Lecture 24: Vector Semantics

Ling 1330/2330 Intro to Computational Linguistics
Na-Rae Han, 11/28/2023

Finally, **meaning**

Computational semantics: key areas

- Formal semantics: Logic, model-theoretic semantics
  - NLTK Book ch.10 Analyzing the meaning of sentences
- Word sense: lexical semantics
  - J&M Ch.23: Word senses and WordNet
  - NLTK Book 2.5 WordNet
- Word sense: vector semantics
  - J&M Ch.6: Vector semantics and embeddings
- Predicate-argument semantics, semantic roles
  - J&M Ch.24: Semantic role labeling
  - NLTK how to, PropBank

Vast landscape, so little time…

Vector semantics

- J&M Ch.6: Vector semantics and embeddings
  - This topic is very dense and gets technical – READ this chapter!

Today's slides borrow heavily from J & M's:
https://web.stanford.edu/~jurafsky/slp3/

Relation: Similarity

Words with similar meanings. Not synonyms per se, but sharing *some* element of meaning

```
car, bicycle
cow, horse
```

Ask humans how similar 2 words are

word1	word2	similarity
vanish	disappear	9.8
behave	obey	7.3
belief	impression	5.95
muscle	bone	3.65
modest	flexible	0.98
hole	agreement	0.3

SimLex-999 dataset (Hill et al., 2015)

But how to capture word similarity through *corpus data*?

What does *ongchoi* mean?

Suppose you see these sentences:

- *Ongchoi is delicious **sautéed with garlic**.*
- *Ongchoi is superb **over rice***
- *Ongchoi **leaves** with salty sauces*

- And you've also seen these:
  - *…spinach **sautéed with garlic over rice***
  - *Chard stems and **leaves** are **delicious***
  - *Collard greens and other **salty** leafy greens*
- Conclusion:
  - *Ongchoi* is a leafy green like spinach, chard, or collard greens

Ongchoi: *Ipomoea aquatica*
"Water Spinach"

Yamaguchi, Wikimedia Commons, public domain

Radically rethinking word meaning

- "If A and B have almost identical environments we say that they are synonyms."
  - Zellig Harris (1954)
- "The meaning of a word is its use in the language"
  - Ludwig Wittgenstein, Philosophical Investigations (1945)
- "You shall know a word by the company it keeps"
  - John Firth (1957)

- These form the philosophical foundation of **distributional semantics**.
  - Words are defined by their environments (the words around them)

We'll build a new model of meaning focusing on similarity

- Each word = a vector
  - Not just `chair.n.01`, `Like(x,y)` or `agree.01`
- Similar words are "nearby in space":

vector: a list of numbers with a particular length n

When arranged in two dimensions (vector length of 2)

We define a word as a vector

- Called an "embedding" because it's embedded into a multi-dimensional space (dimension # is vector length)
- Has quickly become the de-facto standard way to represent meaning in NLP
- Fine-grained model of meaning for similarity
  - NLP tasks like sentiment analysis
    - With words, requires **same** word type to be in training and test
    - With embeddings: ok if **similar** words occurred!!!
  - Question answering, conversational agents, etc

Two kinds of embeddings

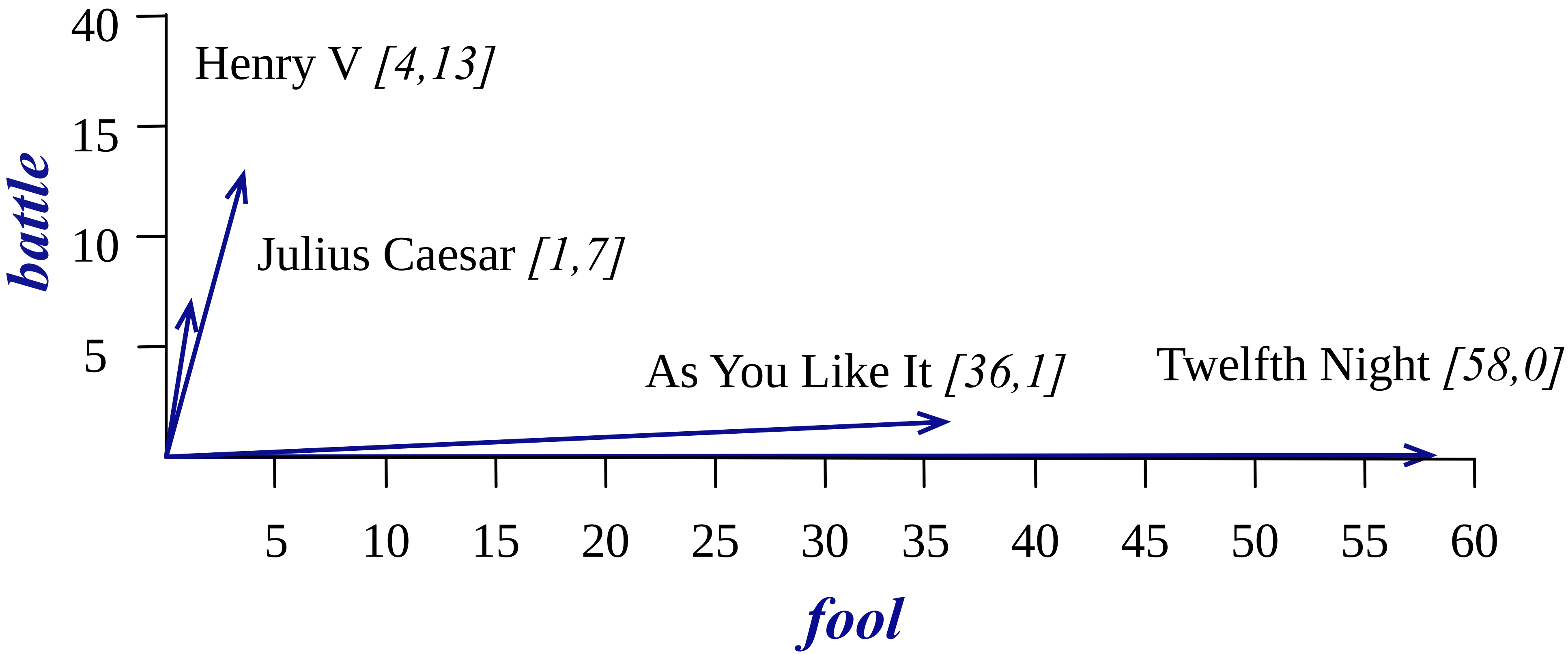
- Tf-idf
  - "Term frequency – inverse document frequency"
  - A common baseline model, long been popular in information retrieval (Karen Spärck Jones, 1972)
  - **Sparse** vectors
  - Words are represented by a simple function of the counts of nearby (= in the same document) words
- Word2vec
  - **Dense** vectors
  - Representation is created by training a classifier to distinguish nearby and far-away words

Term-document matrix

Each document is represented by a vector of word counts:

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Visualizing document vectors



Vectors are the basis of information retrieval

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

comedy

history

- Vectors are similar for the two comedies, different than the history
- Comedies have more *fools* and *wit* and fewer *battles*.

Flipping: words can be vectors too!

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

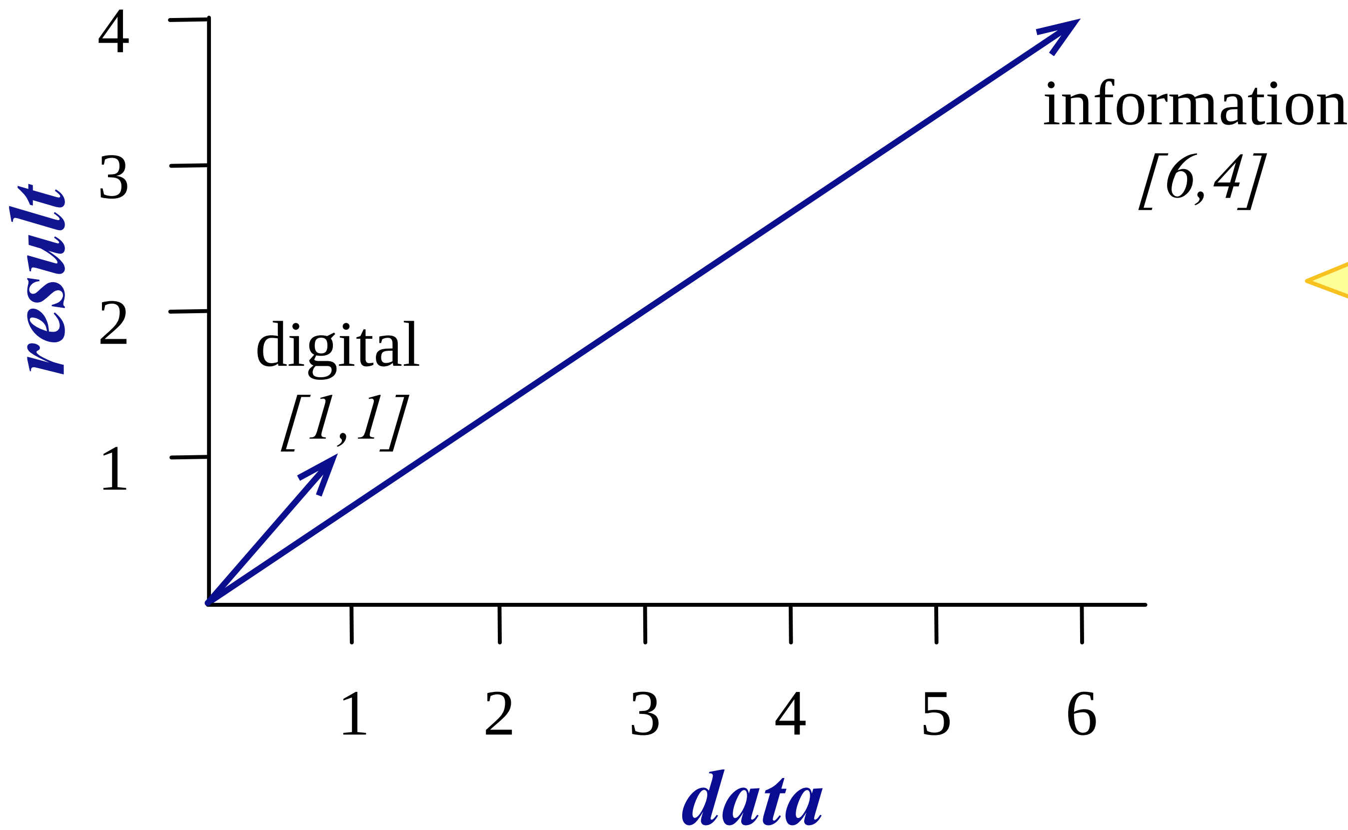
- *battle* is "the kind of word that occurs in Julius Caesar and Henry V"
- *fool* is "the kind of word that occurs in comedies, especially Twelfth Night"

More common: word-word matrix (or "term-context matrix")

- Two **words** are similar in meaning if their **context vectors** are similar

sugar, a sliced lemon, a tablespoonful of	**apricot**	jam, a pinch each of,
their enjoyment. Cautiously she sampled her first	**pineapple**	and another fruit whose taste she likened
well suited to programming on the digital	**computer**.	In finding the optimal R-stage policy from
for the purpose of gathering data and	**information**	necessary for the study authorized in the

	aardvark	digital	data	pinch	result	sugar	…
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
computer	0	2	1	0	1	0	
information	0	1	6	0	4	0	



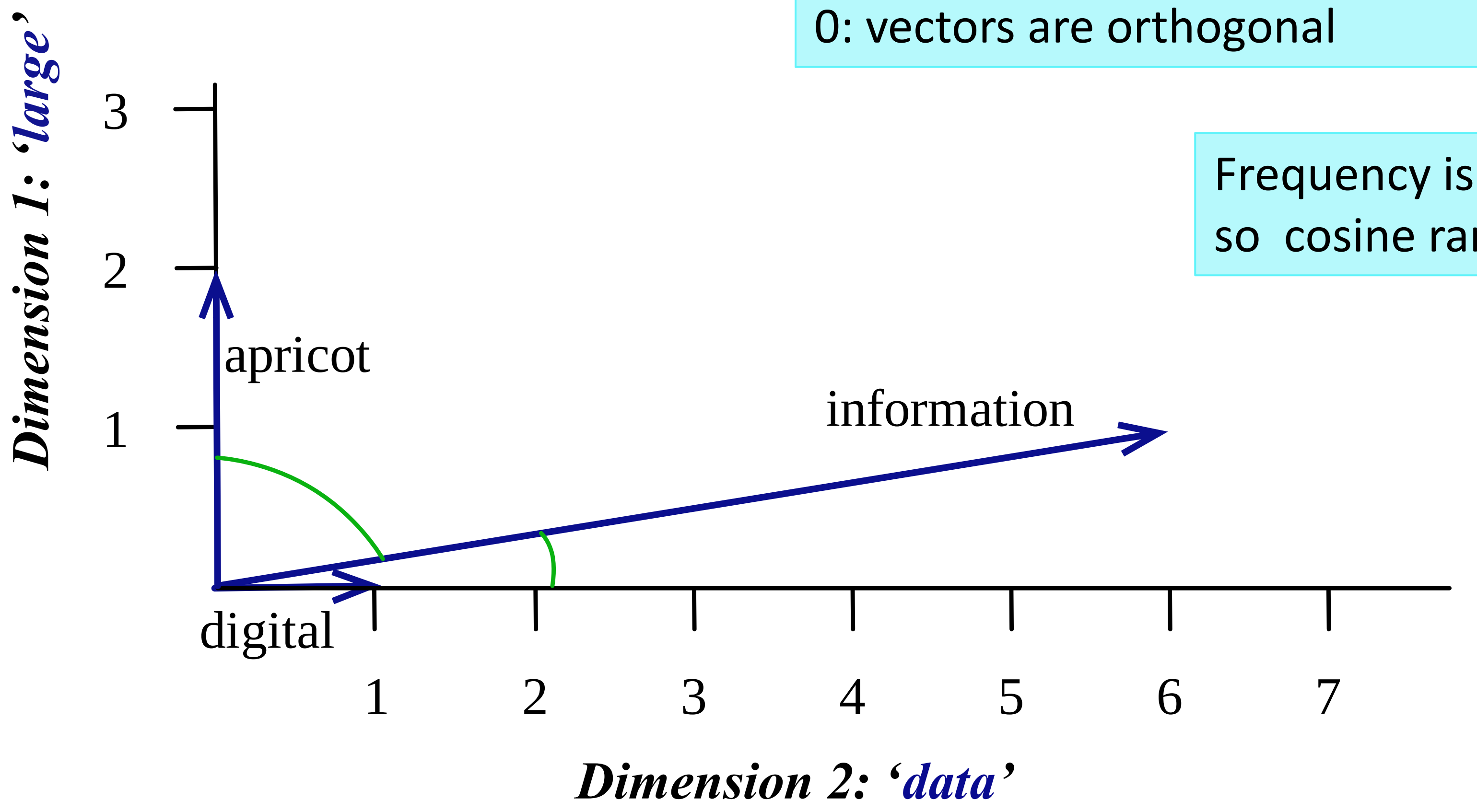
But how to measure similarity between "digital" and "information"?

cosine similarity

Cosine as a similarity metric

-1: vectors point in opposite directions
+1: vectors point in same directions
0: vectors are orthogonal

Frequency is non-negative,
so cosine range 0-1 here



But raw frequency is a bad representation

- Frequency is clearly useful; if *sugar* appears a lot near *apricot*, that's useful information.
- But overly frequent words like *the*, *it,* or *they* are not very informative about the context
- Need a function that resolves this frequency paradox!

🡐 Term frequency – **inverse document frequency**

tf-idf: combine two factors

- **tf: term frequency**. frequency count (usually log-transformed):

$$\mathrm{tf}_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{count}(t,d) & \text{if } \mathrm{count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$$

- **Idf: inverse document frequency**

Total # of docs in collection

$$\mathrm{idf}_i = \log\left(\frac{N}{\mathrm{df}_i}\right)$$

# of docs that have word i

Words like "the" or "good" have very low idf,
Words like "linguistics" have higher idf

tf-idf value for word t in document d: $w_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t$

Tf-idf demo

- Demo via Jupyter Notebook

Tf-idf representation is sparse

- tf-idf vectors are
  - **long** (length |V|= 20,000 to 50,000)
  - **sparse** (most elements are zero)
  - dimensions are tied to specifics of data

Depends on # of word types in document set

- Alternative: dense vectors

  Vectors which are:
  - **short** (typically 50 – 1000 dimensions)
  - **dense** (most elements are non-zero)
  - Dimension *d* size can be arbitrary, doesn't have a clear interpretation

Sparse vs. dense vectors

- Why dense vectors?
  - Short vectors may be easier to use as **features** in machine learning (less weights to tune)
  - Dense vectors may **generalize** better than storing explicit counts
  - They may do better at capturing synonymy:
    - *car* and *automobile* are synonyms; but are distinct dimensions → a word with *car* as a neighbor and a word with *automobile* as a neighbor should be similar, but aren't
  - **In practice, they work better**

Dense embeddings you can download!

- **Word2vec** (Mikolov et al.)
  - https://code.google.com/archive/p/word2vec/
- **Fasttext** http://www.fasttext.cc/

- **GloVe** (Pennington, Socher, Manning)
  - https://nlp.stanford.edu/projects/glove/

Word2Vec

- Introduced by Mikolov (2013)
- Popular embedding method
- Very fast to train
- Code available on the web
- Idea: **predict** rather than **count**

Word2Vec: a rough sketch

- Instead of **counting** how often each word w occurs near "apricot", **train a classifier** on a binary prediction task:
  - Is w likely to show up near "apricot"?
- We don't actually care about this task itself, but we'll take the learned classifier **weights** as the word embeddings
- Brilliant insight: Use running text as implicitly supervised training data!
  - A word s near apricot
  - Acts as gold 'correct answer' to the question "Is word w likely to show up near *apricot*?"
- No need for hand-labeled supervision!
- Idea comes from **neural language modeling**

Skip-gram algorithm

1. Treat the target word and a neighboring context word as positive examples.
2. Randomly sample other words in the lexicon to get negative samples
3. Use logistic regression to train a classifier to distinguish those two cases
4. Use the weights as the embeddings

skip-gram with negative sampling (SGNS)
← one of multiple tasks provided by Word2Vec

Skip-Gram Training

Training sentence:

- ... lemon, a tablespoon of **apricot** jam a pinch ...
- c1 c2 t c3 c4

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	preserves
apricot	or

negative examples -

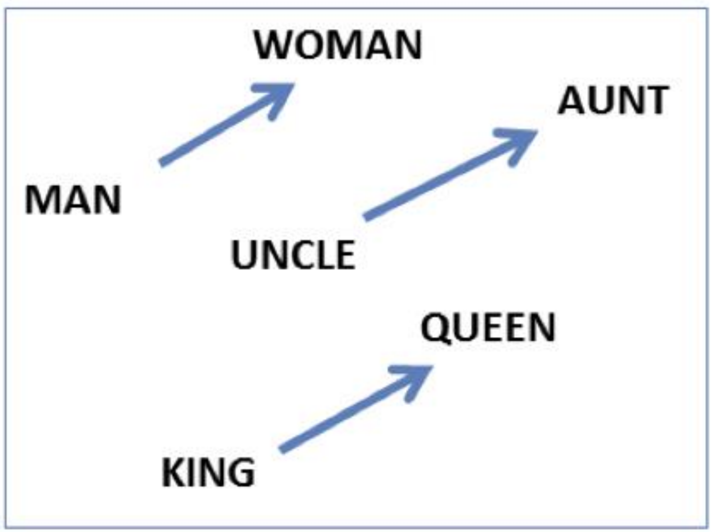
t	c	t	c
apricot	aardvark	apricot	twelve
apricot	puddle	apricot	hello
apricot	where	apricot	dear
apricot	coaxial	apricot	forever

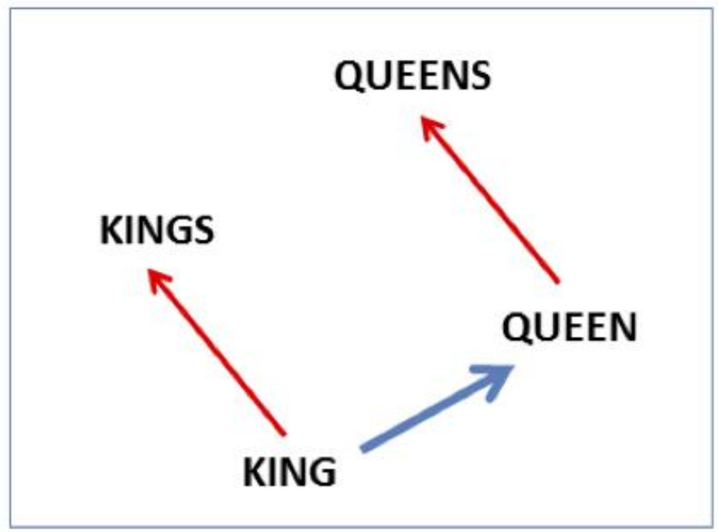
We will skip the details.
Refer to the book chapter!

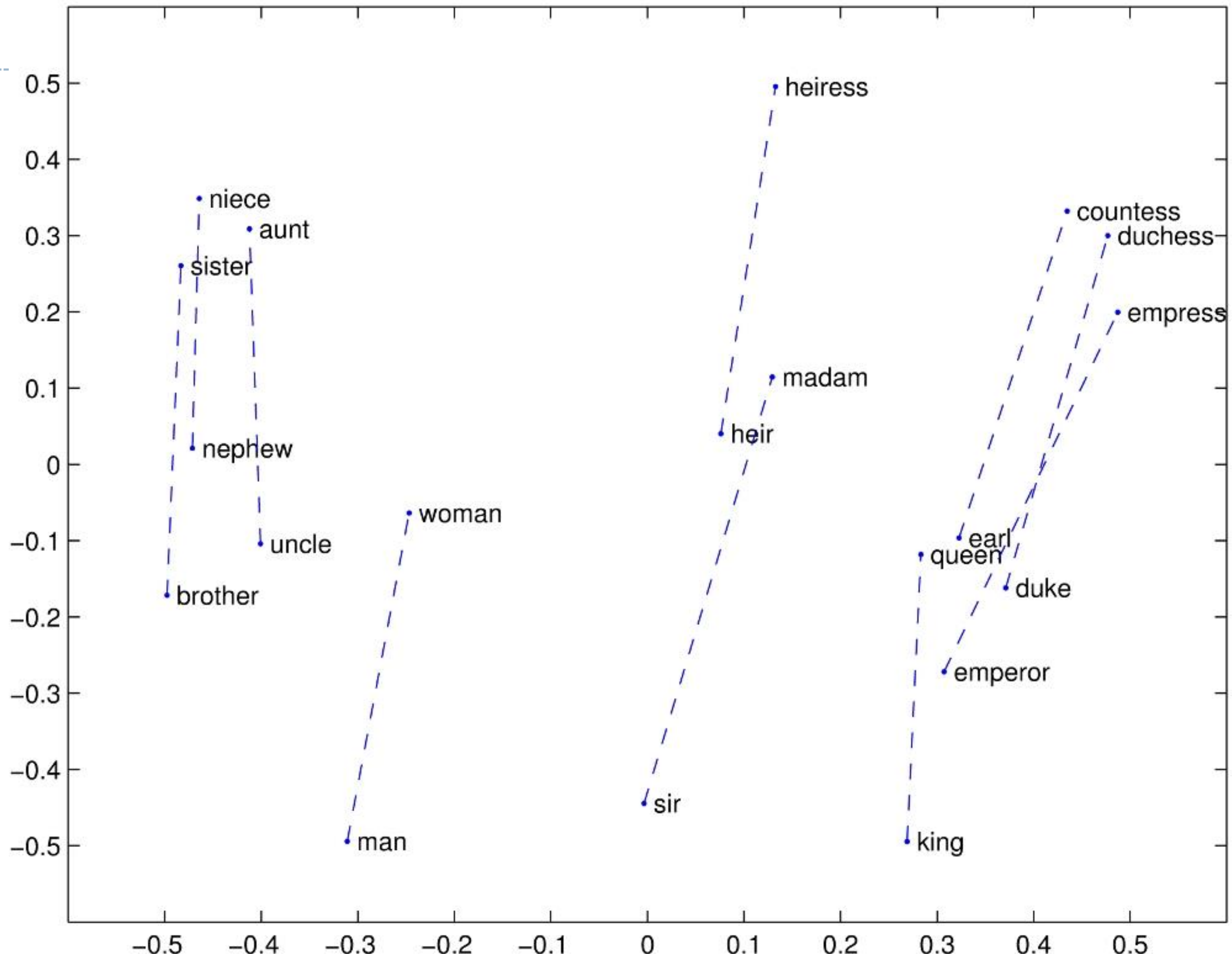
Analogy: Embeddings capture relational meaning!

vector(*'Paris'*) - vector(*'France'*) + vector(*'Italy'*)
≈ vector('Rome')

vector(*'king'*) - vector(*'man'*) + vector(*'woman'*)
≈ vector('queen')







0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.5
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5
niece
aunt
sister
nephew
uncle
brother
woman
man
heiress
madam
heir
sir
countess
duchess
empress
earl
queen
duke
emperor
king

Word2Vec demo

- Demo via Jupyter Notebook

Wrapping up

- Next class:
  - Deep Learning Language Models: guest lecture by Tianyi
  - Machine Translation
- Homework 9 due on Thu
  - genism installation: do it TODAY
  - Make sure to refresh the page! Recent changes in PART 2.
- Final exam ➔

Final exam

- 12/13 (Wed), 4—5:50pm
- At G17 CL (Language Media Center)

- 150 total points (50% larger than midterm)
- All pen-and-pencil based.
- **1 cheat sheet allowed**:
  - letter-sized, front-and-back, hand-written.
- Cumulative! 10-20% will be from first half of the semester.
- Make sure to study book chapters and other linked materials. Post-midterm, my slides are not as "comprehensive".