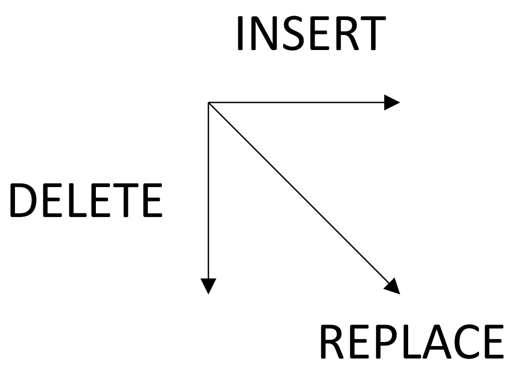


Levenshtein distance - Table

Levenshtein distance - Table

	#	E	X	E	C	U	T	I	O	N
#	0	1	2	3	4	5	6	7	8	9
I	1	2	3	4	5	6	7	6	7	8
N	2	3	4	5	6	7	8	7	8	7
T	3	4	5	6	7	8	7	8	9	8
E	4	3	4	5	6	7	8	9	10	9
N	5	4	5	6	7	8	9	10	11	10
T	6	5	6	7	8	9	8	9	10	11
I	7	6	7	8	9	10	9	8	9	10
O	8	7	8	9	10	11	10	9	8	9
N	9	8	9	10	11	12	11	10	9	8

$$D(i,j) = min \begin{cases} D(i-1,j)+1 \\ D(i,j-1)+1 \\ D(i-1,j-1), if X[i] = Y[j] \\ D(i-1,j-1)+2\,, if X[i] \neq Y[j] \end{cases}$$



Computing alignments

- The distance is not enough, we get the alignment by storing a backpointer when filling the cells!
- These backpointers are followed on the backward pass

Levenshtein distance - Best sequence

	#	E	X	E	C	U	T	I	O	N
#	0	1	2	3	4	5	6	7	8	9
I	1	2	3	4	5	6	7	6	7	8
N	2	3	4	5	6	7	8	7	8	7
T	3	4	5	6	7	8	7	8	9	8
E	4	3	4	5	6	7	8	9	10	9
N	5	4	5	6	7	8	9	10	11	10
T	6	5	6	7	8	9	8	9	10	11
I	7	6	7	8	9	10	9	8	9	10
O	8	7	8	9	10	11	10	9	8	9
N	9	8	9	10	11	12	11	10	9	8

- Determine the best sequence by backtracking

Complexity

- Time: O(nm)
- Space: O(nm)
- Backtrace: O(n+m)

Minimum Edit Distance

Weighted Minimum Edit Distance

Weighted Edit Distance

- Why would we add weights to the computation?
  - Spell Correction: some letters are more likely to be mistyped than others
  - Biology: certain kinds of deletions or insertions are more likely than others

sub[X, Y] = Substitution of X (incorrect) for Y (correct)

X	Y (correct)																								
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	0	0	7	1	342	0	0	2	118	0	1	0	0	3	76	0	0	1	35	9	9	0	1	0	5	0
b	0	0	9	9	2	2	3	1	0	0	0	5	11	5	0	10	0	0	2	1	0	0	8	0	0	0
c	6	5	0	16	0	9	5	0	0	0	1	0	7	9	1	10	2	5	39	40	1	3	7	1	1	0
d	1	10	13	0	12	0	5	5	0	0	2	3	7	3	0	1	0	43	30	22	0	0	4	0	2	0
e	388	0	3	11	0	2	2	0	89	0	0	3	0	5	93	0	0	14	12	6	15	0	1	0	18	0
f	0	15	0	3	1	0	5	2	0	0	0	3	4	1	0	0	0	6	4	12	0	0	2	0	0	0
g	4	1	11	11	9	2	0	0	0	1	1	3	0	0	2	1	3	5	13	21	0	0	1	0	3	0
h	1	8	0	3	0	0	0	0	0	0	2	0	12	14	2	3	0	3	1	11	0	0	2	0	0	0
i	103	0	0	0	146	0	1	0	0	0	0	6	0	0	49	0	0	0	2	1	47	0	2	1	15	0
j	0	1	1	9	0	0	1	0	0	0	0	2	1	0	0	0	0	0	5	0	0	0	0	0	0	0
k	1	2	8	4	1	1	2	5	0	0	0	0	5	0	2	0	0	0	6	0	0	0	4	0	0	3
l	2	10	1	4	0	4	5	6	13	0	1	0	0	14	2	5	0	11	10	2	0	0	0	0	0	0
m	1	3	7	8	0	2	0	6	0	0	4	4	0	180	0	6	0	0	9	15	13	3	2	2	3	0
n	2	7	6	5	3	0	1	19	1	0	4	35	78	0	0	7	0	28	5	7	0	0	1	2	0	2
o	91	1	1	3	116	0	0	0	25	0	2	0	0	0	0	14	0	2	4	14	39	0	0	0	18	0
p	0	11	1	2	0	6	5	0	2	9	0	2	7	6	15	0	0	1	3	6	0	4	1	0	0	0
q	0	0	1	0	0	0	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	0	14	0	30	12	2	2	8	2	0	5	8	4	20	1	14	0	0	12	22	4	0	0	1	0	0
s	11	8	27	33	35	4	0	1	0	1	0	27	0	6	1	7	0	14	0	15	0	0	5	3	20	1
t	3	4	9	42	7	5	19	5	0	1	0	14	9	5	5	6	0	11	37	0	0	2	19	0	7	6
u	20	0	0	0	44	0	0	0	64	0	0	0	0	2	43	0	0	4	0	0	0	0	2	0	8	0
v	0	0	7	0	0	3	0	0	0	0	0	1	0	0	1	0	0	0	8	3	0	0	0	0	0	0
w	2	2	1	0	1	0	0	2	0	0	1	0	0	0	0	7	0	6	3	3	1	0	0	0	0	0
x	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0
y	0	0	2	0	15	0	1	7	15	0	0	0	2	0	6	1	0	7	36	8	5	0	0	1	0	0
z	0	0	0	7	0	0	0	0	0	0	0	7	5	0	0	0	0	2	21	3	0	0	0	0	3	0

Weighted Min Edit Distance

- Initialization:

```
D(0,0) = 0
D(i,0) = D(i-1,0) + del[x(i)];     1 < i ≤ N
D(0,j) = D(0,j-1) + ins[y(j)];     1 < j ≤ M
```

- Recurrence Relation:

$$D(i,j) = \min \begin{cases} D(i-1,j) + del[x(i)] \\ D(i,j-1) + ins[y(j)] \\ D(i-1,j-1) + sub[x(i),y(j)] \end{cases}$$

- Termination:

```
D(N,M) is distance
```

2. String (subsequence) Kernel

Definition and Example

String (subsequence) Kernel

- The "string (subsequence) kernel" was first proposed by Lodhi in 2002
- Used for text classification:
  - two texts are more similar the more common substrings can be found
- **<u>Similarity measure</u>** instead of **<u>distance measure</u>**

- Characteristics:
  - Substrings can be unconnected, gaps are taken into account
  - Substring are assigned a weight that evaluates the degree of connection

String (subsequence) Kernel: Example

- Consider strings s
  - sectionalization
  - segmentation
- And a substring u
  - s-e-t
- Define a function $subseq(s, u)$ as follows:
  - $subseq(s, u)$ returns the closest indices of the letters from u in s
- Example:
  - $subseq(sectionalization, set) = [[1,2,4]]$
  - $subseq(segmentation, set) = [[1,2,7]]$
- Each detected substring provides a contribution λ to the similarity, corresponding to the length of the interval of $subseq(s, u)$

$$\lambda^{len([1,2,7])} = \lambda^7 \text{ , für } \lambda < 1$$

➔ The more non-related, the lower the contribution λ

String (subsequence) Kernel: Definition

- We define the kernel function K_n of two strings ***s*** and ***t*** :

$$K_n(s,t) = \sum_{u\in\Sigma^n} \langle\, \phi_u(s) \cdot \phi_u(t) \rangle$$
$$= \sum_{u\in\Sigma^n} \sum_{\boldsymbol{i}\in subseq(s,u)} \sum_{\boldsymbol{j}\in subseq(t,u)} \lambda^{l(\boldsymbol{i})+l(\boldsymbol{j})}$$

- Simply put
  - For each subsequence up to length n
    - <u>**Do:**</u> for any indexing **i** in s and **j** in t for the substring **u** **add** $\boldsymbol{\lambda}^{\boldsymbol{l}(\boldsymbol{i})+\boldsymbol{l}(\boldsymbol{j})}$ to the similarity

→ Efficient computation requires dynamic programming

String (subsequence) Kernel: Full Example

- Consider the strings “fog” and “fob”:

	f-o	f-g	o-g	f-b	o-b
$\phi(fog)$	λ^2	λ^3	λ^2	0	0
$\phi(fob)$	λ^2	0	0	λ^3	λ^2

- $k(fog, fog) = 2\lambda^4 + \lambda^6$
- $k(fob, fob) = 2\lambda^4 + \lambda^6$
- $k(fog, fob) = \lambda^4$

➔ For λ often 0.5 is assumed

Word and String similarity

Semantic Similarity

Semantic Similarity

- How can we measure **semantic** similarity between words?
  - E.g. what is $semSim(\text{"house"}, \text{"building"})$?

- To provide a meaningful answer to this question, we need data:
  1. Determine similarity using a large amount of text (e.g. Brown-Cluster)
  2. Determine the similarity using a thesaurus in which the words have been manually classified into a hierarchy (e.g. WordNet, GermaNet)
  3. Compute similarity using human labeled data
     (e.g. $semSim(\text{"house"}, \text{building}) = 8$, on a scale from 1 to 10)
     ➔ WordSim-353

Word and String similarity

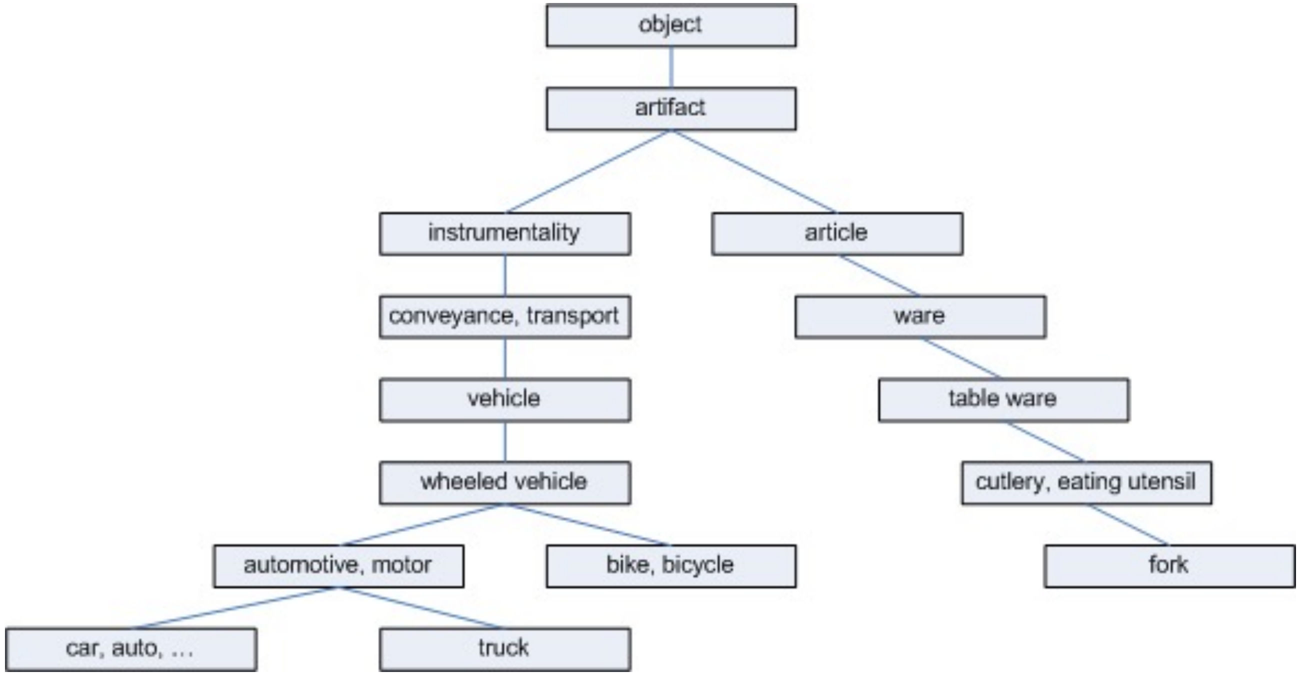
Semantic Similarity - WordNet

Semantic Similarity -WordNet

WordNet® is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations.
WordNet superficially resembles a thesaurus, in that it groups words together based on their meanings. However, there are some important distinctions. First, WordNet interlinks not just word forms—strings of letters—but specific senses of words. As a result, words that are found in close proximity to one another in the network are semantically disambiguated. Second, WordNet labels the semantic relations among words, whereas the groupings of words in a thesaurus does not follow any explicit pattern other than meaning similarity.

Semantic Similarity -WordNet

- For us, WordNet is a manually created hierarchy of concepts according to their semantics
- Excerpt:



Semantic Similarity -WordNet

- How can we use WordNet to determine semantic similarity between words (concepts)?

- Generally, 2 methods
  1. **Edge-based:**
     Calculate the similarity between 2 concepts over a (weighted) path between them
  2. **Node-based:**
     Calculate an information measure for each node and use it to calculate a similarity between 2 concepts

Semantic Similarity - Edge-based

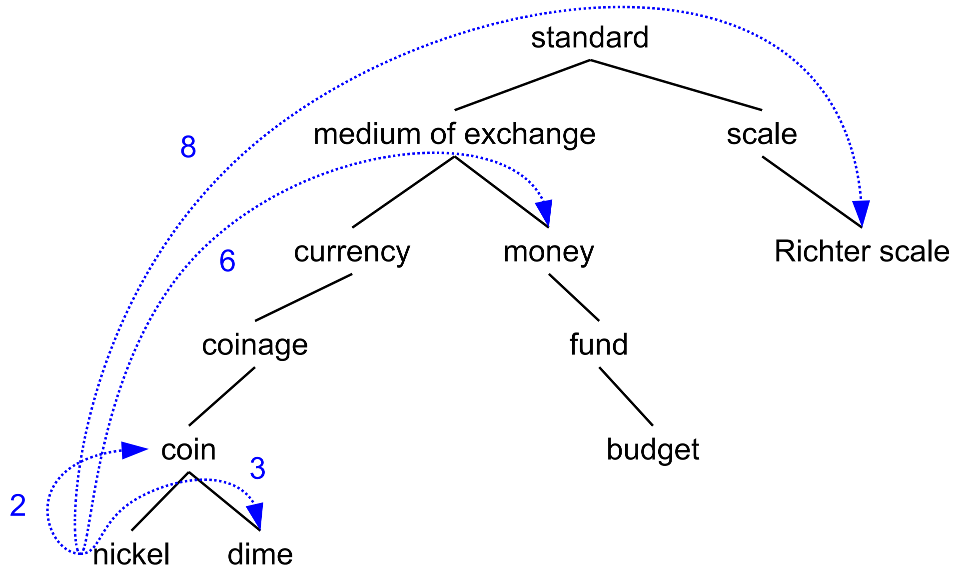
- The naivest approach to calculate a similarity is simply to count the number of edges on the shortest path between two concepts (c_1, c_2)

- <u>We define:</u>
  - $\mathrm{pathlen}(c_1, c_2)$ = 1 + Number of edges in the "hypernym graph" between nodes c_1 and c_2
  - $\mathrm{simpath}\,(c_1, c_2) = \frac{1}{pathlen(c_1,c_2)}$

$$wordsim(w_1, w_2) = \max_{c_1 \in \mathrm{senses}(w_1), c_2 \in \mathrm{senses}(w_2)} \mathrm{simpath}(c_1, c_2)$$

Example: path-based similarity

$\text{simpath}(c_1, c_2) = 1/\text{pathlen}(c_1, c_2)$



simpath(*nickel,coin*) = 1/2 = .5

simpath(*fund,budget*) = 1/2 = .5

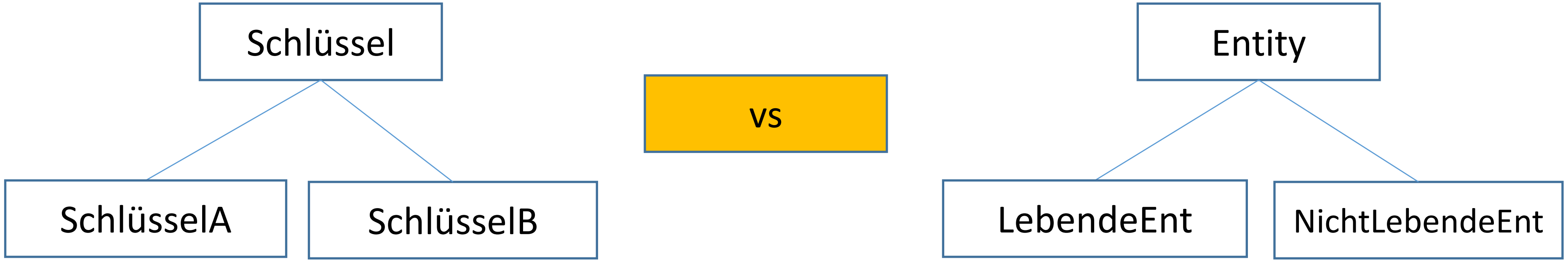
simpath(*nickel,currency*) = 1/4 = .25

simpath(*nickel,money*) = 1/6 = .17

simpath(*coinage,Richter scale*) = 1/6 = .17

Problems of the standard path-based similarity

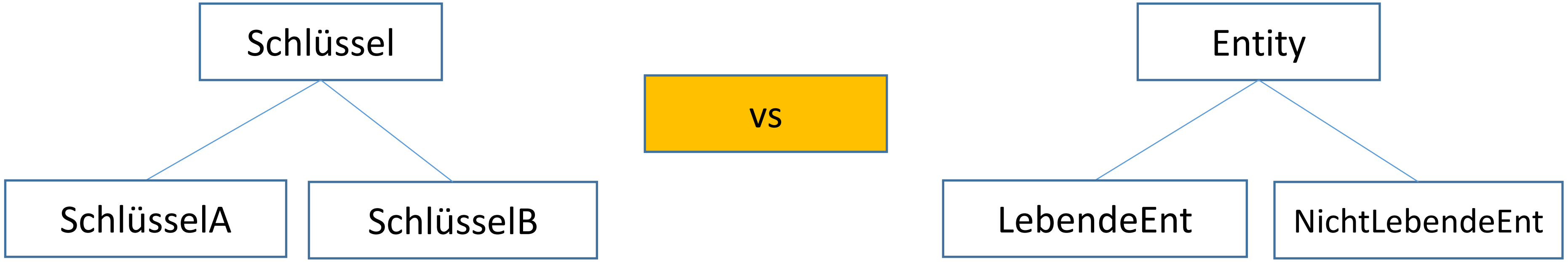
- No influence of the "network density”



➔ The influence of a parent node is distributed among its children, so that more children indicate a greater similarity

Problems of the standard path-based similarity

- No influence of the "network density”
- No influence of the node depth



➔ The deeper the node the more similar are the concepts

Problems of the standard path-based similarity

- No influence of the "network density”
- No influence of the node depth
- Same distances, regardless of the domain!

Bank ⇔ Money (Stock market domain)

Vs

Bank ⇔ Money (Landscaping)

➔ Influence of corpus statistics required!

Semantic Similarity – Integration of a corpus

- For domain specific assertions, a measure – **Information Content (IC)** – is specified for each node:

$$IC(c) = -\log(P(c))$$

- $P(c)$ describes the probability of encountering a concept c in a corpus

Semantic Similarity – Information Content

- To determine the IC, a labeled corpus of a domain must be obtained
- Marked with all concepts (and the corresponding node!) in the hierarchy

This text describes entities like pizza, ice cream or simply buildings, which are sometimes like trees but sometimes also simply like a path in search of knowledge

Semantic Similarity – Information Content

- To determine the IC, a labeled corpus of a domain must be obtained
- Marked with all concepts (and the corresponding node!) in the hierarchy
- For each concept we now determine the frequency of its occurrence

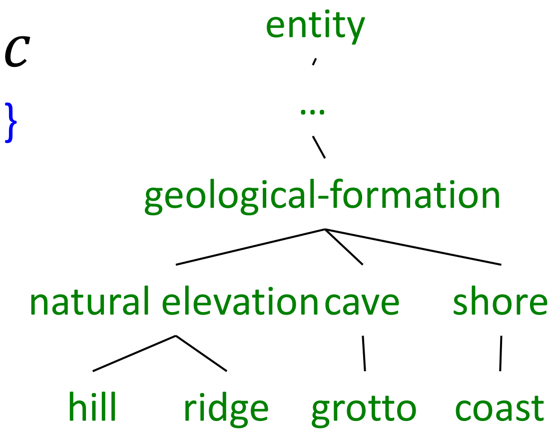
$$freq(c) = \sum_{w \in words(c)} freq(w)$$

- The words of a concept are all the direct words and all the words of the descendants
- This determines its probability

$$\mathrm{P}(c) = \frac{freq(c)}{N}$$

Information content similarity

- Train by counting in a corpus
  - Each instance of `hill` counts toward frequency of *natural elevation*, *geological formation*, *entity*, etc.
  - Let $words(c)$ be the set of all words that are children of node c
    - $words$("geo-formation") = {hill,ridge,grotto,coast,cave,shore,natural elevation}
    - $words$("natural elevation") = {hill, ridge}

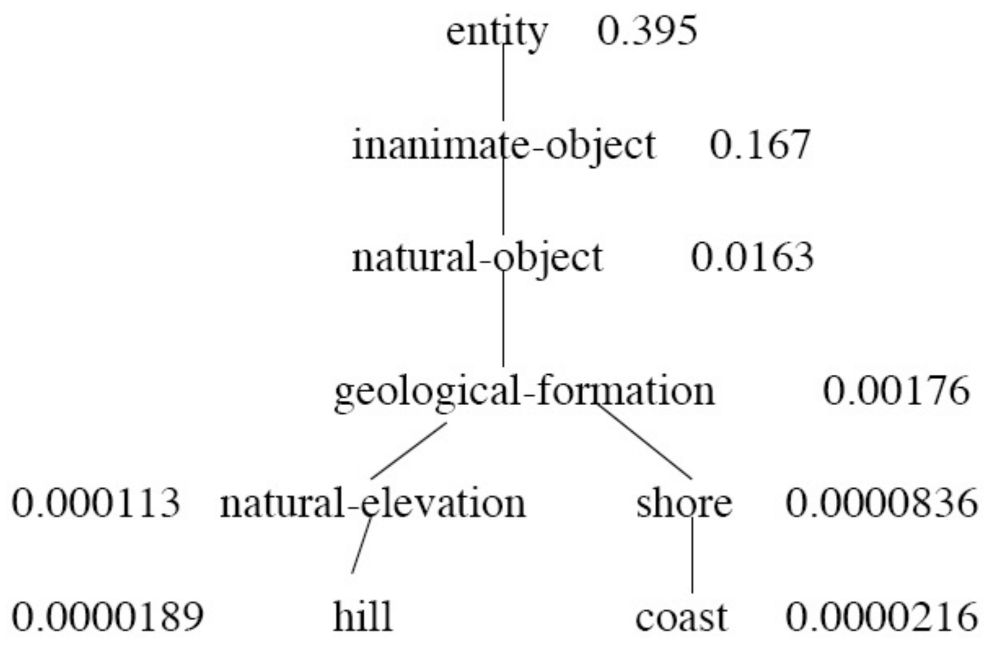


$$\mathrm{P}(c) = \frac{\sum_{w \in words(c)} freq(w)}{N}$$

Information content similarity

- WordNet hierarchy augmented with probabilities P(c)

D. Lin. 1998. An Information-Theoretic Definition of Similarity. ICML 1998



Insertion: Information content and probability

- The **self-information** of an event, also called its **surprisal**
  - How surprised we are to know it; how much we learn by knowing it
  - The more surprising something is, the more it tells us when it happens
  - We'll measure self-information in **bits**.

$$I(w) = -\log 2\, P(w)$$

- An example: Coin-flip
  - $P(heads) = 0.5$
  - How many bits of information do I learn by flipping it?

$$I(heads) = -\log 2(0.5) = -\log 2\;(1/2) = \log 2\;(2) = 1\; bit$$

  - I flip a biased coin: $P(heads\;) = 0.8$ I don't learn as much

$$I(heads) = -\log 2(0.8) = -\log 2(0.8) = .32\; bits$$

Information content: definitions

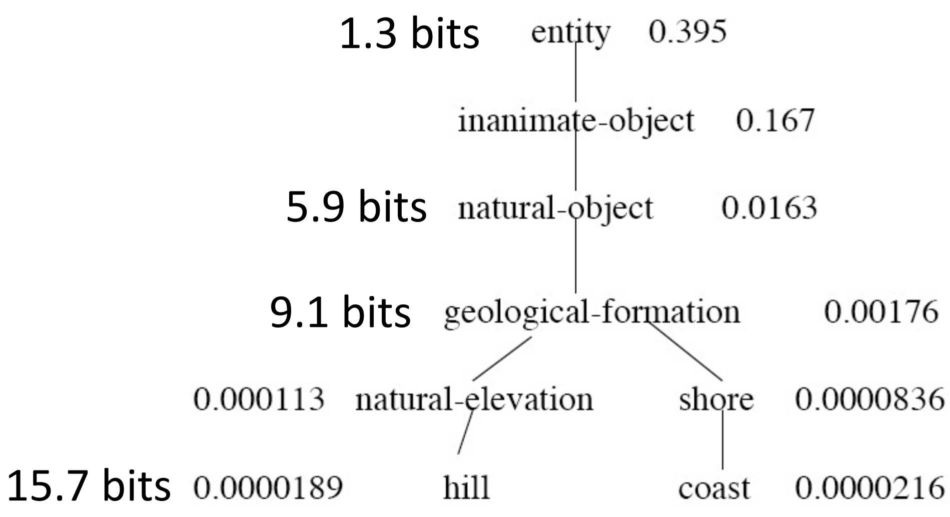
- Information content:

$$IC(c) = -\log P(c)$$

- Most informative subsumer (Lowest common subsumer)

  $LCS(c_1,c_2)$

  The most informative (lowest) node in the hierarchy subsuming both c_1 and c_2



Using information content for similarity: Resnik method

Philip Resnik. 1995. Using Information Content to Evaluate Semantic Similarity in a Taxonomy. IJCAI 1995.
Philip Resnik. 1999. Semantic Similarity in a Taxonomy: An Information-Based Measure and its Application to Problems of Ambiguity in Natural Language. JAIR 11, 95-130.

- The similarity between two words is related to their common information
- The more two words have in common, the more similar they are
- Resnik: measure common information as
  - The information content of the most informative (lowest) subsumer (MIS/LCS) of the two nodes
  - $sim_{resnik}(c_1, c_2) = -\log P(\, LCS(c_1, c_2)\,)$

Dekang Lin method

Dekang Lin. 1998. An Information-Theoretic Definition of Similarity. ICML

- Intuition: Similarity between A and B is not just what they have in common
- The more **differences** between A and B, the less similar they are:
  - Commonality: the more A and B have in common, the more similar they are
  - Difference: the more differences between A and B, the less similar
- Commonality: $IC(common(A, B))$
- Difference: $IC(description(A, B) - IC(common(A, B))$

Dekang Lin similarity theorem

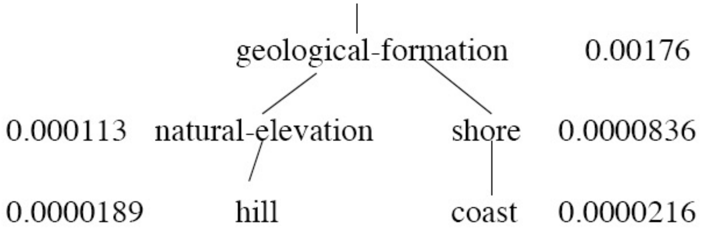
- The similarity between A and B is measured by the ratio between the amount of information needed to state the commonality of A and B and the information needed to fully describe what A and B are

$$sim_{Lin}(A,B) \propto \frac{IC(common(A,B))}{IC(description(A,B))}$$

- Lin alter Resnik and define $IC\big(common(A,B)\big)$ as $2 *$ information of the LCS

$$sim_{Lin}(c_1,c_2) = \frac{2\log P(LCS(c_1,c_2))}{\log P(c_1) + \log P(c_2)}$$

Lin similarity function



$$sim_{Lin}(A,B) = \frac{2\log P(LCS(c_1,c_2))}{\log P(c_1) + \log P(c_2)}$$

$$\begin{aligned} sim_{Lin}(\text{hill}, \text{coast}) &= \frac{2\log P(\text{geological-formation})}{\log P(\text{hill}) + \log P(\text{coast})} \\ &= \frac{2\ln 0.00176}{\ln 0.0000189 + \ln 0.0000216} \\ &= .59 \end{aligned}$$

Sim Jiang Conrath similarity

$$\text{sim}_{\text{jiang-conrath}}(c_1, c_2) = \frac{1}{2\log P(LCS(c_1, c_2)) - \log P(c_1) - \log P(c_2)}$$

Summary: thesaurus-based similarity

$$\text{sim}_{\text{path}}(c_1,c_2) = \frac{1}{pathlen(c_1,c_2)}$$

$$\text{sim}_{\text{resnik}}(c_1,c_2) = -\log P(LCS(c_1,c_2)) \quad \text{sim}_{\text{lin}}(c_1,c_2) = \frac{2\log P(LCS(c_1,c_2))}{\log P(c_1)+\log P(c_2)}$$

$$\text{sim}_{\text{jiang-conrath}}(c_1,c_2) = \frac{1}{2\log P(LCS(c_1,c_2)) - \log P(c_1) - \log P(c_2)}$$

Libraries for computing thesaurus-based similarity

- NLTK
  - http://nltk.github.com/api/nltk.corpus.reader.html?highlight=similarity - nltk.corpus.reader.WordNetCorpusReader.res_similarity

- WordNet::Similarity
  - http://wn-similarity.sourceforge.net/
  - Web-based interface: http://marimba.d.umn.edu/cgi-bin/similarity/similarity.cgi

Evaluating similarity

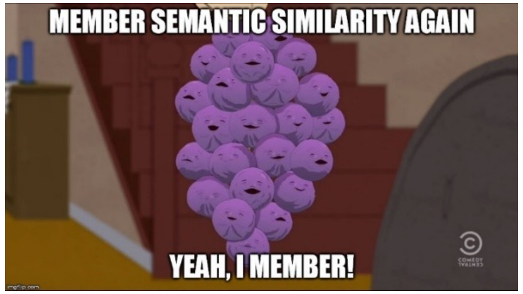
- Extrinsic (task-based, end-to-end) evaluation
  - Question answering
  - Spell checking
  - Essay grading

- Intrinsic evaluation
  - Correlation between algorithm and human word similarity ratings
    - Wordsim353: 353 noun pairs rated 0-10: $sim(plane, car) = 5.77$
  - Taking TOEFL multiple-choice vocabulary tests
    - Levied is closest in meaning to: imposed, believed, requested, correlated

Word and String similarity II

Metric Learning

Semantic Similarity



- How can we measure **semantic** similarity between words?
  - E.g. what is $semSim(\text{"house"}, \text{"building"})$?

- To provide a meaningful answer to this question, we need data:
  1. Determine similarity using a large amount of text (e.g. Brown-Cluster)
  2. Determine the similarity using a thesaurus in which the words have been manually classified into a hierarchy (e.g. WordNet, GermaNet)
  3. Compute similarity using human labeled data
     (e.g. $semSim(\text{"house"}, \text{building}) = 8$, on a scale from 1 to 10)
     ➔ WordSim-353

WordSim-353

- Human-generated rating of 353 word pairs
- Evaluation of the similarity between
  - 0 („totally unrelated words“)
  - 10 (“very much related or identical words”)
- Divided into 2 sets:
  - Set1, labeled by 13 people, 153 word pairs
  - Set2, labeled by 16 people, 200 word pairs

problem,challenge,6.75
size,prominence,5.31
country,citizen,7.31
planet,people,5.75
development,issue,3.97
experience,music,3.47
music,project,3.63
glass,metal,5.56

Metric Learning

- Also called "Similarity Learning"
- There are 3 "classic" setups
  1. "Regression Similarity Learning"
     - *Given:* labeled pairs (x_1, x_2) with a similarity $\mathrm{y_i} \in \mathbb{R}$
     - *Wanted:* A function, with $f(x_1, x_2) \sim y_i$ for each triple
  2. "Classification Similarity Learning"
     - *Given:* A set S ("similar objects") and a set D ("dissimilar objects") together with a label $y_i \in \{0,1\}$
     - *Wanted:* A classifier that predicts the label for new triples
  3. "Ranking Similarity Learning"
     - *Given:* Object triplet (x_i, x_i^+, x_i^-), where $f(x_i, x_i^+) > f(x_i, x_i^-)$
     - *Wanted:* The function f

Metric Learning

- What does such a function $f(x_1, x_2)$ look like?
- It is one of the classical distance functions, e.g.
  - Euklidean distance $f(x_1, x_2) = \sqrt{(x_1 - x_2)^T (x_1 - x_2)}$
  - Cosine similarity $f(x_1, x_2) = \frac{x_1^T x_2}{\sqrt{x_1^T x_1}\sqrt{x_2^T x_2}}$
- However, parameterised with a matrix M:
  - Euklidean distance $f(x_1, x_2) = \sqrt{(x_1 - x_2)^T M (x_1 - x_2)}$ } Mahalanobis distance
  - Cosine similarity $f(x_1, x_2) = \frac{x_1^T M x_2}{\sqrt{x_1^T M x_1}\sqrt{x_2^T M x_2}}$

Interpretation of Metric Learning

$$f(x_1, x_2) = \sqrt{(x_1 - x_2)^T M (x_1 - x_2)}$$

$$= f(x_1, x_2) = \sqrt{(x_1 - x_2)^T \underbrace{L^T L}_{\boxed{\text{Cholesky decomposition}}} (x_1 - x_2)}$$

- Note: For the Cholesky decomposition, the matrix M must be symmetric and positive (semi-)definite!

→ We leave that to the optimisation procedure!

Interpretation of Metric Learning

$$f(x_1, x_2) = \sqrt{(x_1 - x_2)^T M (x_1 - x_2)}$$

$$=f(x_1, x_2) = \sqrt{(x_1 - x_2)^T L^T L (x_1 - x_2)}$$

$$=f(x_1, x_2) = \sqrt{(Lx_1 - Lx_2)^T (Lx_1 - Lx_2)}$$

Interpretation of Metric Learning

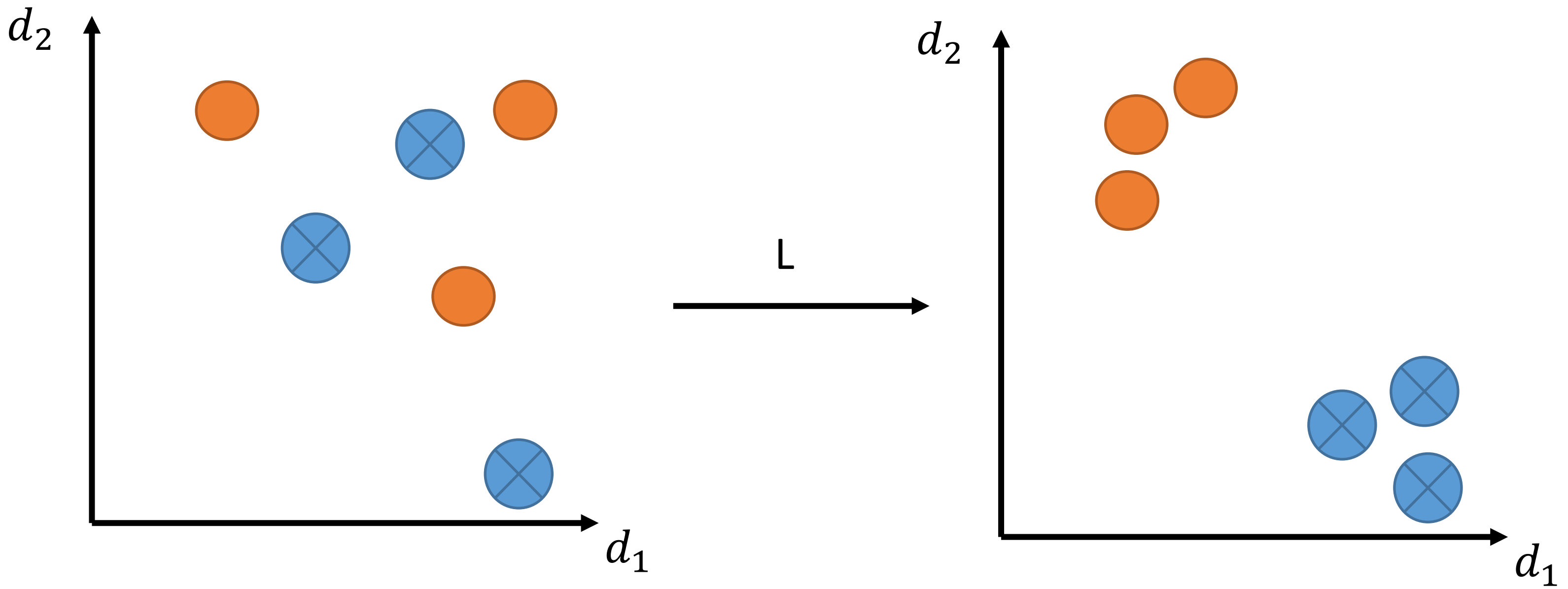
$$f(x_1, x_2) = \sqrt{(x_1 - x_2)^T M (x_1 - x_2)}$$
$$=f(x_1, x_2) = \sqrt{(x_1 - x_2)^T L^T L (x_1 - x_2)}$$
$$=f(x_1, x_2) = \sqrt{(Lx_1 - Lx_2)^T (Lx_1 - Lx_2)}$$

- Set $\widehat{x_1} = Lx_1$ and $\widehat{x_2} = Lx_2$

➔ $f(x_1, x_2) = \sqrt{(\widehat{x_1} - \widehat{x_2})^T (\widehat{x_1} - \widehat{x_2})}$

Metric learning is therefore nothing more than learning a (linear) transformation and then applying a standard distance metric

Interpretation of Metric Learning



→ For this to be possible in a linear way, you need a suitable representation of your data!
→ Subsequent classification e.g. by means of k-NN

Process of Metric Learning

- Training:

  - Input data
  - Distance metric

  + Brown-Clustering
  + Word2Vec
  + GloVe

  → Vektor representation → Learning of matrix M

- Application:

  - Input data
  - Matrix M

  → Transformation through M into the new space → Output similarity

Learning the Matrix M

- Having understood the procedure, how do we learn the matrix M?

→ We set up an optimization problem

- Example: MMC (Euclidean Distance)

$$\max_{M \geq 0} \sum\nolimits_{(x_i, x_j) \in D} d_M(x_i, x_j)$$

so dass:

$$\sum\nolimits_{(x_i, x_j) \in S} d_M(x_i, x_j) \leq 1$$

- In words: Maximize the distance of dissimilar pairs D, and ensure that all similar pairs S get a distance ≤ 1

Learning the Matrix M – Projected Gradient Ascent

- Idea: Transform the problem first

$$\max_{M} \sum_{(x_i, x_j) \in D} d_M(x_i, x_j)$$

so dass:

$$\sum_{(x_i, x_j) \in S} d_M(x_i, x_j) \leq 1$$

$$M \geq 0$$

- Apply Gradient Ascent for the maximization and correct the result so that the constraints are preserved

Metric Learning at the Lecture Chair

Relative Relatedness Learning

Relative Relatedness Learning

- RRL learns a matrix M to **parameterize the cosine measure**

$$\cos_M(x, y) := x^T M y \cdot (\|x\|_M \|y\|_M)^{-1}$$

- such that it satisfies a set of **relative relatedness** constraints

$$\mathcal{C} := \{(x, x', y, y') : rel(x, x') > rel(y, y')\}$$

Relative Relatedness Learning
Constraint Generation

Money – Bank	0.850
Alcohol – Chemistry	0.554
Drink – Ear	0.131

(Money, Bank, Alcohol, Chemistry)	$0.850 > 0.554$
(Money, Bank, Drink, Ear)	$0.850 > 0.131$
(Alcohol, Chemistry, Drink, Ear)	$0.554 > 0.131$

Relative Relatedness Learning
The Algorithm

(Money, Bank, Alcohol, Chemistry)	$0.850 > 0.554$
(Money, Bank, Drink, Ear)	$0.850 > 0.131$
(Alcohol, Chemistry, Drink, Ear)	$0.554 > 0.131$

Data: $\mathbf{V} \subset \mathbb{S}^{n-1}$: word vectors; $\mathcal{H}$: semantic relatedness dataset (e.g. MEN); learning rate l

Result: a relatedness matrix M for Equation (1)

Let $M := I_n$

while M *not converged* **do**

$$\text{loss}(M) = \sum_{\mathcal{C}(\mathcal{H})} 0.5 \cdot \left(\max\left\{0, \sqrt{1-\cos_M(w_1, w_1')} - \sqrt{1-\cos_M(w_2, w_2')}\right\}\right)^2 + \frac{\text{tr}(M) - \log\det(M)}{n^2}$$

$$M \leftarrow M - l \cdot \nabla \text{loss}(M)$$

$$M \leftarrow \min_{M'} \{\|M - M'\|_{\mathcal{F}} | M' \in PSD\}$$

end

return M

Experiments and Results: ConceptNet + MEN