INF4820: Algorithms for Artificial Intelligence and Natural Language Processing

Semantic Spaces

Milen Kouylekov & Stephan Oepen

Language Technology Group (LTG)

September 17, 2014

Today

- Distributional semantics
- Vector spaces: Spatial models for representing data
- Semantic spaces

Example

- Let's build Jeopardy!
  - Question: Who invented the electricity?
  - Answer: In about 600 BC, the Ancient Greeks discovered that rubbing fur on amber caused electric current.
- One of the first problems to solve is similarity.

Today and the next lectures

- Can a program automatically learn which words have similar meanings?
  - Just by looking at data of actual language use?
  - Without any prior knowledge?
- How can we represent word meaning in a mathematical model?

The Distributional Hypothesis

AKA The Contextual Theory of Meaning

- *Meaning is use.* (Wittgenstein, 1953)
- *You shall know a word by the company it keeps.* (Firth, 1957)
- *The meaning of entities, and the meaning of grammatical relations among them, is related to the restriction of combinations of these entities relative to other entities.* (Harris, 1968)

He was hungover after drinking too many shots of **retawerif** at the party last night.

The Distributional Hypothesis (cont'd)

- The hypothesis: If two words share similar contexts, we can assume that they have similar meanings.
- Comparing meaning reduced to comparing contexts,
  — no need for prior knowledge!
- Given the processing power of modern computers and the availability of vast amounts of electronic texts...
- ...we can now implement in practice the classic empiricist claims of Firth, Harris, Wittgenstein, et al.

Distributional semantics in practice

A distributional approach to lexical semantics:

- Record contexts of words across a large collection of texts (corpus).
- Each word o_i is represented by a set of features $\{x_1, \ldots, x_n\}$.
- Each feature x_j records some property of the observed contexts of o_i.
- Words that are found to have similar features are expected to also have similar meaning.

- But before we start looking at the details of how to compare the context features for words, a couple of design decisions;
  - How do we define '*context*'?
  - How do we define a '*word*'?

Defining ‘context’

- Let’s say we’re extracting features for the target *bread* in:

> I bake bread for breakfast.

Context windows

- Context = neighborhood of $\pm n$ words left/right of the focus word.
- Features for ± 1: {`left:bake`, `right:for`}
- Some variants: distance weighting, $n$grams.

Bag-of-Words

- BoW; include all co-occurring words, ignoring the linear ordering.
- Features: {`I`, `bake`, `for`, `breakfast`}
- Some variants: sentence-level, document-level.

Defining 'context' (cont'd)

I bake bread for breakfast.

Grammatical context

- Context = the grammatical relations to other words.
- Intuition: When words combine in a construction they often impose semantic constraints on each-other.
- Requires deeper linguistic analysis than simple BoW approaches.
- Features: {`dir_obj(bake)`, `prep_for(breakfast)`}

What is a word?

Raw:	The programmer's programs had been programmed.
Tokenized:	the programmer 's programs had been programmed .
Lemmatized:	the programmer 's program have be program .
W/ stop-list:	programmer program program
Stemmed:	program program program

- Tokenization: Splitting a text into sentences and words or other units.
- Different levels of abstraction and morphological normalization:
  - What to do with case, numbers, punctuation, compounds, ...?
  - Full-form words vs. lemmas vs. stems ...
- Stop-list: filter out closed-class words or function words.
  - The idea is that only *content words* provide relevant context.

Different contexts → different similarities

- What do we mean by *similar*?
- The type of context dictates the type of semantic similarity.
- 'Relatedness' *vs.* 'sameness'. Or domain vs. content.
- Similarity in domain: {*car, road, gas, service, traffic, driver, license*}
- Similarity in content: {*car, train, bicycle, truck, vehicle, airplane, buss*}
- While broader definitions of context tend to give clues for *domain-based relatedness*, more fine-grained and linguistically informed contexts give clues for *content-based similarity*.

Representation / model

- We've outlined the distributional approach to word meaning.
- But how exactly should we represent our words and context features?
- How exactly can we the compare features of different words?

Vector space model

- A general model for representing data based on a spatial metaphor.
- Each object is represented as a vector (or point) positioned in a coordinate system.
- Each coordinate (or dimension) of the space corresponds to some descriptive and measurable property (feature) of the objects.
- To measure similarity of two objects, we can measure their geometrical distance / closeness in the model.
- Vector representations are foundational to a wide range of ML methods.

Semantic spaces

- AKA distributional semantic models or word space models.
- A semantic space is a vector space model where
- points represent words,
- dimensions represent context of use,
- and distance in the space represents semantic similarity.
- How do we define the vector values?
- How do we measure distance?

Feature vectors

- A vector space model is defined by a system of n dimensions — objects are represented as real valued vectors in the space $\Re^n$.
- Our observations contextual features must be encoded numerically:
  - Each context feature is mapped to a dimension $j \in [1, n]$.
  - For a given word, the value of a given feature is its number of co-occurrences for the corresponding context across our corpus.
- Let the set of n features describing the lexical contexts of a word o_i be represented as a feature vector $\vec{x}_i = \langle x_{i1}, \ldots, x_{in} \rangle$.

Example

- If we assume that
- the ith word is *cake* and
- the jth feature is OBJ_OF(`bake`), then
- $x_{ij} = 4$ would mean that we have observed *cake* to be the object of the verb *bake* in our corpus 4 times.

Euclidean distance

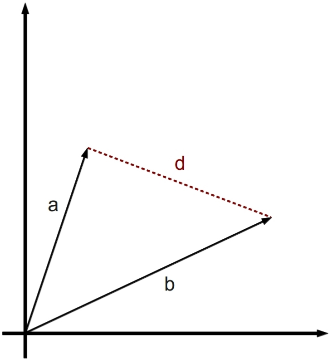
- We can now compute *semantic similarity* in terms of *spatial proximity*.
- One standard metric for this is the *Euclidean distance*:

$$d(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^{n} (\vec{x}_i - \vec{y}_i)^2}$$

- Computes the norm (or *length*) of the *difference* of the vectors.
- The norm of a vector is:

$$\|\vec{x}\| = \sqrt{\sum_{i=1}^{n} \vec{x}_i^2} = \sqrt{\vec{x} \cdot \vec{x}}$$

- Intuitive interpretation: The distance between two points corresponds to the length of the straight line connecting them.



Euclidean distance - Example

Let's have two vectors:

- `food [2, 2, 0, 0 , 1]`
- `drink [1, 3, 0, 1 , 1]`

Eucledean distance =

```
(sqrt (+ (expt (- 2 1) 2) (expt (- 2 3) 2)
         (expt (- 0 0) 2) (expt (- 0 1) 2)
         (expt (- 1 1) 2)))
```

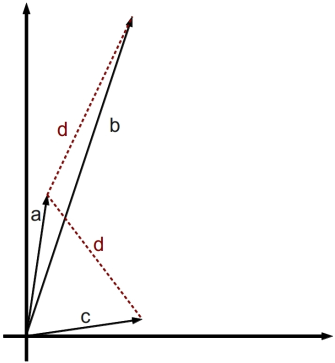
→ `1.7320508`

|food| =

```
(sqrt (+ (expt 2 2) (expt 2 2)
         (expt 0 2) (expt 0 2)
         (expt 1 2)))
```

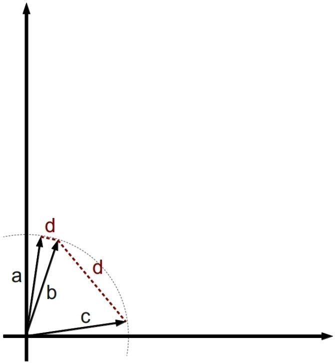
→ `3.0`

Euclidean distance and length bias



- However, a potential problem with Euclidean distance is that it is very sensitive to extreme values and the length of the vectors.
- As vectors of words with different *frequencies* will tend to have different length, the frequency will also affect the similarity judgment.

Overcoming length bias by normalization



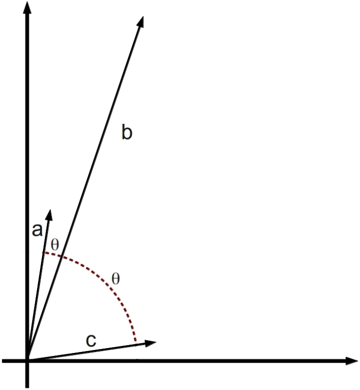
- One way to reduce frequency effects is to first normalize all our vectors to have unit length, i.e. $\|\vec{x}\| = 1$
- Can be achieved by simply dividing each element by the length: $\vec{x}\frac{1}{\|\vec{x}\|}$
- Amounts to all vectors pointing to the surface of a unit sphere.

Cosine similarity

- Another way to deal with length bias: use the *cosine* measure.
- Computes similarity as a function of the angle between the vectors:

$$\cos(\vec{x}, \vec{y}) = \frac{\sum_i \vec{x}_i \vec{y}_i}{\sqrt{\sum_i \vec{x}_i^2} \sqrt{\sum_i \vec{y}_i^2}} = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\| \|\vec{y}\|}$$

- Constant range between 0 and 1.
- Avoids the arbitrary scaling caused by dimensionality, frequency, etc.
- As the angle between the vectors shortens, the cosine approaches 1.



Cosine similarity (cont'd)

- For *normalized* (unit) vectors, the cosine is simply the *dot product*:

  $\cos(\vec{x}, \vec{y}) = \vec{x} \cdot \vec{y} = \sum_{i=1}^{n} \vec{x}_i \vec{y}_i$

- Can be computed very efficiently.
- Note; the cosine measures *proximity* rather than *distance*.
- The *same relative rank order* as the Euclidean distance for unit vectors!

Word–context association

- Problem: Raw co-occurrence frequencies are not always the best indicators of relevance.
- Imagine we have some features recording information about direct objects and we've collected the following counts for the noun *wine*:
  - OBJ_OF$(\texttt{buy}) = 14$
  - OBJ_OF$(\texttt{pour}) = \ 8$
  - ... but the feature OBJ_OF$(\texttt{pour})$ seems more indicative of the semantics of *wine* than OBJ_OF$(\texttt{buy})$.
- Solution: Weight the counts by an *association function*, "normalizing" our observed frequencies for chance co-occurrence.
- A range of different tests of statistical are used; e.g. *pointwise mutual information*, *log odds ratio*, *the t-test*, *log likelihood*, ...
- Note: We'll skip this step in our implementation (assignment 2a).

Practical comments: Numerical representations

- The input for our model is symbolic and categorical, with both words and feature types being strings.
- Model internally we want to work with *numerical identifiers* instead.
  - Means we can store and index the data more compactly and efficiently.
  - Most learning algorithms expect numerical input.
- For our vector space model, you should therefore implement functionality for mapping strings to integer ids (and back to strings).
- Sometimes called a *symbol table*.
- For generality, maintain separate mappings for words and features.

Practical comments: Sparsity

- Conceptually, a vector space is often thought of as a matrix.
  - Dimensions correspond to columns; each feature vector is a row.
  - For m words and n features we have an $m \times n$ co-occurrence matrix.
- Note; although the space will be extremely high-dimensional, the number of *non-zero* elements will be very low.
- Few active features per word.
- We say that the vectors are sparse.
- This has implications for how to implement our data structures and vector operations:
- Don't want to waste space representing zero-valued features.
- Don't want to waste time iterating over zero-valued features.

Practical comments: Data structures

- Given the comments about sparsity and the preference for working with numerical identifiers:
- What is a good Lisp data type for implementing our co-occurrence matrix / feature vectors?
  - Association lists, hash-tables, multi-dimensional arrays, array of arrays, list of arrays, array of hash-tables, hash-table of hash-tables, arrays of lists, . . . ?

Practical comments: Vector operations

- In theory, you can view formulas like Euclidean norm and cosine as "pseudo-code" that you can translate directly into Lisp.
- But again; our feature vectors are sparse.
- Taken directly, a formula like the Euclidean norm requires iterating over every dimension n in our space.
- But we don't want to waste time iterating over zero elements if we don't have to!

Tommorrow

- Computing neighbor relations in the semantic space
- Representing classes
- Representing class membership
- Classification algorithms: KNN-classification / c-means, etc.

Firth, J. R. (1957). A synopsis of linguistic theory 1930–1955. In *Studies in linguistic analysis.* Philological Society, Oxford.

Harris, Z. S. (1968). *Mathematical structures of language*. New York: Wiley.

Wittgenstein, L. (1953). *Philosophical investigations*. Oxford: Blackwell.