CS 4650/7650, Lecture 19
Vector-space models for lexical semantics

Jacob Eisenstein

October 31, 2013

1 Recap of semantics so far

- **Compositional semantics**
  - assemble the meaning of a sentence from its components
  - *What state borders Texas?* $\rightarrow \lambda x.\textsc{state}(x) \wedge \textsc{borders}(x, \textsc{Texas})$
- **Shallow semantics**
  - identify the key predicates and arguments in sentences
  - $[_{agent}$ Doris$]$ **gave** $[_{goal}$ Cary$]$ $[_{theme}$ the book$]$.
- **Today: lexical semantics**
  vector-space models for the meaning of individual words

A recurring theme in this course is that the mapping from words to meaning is complex.

- **Word sense disambiguation**: multiple meanings for the same form (e.g., *bank*)
- **Morphological analysis**: shared semantic basis among multiple forms (e.g., *speak*, *spoke*, *speaking*)
- **Synonymy**: in English we have lots of synonyms and near neighbors, as English combines influence from lots of other languages (French, Latin, German, etc)
- Both **compositional** and **frame** semantics assume hand-crafted resources that map from words to predicates.

How do we do semantic analysis of words that we've never seen before?

2 The distributional hypothesis

Here's a word you may not know: *tezgüino*. If we encounter this word, what can we do? It seems like a big problem for any NLP system, from POS tagging to semantic analysis.

Suppose we see that *tezgüino*is used in the following contexts:

1. A bottle of __________ is on the table.
2. Everybody likes __________.
3. Don't have __________ before you drive.
4. We make __________ out of corn.

What other words fit into these contexts? How about: *loud, motor oil, tortillas, choices, wine*?

We can create a vector for each word, based on whether it can be used in each context.

	C1	C2	C3	C4	...
tezgüino	1	1	1	1	
loud	0	0	0	0	
motor oil	1	0	0	1	
tortillas	0	1	0	1	
choices	0	1	0	0	
wine	1	1	1	1	

- Based on these vectors, we see:
  - *wine* is very similar to *tezgüino*
  - *motor oil* and *tortillas* are fairly similar to *tezgüino*
  - *loud* is quite different.
- The vectors describe the **distributional** properties of each word.
- Does vector similarity imply semantic similarity? This is the **distributional hypothesis.** "You shall know a word by the company it keeps." (Firth 1957)

- It is also known as a **vector-space model**, since each word's meaning is captured by a vector.

Vector-space models and distributional semantics are relevant to a wide range of NLP applications.

- **Query expansion**: search for *bike*, match *bicycle*
- **Semi-supervised learning**: use large unlabeled datasets to acquire features which are useful in supervised learning
- **Lexicon and thesaurus induction**: automatically expand hand-crafted lexical resources, or induce them from raw text

Here are some of the practical questions that we encounter when working with vector space representations of distributional semantics:

- What kinds of context should we consider? (see slides)
- How do measure similarity?
- How do we properly weigh frequent versus infrequent events?

3 Local context

The Brown et al (1992) clustering algorithm is over 20 years old and is still widely used in NLP!

- Context is just the immediately adjacent words.
- A generative probability model:
  - Assume each word w_i has a class c_i
  - Assume a generative model $\log P(w) = \sum_i \log P(w_i|c_i) + \log P(c_i|c_{i-1})$ (What does this remind you of?)
- Hierarchical clustering algorithm:
  - Start with every word in its own cluster
  - Until tired,
    * Choose two clusters c_i and c_j such that merging them will give the maximum improvement in $\log P(w)$
    * Equivalently, merge the clusters with the greatest mutual information.
  - The merge path of a word describes its semantics.

3.1 Model specifics

- $\mathcal{V}$ is the set of all words
- N number of observed word tokens
- $n(w)$ is the number of times we see word $w \in \mathcal{V}$
- $n(w, v)$ is the number of times w precedes v
- Let $C \to \{1, 2, \ldots, k\}$ define a partition of words into k classes

$$\begin{aligned} P(w_1, w_2, \ldots, w_T; C) &= \prod_i P(w_i|C(w_i))P(C(w_i)|C(w_{i-1})) \\ \log P(w_1, w_2, \ldots, w_T; C) &= \sum_i \log P(w_i|C(w_i))P(C(w_i)|C(w_{i-1})) \end{aligned}$$

This is kind of like an HMM, but each word can only be produced by a single cluster.

Let's define the "quality" of a clustering as the average log-likelihood:

$$\begin{aligned} J(C) =& \frac{1}{N} \sum_i^N \log P(w_i|C(w_i))P(C(w_i)|C(w_{i-1})) \\ =& \sum_{w,w'} \frac{n(w,w')}{N} \log P(w'|C(w'))P(C(w')|C(w')) && \text{sum over word types instead} \\ =& \sum_{w,w'} \frac{n(w,w')}{N} \log \frac{n(w')}{n(C(w'))} \frac{n(C(w),C(w'))}{n(C(w))} && \text{definition of probabilities} \\ =& \sum_{w,w'} \frac{n(w,w')}{N} \log \frac{n(w')}{1} \frac{n(C(w),C(w'))}{n(C(w))n(C(w'))} \frac{N}{N} && \text{re-arrange terms, multiply by one} \\ =& \sum_{w,w'} \frac{n(w,w')}{N} \log \frac{n(C(w),C(w')) \times N}{n(C(w))n(C(w'))} + \frac{n(w,w')}{N} \log \frac{n(w')}{N} && \text{distributive law} \\ =& \sum_{c,c'} \frac{n(c,c')}{N} \log \frac{n(c,c') \times N}{n(c)n(c')} + \sum_{w'} \frac{n(w')}{N} \log \frac{n(w')}{N} && \text{sum across classes} \\ =& \sum_{c,c'} P(c,c') \log \frac{P(c,c')}{P(c)P(c')} + \sum_{w'} P(w') \log P(w') && \text{multiply left side by } \frac{N^{-2}}{N^{-2}} \text{ inside log} \\ =& I(C) - H && \text{def. of mutual information and entropy} \end{aligned}$$

So the average log-likelihood is proportional to the mutual information of the clustering. Choosing a clustering with mutual information will maximize the log-likelihood. Now let's see how to do that efficiently.

3.2 $V \log V$ approximate algorithm

- Take m most frequent words, put each in its own cluster $c_1, c_2, \ldots, c_m$.
- For $i = (m+1) : |\mathcal{V}|$
  - Create a new cluster for the c_{m+1} for word i (ordered by frequency).
  - Choose two clusters c and c' to merge, minimizing the decrease in $I(C)$. This requires $\mathcal{O}(m^2)$ operations.
- Carry out $(m-1)$ final merges, to build full hierarchy

Cost: $\mathcal{O}(|\mathcal{V}|m^2 + n)$, plus time to sort words, $\mathcal{O}(|\mathcal{V}| \log |\mathcal{V}|)$.

4 Syntactic context

Local context is contingent on syntactic decisions that may have little to do with semantics:

- I gave Tim the ball.
- I gave the ball to Tim.

Using the syntactic structure of the sentence might give us a more meaningful context, yielding better clusters.

- Pereira et al (1993) cluster nouns based on the verbs for which they are the direct object.
  - The context vector for each noun is **the count of occurences as a direct object of each verb**.
  - As with Brown clustering, a class-based probability model:

$$\begin{aligned}\hat{p}(n, v) &= \sum_{c \in \mathcal{C}} p(c, n) p(v|c) \\ &= \sum_{c \in \mathcal{C}} p(c) p(n|c) p(v|c)\end{aligned}$$

    where n is the noun, v is the verb, and c is the class

- Objective: find the maximum likelihood cluster centroids.

- Dekang Lin (1997) extends this to all words, using incoming dependency edges (see slide)
  - For any pair of words i and j and relation r, we can compute:

$$P(i, j|r) = \frac{c(i, j, r)}{\sum_{i', j'} c(i', j', r)}, \qquad P(i|r) = \sum_j P(i, j|r)$$

  - Let $T(i)$ be the set of pairs $\langle j, r \rangle$ such that $P(i, j|r) > P(i|r)P(j|r)$
    * $T(i)$ contains words j that are especially likely to be joined with word i in relation r.
    * Note the connection to pointwise mutual information.
  - Similarity between u and v is defined through $T(u)$ and $T(v)$.
    * Lin considers several similarity measures for $T(u)$ and $T(v)$.
    * Many of these are used widely, and are worth knowing:
      · Cosine similarity: $\frac{|T(u) \cap T(v)|}{\sqrt{|T(u)||T(v)|}}$
      · Dice similarity: $\frac{2 \times |T(u) \cap T(v)|}{|T(u)| + |T(v)|}$
      · Jaccard similarity: $\frac{|T(u) \cap T(v)|}{|T(u)| + |T(v)| - |T(u) \cap T(v)|}$
    * Lin's metric is more complex:

$$\frac{\sum_{\langle r, w \rangle \in T(u) \cup T(v)} I(u, r, w) + I(v, r, w)}{\sum_{\langle r, w \rangle \in T(u)} I(u, r, w) + \sum_{\langle r, w \rangle \in T(v)} I(v, r, w)}$$

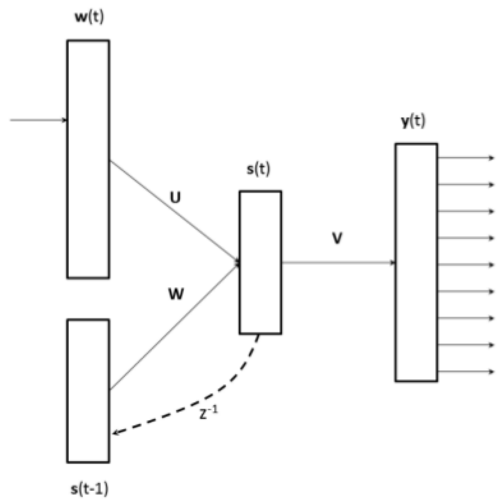
      where $I(u, r, w)$ is the mutual information between u and w, conditioned on r.
  - See slides for results.

5 Document context

See slides.

6 Neural word embeddings

This is currently a very hot area in NLP: use discriminative methods to learn **dense vector embeddings** for each word.



$$
\begin{aligned}
\boldsymbol{s}(t) =& f(\mathbf{U}\boldsymbol{w}(t) + \mathbf{W}\boldsymbol{s}(t-1)) && (1)\\
\boldsymbol{y}(t) =& g(\mathbf{V}\boldsymbol{s}(t)) && (2)\\
f(z) =& \text{Logistic}(z) = \frac{1}{1+e^{-z}} && (3)\\
g(z_m) =& \text{Soft-max}(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}} && (4)\\
& && (5)
\end{aligned}
$$

- $\boldsymbol{w}(t)$ is a one-hot (indicator) vector for the word at token t
- $\boldsymbol{s}(t)$ is a dense latent vector for the current history
- $\mathbf{U}$ projects from words into latent space. The rows of $\mathbf{U}$ are word embeddings.
- $\mathbf{W}$ projects from latent space at time t to $t+1$
- $\mathbf{V}$ projects from the latent space to predict the next word

The model is trained via back-propagation to optimize the log-likelihood of $\boldsymbol{w}$, with respect to the parameters $\mathbf{U}$, $\mathbf{V}$, $\mathbf{W}$, and $\boldsymbol{s}$. They don't say much about how this training works.

Cosine similarity in the resulting embeddings does remarkably well on analogy tasks. Given the analogy $a : b$ as $c : d$, they compute

$$\hat{d} = \arg\max_{d} \cos(u_a - u_b + u_c, u_d) \tag{6}$$

See slides for more details.