Word Meaning
What do words mean?

N-gram or text classification methods we've seen so far

- Words are just strings (or indices $w_i$ in a vocabulary list)
- That's not very satisfactory!

Introductory logic classes:

- The meaning of "dog" is DOG; cat is CAT
  \[ \forall x \text{ DOG}(x) \rightarrow \text{MAMMAL}(x) \]

Old linguistics joke by Barbara Partee in 1967:

- Q: What's the meaning of life?
- A: LIFE

That seems hardly better!
Desiderata

What should a theory of word meaning do for us?
Let's look at some desiderata

From *lexical semantics*, the linguistic study of word meaning
Lemmas and senses

**lemma**

**mouse (N)**

1. any of numerous small rodents...
2. a hand-operated device that controls a cursor...

**sense**

A *sense* or “*concept*” is the meaning component of a word. Lemmas can be *polysemous* (have multiple senses).

Modified from the online thesaurus WordNet
Relations between senses: Synonymy

Synonyms have the same meaning in some or all contexts.

- filbert / hazelnut
- couch / sofa
- big / large
- automobile / car
- vomit / throw up
- water / H₂O
Relations between senses: Synonymy

Note that there are probably no examples of perfect synonymy.

◦ Even if many aspects of meaning are identical
◦ Still may differ based on politeness, slang, register, genre, etc.
Relation: Synonymy?

water/H₂O
"H₂O" in a surfing guide?
big/large
my big sister != my large sister
The Linguistic Principle of Contrast

Difference in form $\rightarrow$ difference in meaning
Re: "exact" synonyms

"je ne crois pas qu’il y ait de mot synonime dans aucune Langue."

[I do not believe that there is a synonymous word in any language]
Relation: **Similarity**

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

- car, bicycle
- cow, horse
Ask humans how similar 2 words are

<table>
<thead>
<tr>
<th>word1</th>
<th>word2</th>
<th>similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>vanish</td>
<td>disappear</td>
<td>9.8</td>
</tr>
<tr>
<td>behave</td>
<td>obey</td>
<td>7.3</td>
</tr>
<tr>
<td>belief</td>
<td>impression</td>
<td>5.95</td>
</tr>
<tr>
<td>muscle</td>
<td>bone</td>
<td>3.65</td>
</tr>
<tr>
<td>modest</td>
<td>flexible</td>
<td>0.98</td>
</tr>
<tr>
<td>hole</td>
<td>agreement</td>
<td>0.3</td>
</tr>
</tbody>
</table>

SimLex-999 dataset (Hill et al., 2015)
Relation: Word relatedness

Also called "word association"

Words can be related in any way, perhaps via a semantic frame or field

- coffee, tea: similar
- coffee, cup: related, not similar
Semantic field

Words that
  ◦ cover a particular semantic domain
  ◦ bear structured relations with each other.

hospitals
  surgeon, scalpel, nurse, anaesthetic, hospital

restaurants
  waiter, menu, plate, food, menu, chef

houses
  door, roof, kitchen, family, bed
Relation: Antonymy

Senses that are opposites with respect to only one feature of meaning

Otherwise, they are very similar!

dark/light  short/long  fast/slow  rise/fall
hot/cold  up/down  in/out

More formally: antonyms can

◦ define a binary opposition or be at opposite ends of a scale
  ◦ long/short, fast/slow
  ◦ Be *reversives*:
  ◦ rise/fall, up/down
Connotation (sentiment)

• Words have **affective** meanings
  • Positive connotations (*happy*)
  • Negative connotations (*sad*)

• Connotations can be subtle:
  • Positive connotation: *copy, replica, reproduction*
  • Negative connotation: *fake, knockoff, forgery*

• Evaluation (sentiment!)
  • Positive evaluation (*great, love*)
  • Negative evaluation (*terrible, hate*)
Connotation

Osgood et al. (1957)

Words seem to vary along 3 affective dimensions:
- **valence**: the pleasantness of the stimulus
- **arousal**: the intensity of emotion provoked by the stimulus
- **dominance**: the degree of control exerted by the stimulus

### Table

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Word</th>
<th>Score</th>
<th>Word</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valence</td>
<td>love</td>
<td>1.000</td>
<td>toxic</td>
<td>0.008</td>
</tr>
<tr>
<td></td>
<td>happy</td>
<td>1.000</td>
<td>nightmare</td>
<td>0.005</td>
</tr>
<tr>
<td>Arousal</td>
<td>elated</td>
<td>0.960</td>
<td>mellow</td>
<td>0.069</td>
</tr>
<tr>
<td></td>
<td>frenzy</td>
<td>0.965</td>
<td>napping</td>
<td>0.046</td>
</tr>
<tr>
<td>Dominance</td>
<td>powerful</td>
<td>0.991</td>
<td>weak</td>
<td>0.045</td>
</tr>
<tr>
<td></td>
<td>leadership</td>
<td>0.983</td>
<td>empty</td>
<td>0.081</td>
</tr>
</tbody>
</table>

Values from NRC VAD Lexicon (Mohammad 2018)
So far

**Concepts** or word senses
- Have a complex many-to-many association with *words* (homonymy, multiple senses)

Have relations with each other
- Synonymy
- Antonymy
- Similarity
- Relatedness
- Connotation
Word Meaning
Vector Semantics
Computational models of word meaning

Can we build a theory of how to represent word meaning, that accounts for at least some of the desiderata?

We'll introduce vector semantics

  The standard model in language processing!

Handles many of our goals!
Ludwig Wittgenstein

PI #43:
"The meaning of a word is its use in the language"
Let's define words by their usages

One way to define "usage":
words are defined by their environments (the words around them)

Zellig Harris (1954):
If A and B have almost identical environments we say that they are synonyms.
What does recent English borrowing *ongchoi* mean?

Suppose you see these sentences:

- Ong choi is delicious *sautéed with garlic*.
- Ong choi is superb *over rice*
- Ong choi *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
- We could conclude this based on words like "leaves" and "delicious" and "sauteed"
Ongchoi: *Ipomoea aquatica* "Water Spinach"

空心菜
*kangkong*
rau muống
...

Yamaguchi, Wikimedia Commons, public domain
Idea 1: Defining meaning by linguistic distribution

Let's define the meaning of a word by its distribution in language use, meaning its neighboring words or grammatical environments.
Idea 2: Meaning as a point in space (Osgood et al. 1957)

3 affective dimensions for a word

- **valence**: pleasantness
- **arousal**: intensity of emotion
- **dominance**: the degree of control exerted

<table>
<thead>
<tr>
<th></th>
<th>Word</th>
<th>Score</th>
<th></th>
<th>Word</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Valence</strong></td>
<td>love</td>
<td>1.000</td>
<td>toxic</td>
<td>0.008</td>
<td></td>
</tr>
<tr>
<td></td>
<td>happy</td>
<td>1.000</td>
<td>nightmare</td>
<td>0.005</td>
<td></td>
</tr>
<tr>
<td><strong>Arousal</strong></td>
<td>elated</td>
<td>0.960</td>
<td>mellow</td>
<td>0.069</td>
<td></td>
</tr>
<tr>
<td></td>
<td>frenzy</td>
<td>0.965</td>
<td>napping</td>
<td>0.046</td>
<td></td>
</tr>
<tr>
<td><strong>Dominance</strong></td>
<td>powerful</td>
<td>0.991</td>
<td>weak</td>
<td>0.045</td>
<td></td>
</tr>
<tr>
<td></td>
<td>leadership</td>
<td>0.983</td>
<td>empty</td>
<td>0.081</td>
<td></td>
</tr>
</tbody>
</table>

Hence the connotation of a word is a vector in 3-space.
Idea 1: Defining meaning by linguistic distribution

Idea 2: Meaning as a point in multidimensional space
Defining meaning as a point in space based on distribution
Each word = a vector (not just "good" or "w_{45}")
Similar words are "nearby in semantic space"
We build this space automatically by seeing which words are nearby in text
We define meaning of a word as a vector

Called an "embedding" because it's embedded into a space (see textbook)

The standard way to represent meaning in NLP

Every modern NLP algorithm uses embeddings as the representation of word meaning

Fine-grained model of meaning for similarity
Intuition: why vectors?

Consider sentiment analysis:

- **With words,** a feature is a word identity
  - Feature 5: 'The previous word was "terrible"'
  - requires **exact same word** to be in training and test

- **With embeddings:**
  - Feature is a word vector
  - 'The previous word was vector [35,22,17...]
  - Now in the test set we might see a similar vector [34,21,14]
  - We can generalize to **similar but unseen** words!!!
We'll discuss 2 kinds of embeddings

**tf-idf**
- Information Retrieval workhorse!
- A common baseline model
- **Sparse** vectors
- Words are represented by (a simple function of) the counts of nearby words

**Word2vec**
- **Dense** vectors
- Representation is created by training a classifier to predict whether a word is likely to appear nearby
- Later we'll discuss extensions called **contextual embeddings**
From now on: Computing with meaning representations instead of string representations

Nets are for fish; Once you get the fish, you can forget the net.

Words are for meaning; Once you get the meaning, you can forget the words

庄子(Zhuangzi), Chapter 26
Vector Semantics
Words and Vectors
Term-document matrix

Each document is represented by a vector of words

<table>
<thead>
<tr>
<th></th>
<th>As You Like It</th>
<th>Twelfth Night</th>
<th>Julius Caesar</th>
<th>Henry V</th>
</tr>
</thead>
<tbody>
<tr>
<td>battle</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>13</td>
</tr>
<tr>
<td>good</td>
<td>14</td>
<td>80</td>
<td>62</td>
<td>89</td>
</tr>
<tr>
<td>fool</td>
<td>36</td>
<td>58</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>wit</td>
<td>20</td>
<td>15</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Visualizing document vectors

The term-document matrix for four words in four Shakespeare plays. Each cell contains the number of times the (row) word occurs in the (column) document.

represented as a count vector, a column in Fig. 6.3.

To review some basic linear algebra, a vector is, at heart, just a list or array of numbers. So *As You Like It* is represented as the list [1, 114, 36, 20] (the first column vector in Fig. 6.3) and *Julius Caesar* is represented as the list [7, 62, 1, 2] (the third column vector). A vector space is a collection of vectors, characterized by their dimension. In the example in Fig. 6.3, the document vectors are of dimension 4, just so they fit on the page; in real term-document matrices, the vectors representing each document would have dimensionality $|V|$, the vocabulary size.

The ordering of the numbers in a vector space indicates different meaningful dimensions on which documents vary. Thus the first dimension for both these vectors corresponds to the number of times the word *battle* occurs, and we can compare each dimension, noting for example that the vectors for *As You Like It* and *Twelfth Night* have similar values (1 and 0, respectively) for the first dimension.

As You Like It Twelfth Night Julius Caesar Henry V

Figure 6.4 A spatial visualization of the document vectors for the four Shakespeare play documents, showing just two of the dimensions, corresponding to the words *battle* and *fool*.

The comedies have high values for the *fool* dimension and low values for the *battle* dimension.

Term-document matrices were originally defined as a means of finding similar documents for the task of document information retrieval. Two documents that are
Vectors are the basis of information retrieval

<table>
<thead>
<tr>
<th></th>
<th>As You Like It</th>
<th>Twelfth Night</th>
<th>Julius Caesar</th>
<th>Henry V</th>
</tr>
</thead>
<tbody>
<tr>
<td>battle</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>13</td>
</tr>
<tr>
<td>good</td>
<td>114</td>
<td>80</td>
<td>62</td>
<td>89</td>
</tr>
<tr>
<td>fool</td>
<td>36</td>
<td>58</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>wit</td>
<td>20</td>
<td>15</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Vectors are similar for the two comedies

But comedies are different than the other two

Comedies have more *fools* and *wit* and fewer *battles*. 
Idea for word meaning: Words can be vectors too!!!

<table>
<thead>
<tr>
<th></th>
<th>As You Like It</th>
<th>Twelfth Night</th>
<th>Julius Caesar</th>
<th>Henry V</th>
</tr>
</thead>
<tbody>
<tr>
<td>battle</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>13</td>
</tr>
<tr>
<td>good</td>
<td>114</td>
<td>80</td>
<td>62</td>
<td>89</td>
</tr>
<tr>
<td>fool</td>
<td>36</td>
<td>58</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>wit</td>
<td>20</td>
<td>15</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

*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

<table>
<thead>
<tr>
<th>aardvark</th>
<th>...</th>
<th>computer</th>
<th>data</th>
<th>result</th>
<th>pie</th>
<th>sugar</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>cherry</td>
<td>0</td>
<td>...</td>
<td>2</td>
<td>8</td>
<td>9</td>
<td>442</td>
<td>25</td>
</tr>
<tr>
<td>strawberry</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>60</td>
<td>19</td>
</tr>
<tr>
<td>digital</td>
<td>0</td>
<td>...</td>
<td>1670</td>
<td>1683</td>
<td>85</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>information</td>
<td>0</td>
<td>...</td>
<td>3325</td>
<td>3982</td>
<td>378</td>
<td>5</td>
<td>13</td>
</tr>
</tbody>
</table>
Figure 6.5 Co-occurrence vectors for four words in the Wikipedia corpus, showing six of the dimensions (hand-picked for pedagogical purposes). The vector for digital is outlined in red. Note that a real vector would have vastly more dimensions and thus be much sparser.

Figure 6.6 A spatial visualization of word vectors for digital and information, showing just two of the dimensions, corresponding to the words data and computer. Note that $|\mathbf{V}|$, the length of the vector, is generally the size of the vocabulary, usually between 10,000 and 50,000 words (using the most frequent words in the training corpus; keeping words after about the most frequent 50,000 or so is generally not helpful). But of course since most of these numbers are zero these are sparse vector representations, and there are efficient algorithms for storing and computing with sparse matrices.

Now that we have some intuitions, let's move on to examine the details of computing word similarity. Afterwards we'll discuss the tf-idf method of weighting cells.

6.4 Cosine for measuring similarity

To define similarity between two target words $v$ and $w$, we need a measure for taking two such vectors and giving a measure of vector similarity. By far the most common similarity metric is the cosine of the angle between the vectors. The cosine—like most measures for vector similarity used in NLP—is based on the dot product operator from linear algebra, also called the inner product:

$$\text{dot product} (\mathbf{v}, \mathbf{w}) = \mathbf{v} \cdot \mathbf{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_N w_N \quad (6.7)$$

As we will see, most metrics for similarity between vectors are based on the dot product. The dot product acts as a similarity metric because it will tend to be high just when the two vectors have large values in the same dimensions. Alternatively, vectors that have zeros in different dimensions—orthogonal vectors—will have a dot product of 0, representing their strong dissimilarity.
Words and Vectors
Cosine for computing word similarity
Computing word similarity: Dot product and cosine

The dot product between two vectors is a scalar:

\[ \text{dot product}(v, w) = v \cdot w = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_N w_N \]

The dot product tends to be high when the two vectors have large values in the same dimensions.

Dot product can thus be a useful similarity metric between vectors.
Problem with raw dot-product

Dot product favors long vectors
Dot product is higher if a vector is longer (has higher values in many dimension)

Vector length:

\[ |\mathbf{v}| = \sqrt{\sum_{i=1}^{N} v_i^2} \]

Frequent words (of, the, you) have long vectors (since they occur many times with other words).

So dot product overly favors frequent words
Alternative: cosine for computing word similarity

$$\text{cosine}(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}| |\vec{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2} \sqrt{\sum_{i=1}^{N} w_i^2}}$$

Based on the definition of the dot product between two vectors a and b

$$a \cdot b = |a||b| \cos \theta$$

$$\frac{a \cdot b}{|a||b|} = \cos \theta$$
Cosine as a similarity metric

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

But since raw frequency values are non-negative, the cosine for term-term matrix vectors ranges from 0–1
Cosine examples

\[
\cos(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{||\vec{v}|| \cdot ||\vec{w}||} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2} \cdot \sqrt{\sum_{i=1}^{N} w_i^2}}
\]

\[
\cos(\text{cherry, information}) = \frac{442 \cdot 5 + 8 \cdot 3982 + 2 \cdot 3325}{\sqrt{442^2 + 8^2 + 2^2} \sqrt{5^2 + 3982^2 + 3325^2}} = .017
\]

\[
\cos(\text{digital, information}) = \frac{5 \cdot 5 + 1683 \cdot 3982 + 1670 \cdot 3325}{\sqrt{5^2 + 1683^2 + 1670^2} \sqrt{5^2 + 3982^2 + 3325^2}} = .996
\]
Visualizing cosines (well, angles)

Dimension 1: ‘pie’

500

cherry

Dimension 2: ‘computer’

digital

information

Figure 6.7 A (rough) graphical demonstration of cosine similarity, showing vectors for three words (cherry, digital, and information) in the two dimensional space defined by counts of the words computer and pie nearby. Note that the angle between digital and information is smaller than the angle between cherry and information. When two vectors are more similar, the cosine is larger but the angle is smaller; the cosine has its maximum (1) when the angle between two vectors is smallest (0); the cosine of all other angles is less than 1.

6.5 TF-IDF: Weighing terms in the vector

The co-occurrence matrix in Fig. 6.5 represented each cell by the raw frequency of the co-occurrence of two words. It turns out, however, that simple frequency isn’t the best measure of association between words. One problem is that raw frequency is very skewed and not very discriminative. If we want to know what kinds of contexts are shared by cherry and strawberry but not by digital and information, we’re not going to get good discrimination from words like the, it, or they, which occur frequently with all sorts of words and aren’t informative about any particular word. We saw this also in Fig. 6.3 for the Shakespeare corpus; the dimension for the word good is not very discriminative between plays; good is simply a frequent word and has roughly equivalent high frequencies in each of the plays.

It’s a bit of a paradox. Words that occur nearby frequently (maybe pie nearby cherry) are more important than words that only appear once or twice. Yet words that are too frequent—ubiquitous, like the or good—are unimportant. How can we balance these two conflicting constraints?

The tf-idf algorithm (the '-' here is a hyphen, not a minus sign) is the product of two terms, each term capturing one of these two intuitions:

The first is the term frequency (Luhn, 1957): the frequency of the word $t$ in the document $d$. We can just use the raw count as the term frequency:

$$tf_{t,d} = \text{count}(t,d) \quad (6.11)$$

Alternatively we can squash the raw frequency a bit, by using the log of the frequency instead. The intuition is that a word appearing 100 times in a document doesn’t make that word 100 times more likely to be relevant to the meaning of the document. Because we can’t take the log of 0, we normally add 1 to the count:

$$tf_{t,d} = \log_{10}(\text{count}(t,d)+1) \quad (6.12)$$

Or we can use this alternative:

$$tf_{t,d} = \frac{1}{\log_{10}(\text{count}(t,d)+1)}$$

If we use log weighting, terms which occur 10 times in a document would have a $tf=2$, 100 times in a document $tf=3$, 1000 times $tf=4$, and so on.
Cosine for computing word similarity
Vector Semantics & Embeddings

TF-IDF
But raw frequency is a bad representation

• The co-occurrence matrices we have seen represent each cell by word frequencies.
• 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.
• It's a paradox! How can we balance these two conflicting constraints?
Two common solutions for word weighting

**tf-idf:** tf-idf value for word t in document d:

\[ w_{t,d} = tf_{t,d} \times idf_t \]

Words like "the" or "it" have very low idf

**PMI:** (Pointwise mutual information)

\[ \text{PMI}(w_1, w_2) = \log \frac{p(w_1, w_2)}{p(w_1)p(w_2)} \]

See if words like "good" appear more often with "great" than we would expect by chance
Term frequency (tf) in the tf-idf algorithm

We could imagine using raw count:

$$tf_{t,d} = \text{count}(t,d)$$

But instead of using raw count, we usually squash a bit:

$$tf_{t,d} = \begin{cases} 1 + \log_{10} \text{count}(t,d) & \text{if } \text{count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$$
Document frequency (df)

dfₜ \textit{is} the number of documents t occurs in.

(note this is not collection frequency: total count across all documents)

"Romeo" is very distinctive for one Shakespeare play:

<table>
<thead>
<tr>
<th>Collection Frequency</th>
<th>Document Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Romeo</td>
<td>113</td>
</tr>
<tr>
<td>action</td>
<td>113</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Romeo</td>
<td>1</td>
</tr>
<tr>
<td>action</td>
<td>31</td>
</tr>
</tbody>
</table>

The "tf-idf" algorithm, usually used when the dimensions are documents, balances these two conflicting constraints:

The first factor is the term frequency (Luhn, 1957) : the frequency of the word t in the document d. We can just use the raw count as the term frequency:

\[ tf(t, d) = \text{count}(t, d) \] (6.11)

More commonly we squash the raw frequency a bit, by using the log of the frequency instead. The intuition is that a word appearing 100 times in a document doesn't make that word 100 times more likely to be relevant to the meaning of the document. Because we can't take the log of 0, we normally add 1 to the count:

\[ tf(t, d) = \log_{10}(\text{count}(t, d) + 1) \] (6.12)

The second factor is used to give a higher weight to words that occur only in a few documents. Terms that are limited to a few documents are useful for discriminating those documents from the rest of the collection; terms that occur frequently across the entire collection aren't as helpful. The document frequency dfₜ of a term t is the number of documents it occurs in.

We emphasize discriminative words like Romeo via the inverse document frequency or idf term weight (Sparck Jones, 1972). The idf is defined using the fraction \( N / df_t \), where N is the total number of documents in the collection, and \( df_t \) is the number of documents in which term t occurs. The fewer documents in which a term occurs, the higher this weight. The lowest weight of 1 is assigned to terms that occur in all the documents.

It's usually clear what counts as a document: in Shakespeare we would use a play; when processing a collection of encyclopedia articles like Wikipedia, the document is a Wikipedia page; in processing newspaper articles, the document is a single article. Occasionally your corpus might not have appropriate document divisions and you might need to break up the corpus into documents yourself for the purposes of computing idf.
Inverse document frequency (idf)

\[ \text{idf}_t = \log_{10} \left( \frac{N}{\text{df}_t} \right) \]

N is the total number of documents in the collection

<table>
<thead>
<tr>
<th>Word</th>
<th>df</th>
<th>idf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Romeo</td>
<td>1</td>
<td>1.57</td>
</tr>
<tr>
<td>salad</td>
<td>2</td>
<td>1.27</td>
</tr>
<tr>
<td>Falstaff</td>
<td>4</td>
<td>0.967</td>
</tr>
<tr>
<td>forest</td>
<td>12</td>
<td>0.489</td>
</tr>
<tr>
<td>battle</td>
<td>21</td>
<td>0.246</td>
</tr>
<tr>
<td>wit</td>
<td>34</td>
<td>0.037</td>
</tr>
<tr>
<td>fool</td>
<td>36</td>
<td>0.012</td>
</tr>
<tr>
<td>good</td>
<td>37</td>
<td>0</td>
</tr>
<tr>
<td>sweet</td>
<td>37</td>
<td>0</td>
</tr>
</tbody>
</table>
What is a document?

Could be a play or a Wikipedia article

But for the purposes of tf-idf, documents can be anything; we often call each paragraph a document!
Final tf-idf weighted value for a word

$$w_{t,d} = tf_{t,d} \times idf_t$$

Raw counts:

<table>
<thead>
<tr>
<th></th>
<th>As You Like It</th>
<th>Twelfth Night</th>
<th>Julius Caesar</th>
<th>Henry V</th>
</tr>
</thead>
<tbody>
<tr>
<td>battle</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>13</td>
</tr>
<tr>
<td>good</td>
<td>114</td>
<td>80</td>
<td>62</td>
<td>89</td>
</tr>
<tr>
<td>fool</td>
<td>36</td>
<td>58</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>wit</td>
<td>20</td>
<td>15</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

tf-idf:

<table>
<thead>
<tr>
<th></th>
<th>As You Like It</th>
<th>Twelfth Night</th>
<th>Julius Caesar</th>
<th>Henry V</th>
</tr>
</thead>
<tbody>
<tr>
<td>battle</td>
<td>0.246</td>
<td>0</td>
<td>0.454</td>
<td>0.520</td>
</tr>
<tr>
<td>good</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>fool</td>
<td>0.030</td>
<td>0.033</td>
<td>0.0012</td>
<td>0.0019</td>
</tr>
<tr>
<td>wit</td>
<td>0.085</td>
<td>0.081</td>
<td>0.048</td>
<td>0.054</td>
</tr>
</tbody>
</table>
TF-IDF
Vector Semantics & Embeddings

Word2vec
Sparse versus dense vectors

tf-idf (or PMI) vectors are
- long (length $|V|$ = 20,000 to 50,000)
- sparse (most elements are zero)

Alternative: learn vectors which are
- short (length 50-1000)
- dense (most elements are non-zero)
Sparse versus dense vectors

Why dense vectors?

- Short vectors may be easier to use as features in machine learning (fewer weights to tune)
- Dense vectors may generalize better than explicit counts
- Dense vectors 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
Common methods for getting short dense vectors

“Neural Language Model”-inspired models
- Word2vec (skipgram, CBOW), GloVe

Singular Value Decomposition (SVD)
- A special case of this is called LSA – Latent Semantic Analysis

Alternative to these "static embeddings":
- Contextual Embeddings (ELMo, BERT)
- Compute distinct embeddings for a word in its context
- Separate embeddings for each token of a word
Simple static embeddings you can download!

Word2vec (Mikolov et al)
https://code.google.com/archive/p/word2vec/

GloVe (Pennington, Socher, Manning)
http://nlp.stanford.edu/projects/glove/
Word2vec

Popular embedding method
Very fast to train
Code available on the web

Idea: \textbf{predict} rather than \textbf{count}

Word2vec provides various options. We'll do: 
\textit{skip-gram with negative sampling (SGNS)}
Word2vec

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
- But we'll take the learned classifier weights as the word embeddings

Big idea: self-supervision:
- A word $c$ that occurs near apricot in the corpus cats as the gold "correct answer" for supervised learning
- No need for human labels
- Bengio et al. (2003); Collobert et al. (2011)
Approach: predict if candidate word \( c \) is a "neighbor"

1. Treat the target word \( t \) and a neighboring context word \( c \) as \textbf{positive examples}.
2. Randomly sample other words in the lexicon to get negative examples.
3. Use logistic regression to train a classifier to distinguish those two cases.
4. Use the learned weights as the embeddings.
Skip-Gram Training Data

Assume a +/- 2 word window, given training sentence:

...lemon, a [tablespoon of apricot jam, a] pinch...
Skip-Gram Classifier

(assuming a +/- 2 word window)

...lemon, a [tablespoon of apricot jam, a] pinch...

Goal: train a classifier that is given a candidate (word, context) pair
(apricot, jam)
(apricot, aardvark)
...

And assigns each pair a probability:
\[ P(+ | w, c) \]
\[ P(- | w, c) = 1 - P(+ | w, c) \]
Similarity is computed from dot product

Remember: two vectors are similar if they have a high dot product

- Cosine is just a normalized dot product

So:

- $\text{Similarity}(w,c) \propto w \cdot c$

We’ll need to normalize to get a probability

- (cosine isn't a probability either)
Turning dot products into probabilities

\[ \text{Sim}(w,c) \approx w \cdot c \]

To turn this into a probability

We'll use the sigmoid from logistic regression:

\[
P(+|w,c) = \sigma(c \cdot w) = \frac{1}{1 + \exp(-c \cdot w)}
\]

\[
P(-|w,c) = 1 - P(+|w,c)
\]

\[
= \sigma(-c \cdot w) = \frac{1}{1 + \exp(c \cdot w)}
\]
How Skip-Gram Classifier computes $P(+|w, c)$

$$P( + | w, c) = \sigma(c \cdot w) = \frac{1}{1 + \exp(-c \cdot w)}$$

This is for one context word, but we have lots of context words. We'll assume independence and just multiply them:

$$P( + | w, c_1:L) = \prod_{i=1}^{L} \sigma(c_i \cdot w)$$

$$\log P( + | w, c_1:L) = \sum_{i=1}^{L} \log \sigma(c_i \cdot w)$$
Skip-gram classifier: summary

A probabilistic classifier, given

- a test target word $w$
- its context window of $L$ words $c_{1:L}$

Estimates probability that $w$ occurs in this window based on similarity of $w$ (embeddings) to $c_{1:L}$ (embeddings).

To compute this, we just need embeddings for all the words.
These embeddings we'll need: a set for \( w \), a set for \( c \)

\[
\theta = \begin{bmatrix}
\text{aardvark} & \text{apricot} & \ldots & \text{zebra} \\
\text{aardvark} & \text{apricot} & \ldots & \text{zebra} \\
\text{zebra} & \ldots & \ldots & \text{zebra}
\end{bmatrix}
\]

\( W \) target words

\( C \) context & noise words
Vector Semantics & Embeddings

Word2vec
Word2vec: Learning the embeddings
Skip-Gram Training data

...lemon, a [tablespoon of apricot jam, a] pinch...

c1 c2 [target] c3 c4

positive examples +
t c
_____________________
apricot tablespoon
apricot of
apricot jam
apricot a

This example has a target word (apricot), and 4 context words in the window, resulting in 4 positive training instances (on the left below):

positive examples +
apricot tablespoon
apricot of
apricot jam
apricot a

For training a binary classifier we also need negative examples. In fact skip-gram uses more negative examples than positive examples (with the ratio between them set by a parameter $k$). So for each of these $(t, c)$ training instances we'll create $k$ negative samples, each consisting of the target $t$ plus a 'noise word'. A noise word is a random word from the lexicon, constrained not to be the target word $t$. The right above shows the setting where $k = 2$, so we'll have 2 negative examples in the negative training set for each positive example $t, c$.

The noise words are chosen according to their weighted unigram frequency $p_a(w)$, where $a$ is a weight. If we were sampling according to unweighted frequency $p(w)$, it would mean that with unigram probability $p(\text{the})$ we would choose the word the as a noise word, with unigram probability $p(\text{aardvark})$ we would choose aardvark, and so on. But in practice it is common to set $a = 75$, i.e. use the weighting $p_a(w)$:

$$p_a(w) = \frac{\text{count}(w)}{a \times \text{count}(\text{a})} \quad (6.32)$$

Setting $a = 75$ gives better performance because it gives rare noise words slightly higher probability: for rare words, $p_a(w) > p(w)$.
Skip-Gram Training data

For each positive example we'll grab $k$ negative examples, sampling by frequency.

For training a binary classifier we also need negative examples. In fact skip-gram uses more negative examples than positive examples (with the ratio between them set by a parameter $k$). So for each of these \((t, c)\) training instances we'll create $k$ negative samples, each consisting of the target $t$ plus a 'noise word'. A noise word is a random word from the lexicon, constrained not to be the target word $t$. The right above shows the setting where $k = 2$, so we'll have 2 negative examples in the negative training set for each positive example $t, c$.

The noise words are chosen according to their weighted unigram frequency $p_a(w)$, where $a$ is a weight. If we were sampling according to unweighted frequency $p(w)$, it would mean that with unigram probability $p(\text{"the"})$ we would choose the word \text{the} as a noise word, with unigram probability $p(\text{"aardvark"})$ we would choose aardvark, and so on. But in practice it is common to set $a = 0.75$, i.e. use the weighting $p_3(w)$:

$$P_a(w) = \frac{\text{count}(w)}{a \times \text{count}(w_0) + (1-a) \times \text{count}(w)}$$

Setting $a = 0.75$ gives better performance because it gives rare noise words slightly higher probability: for rare words, $P_a(w) > P(w)$. To visualize this intuition, it might help to work out the probabilities for an example with two events, $P(\text{a}) = 0.99$ and $P(\text{b}) = 0.01$:

$$P_a(\text{a}) = 0.99 \times 0.75^{0.99} + 0.01 \times 0.75^{0.01} = 0.97$$

$$P_a(\text{b}) = 0.01 \times 0.75^{0.99} + 0.99 \times 0.75^{0.01} = 0.03$$

For each positive example we'll grab $k$ negative examples, sampling by frequency.
Skip-Gram Training data

...lemon, a [tablespoon of apricot jam, a] pinch...

c1  c2 [target]  c3  c4

**positive examples**

<table>
<thead>
<tr>
<th>t</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>apricot</td>
<td>tablespoon</td>
</tr>
<tr>
<td>apricot</td>
<td>of</td>
</tr>
<tr>
<td>apricot</td>
<td>jam</td>
</tr>
<tr>
<td>apricot</td>
<td>a</td>
</tr>
</tbody>
</table>

**negative examples**

<table>
<thead>
<tr>
<th>t</th>
<th>c</th>
<th>t</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>apricot</td>
<td>aardvark</td>
<td>apricot</td>
<td>seven</td>
</tr>
<tr>
<td>apricot</td>
<td>my</td>
<td>apricot</td>
<td>forever</td>
</tr>
<tr>
<td>apricot</td>
<td>where</td>
<td>apricot</td>
<td>dear</td>
</tr>
<tr>
<td>apricot</td>
<td>coaxial</td>
<td>apricot</td>
<td>if</td>
</tr>
</tbody>
</table>
Word2vec: how to learn vectors

Given the set of positive and negative training instances, and an initial set of embedding vectors

The goal of learning is to adjust those word vectors such that we:

- **Maximize** the similarity of the target word, context word pairs \((w, c_{pos})\) drawn from the positive data
- **Minimize** the similarity of the \((w, c_{neg})\) pairs drawn from the negative data.
Loss function for one \( w \) with \( c_{pos}, c_{neg1}, \ldots, c_{negk} \)

Maximize the similarity of the target with the actual context words, and minimize the similarity of the target with the \( k \) negative sampled non-neighbor words.

\[
L_{CE} = -\log \left[ P(+|w, c_{pos}) \prod_{i=1}^{k} P(-|w, c_{negi}) \right] \\
= - \left[ \log P(+|w, c_{pos}) + \sum_{i=1}^{k} \log P(-|w, c_{negi}) \right] \\
= - \left[ \log P(+|w, c_{pos}) + \sum_{i=1}^{k} \log (1 - P(+|w, c_{negi})) \right] \\
= - \left[ \log \sigma(c_{pos} \cdot w) + \sum_{i=1}^{k} \log \sigma(-c_{negi} \cdot w) \right]
\]
Learning the classifier

How to learn?
- Stochastic gradient descent!

We’ll adjust the word weights to
- make the positive pairs more likely
- and the negative pairs less likely,
- over the entire training set.
Intuition of one step of gradient descent

move *apricot* and *jam* closer, increasing $c_{pos} \cdot w$

move *apricot* and *matrix* apart decreasing $c_{neg1} \cdot w$

move *apricot* and *Tolstoy* apart decreasing $c_{neg2} \cdot w$

“…apricot jam…”
Reminder: gradient descent

- At each step
  - Direction: We move in the reverse direction from the gradient of the loss function
  - Magnitude: we move the value of this gradient \( \frac{d}{dw} L(f(x;w), y) \) weighted by a learning rate \( \eta \)
  - Higher learning rate means move \( w \) faster

\[
 w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x;w), y)
\]
The derivatives of the loss function

\[
L_{CE} = - \left[ \log \sigma(c_{pos} \cdot w) + \sum_{i=1}^{k} \log \sigma(-c_{neg_i} \cdot w) \right]
\]

\[
\frac{\partial L_{CE}}{\partial c_{pos}} = [\sigma(c_{pos} \cdot w) - 1]w
\]

\[
\frac{\partial L_{CE}}{\partial c_{neg}} = [\sigma(c_{neg} \cdot w)]w
\]

\[
\frac{\partial L_{CE}}{\partial w} = [\sigma(c_{pos} \cdot w) - 1]c_{pos} + \sum_{i=1}^{k} [\sigma(c_{neg_i} \cdot w)]c_{neg_i}
\]
Update equation in SGD

Start with randomly initialized $C$ and $W$ matrices, then incrementally do updates

\[
\begin{align*}
    c_{pos}^{t+1} & = c_{pos}^t - \eta [\sigma(c_{pos}^t \cdot w^t) - 1] w^t \\
    c_{neg}^{t+1} & = c_{neg}^t - \eta [\sigma(c_{neg}^t \cdot w^t)] w^t \\
    w^{t+1} & = w^t - \eta \left[ [\sigma(c_{pos} \cdot w^t) - 1] c_{pos} + \sum_{i=1}^{k} [\sigma(c_{neg_i} \cdot w^t)] c_{neg_i} \right]
\end{align*}
\]
Two sets of embeddings

SGNS learns two sets of embeddings
  Target embeddings matrix W
  Context embedding matrix C
It's common to just add them together, representing word $i$ as the vector $w_i + c_i$
Summary: How to learn word2vec (skip-gram) embeddings

Start with $V$ random $d$-dimensional vectors as initial embeddings

Train a classifier based on embedding similarity

- Take a corpus and take pairs of words that co-occur as positive examples
- Take pairs of words that don't co-occur as negative examples
- Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance
- Throw away the classifier code and keep the embeddings.
Word2vec: Learning the embeddings
Properties of Embeddings
The kinds of neighbors depend on window size

**Small windows** (C= +/- 2) : nearest words are syntactically similar words in same taxonomy

- *Hogwarts* nearest neighbors are other fictional schools
  - *Sunnydale, Evernight, Blandings*

**Large windows** (C= +/- 5) : nearest words are related words in same semantic field

- *Hogwarts* nearest neighbors are Harry Potter world:
  - *Dumbledore, half-blood, Malfoy*
Analogical relations

The classic parallelogram model of analogical reasoning (Rumelhart and Abrahamson 1973)

To solve: "apple is to tree as grape is to _____"

Add tree – apple to grape to get vine
Analogical relations via parallelogram

The parallelogram method can solve analogies with both sparse and dense embeddings (Turney and Littman 2005, Mikolov et al. 2013b)

\[ \text{king} - \text{man} + \text{woman} \text{ is close to queen} \]

\[ \text{Paris} - \text{France} + \text{Italy} \text{ is close to Rome} \]

For a problem \( a : a^* : : b : b^* \), the parallelogram method is:

\[ \hat{b}^* = \arg\max_x \text{distance}(x, a^* - a + b) \]
Caveats with the parallelogram method

It only seems to work for frequent words, small distances and certain relations (relating countries to capitals, or parts of speech), but not others. (Linzen 2016, Gladkova et al. 2016, Ethayarajh et al. 2019a)

Understanding analogy is an open area of research (Peterson et al. 2020)
Embeddings as a window onto historical semantics

Train embeddings on different decades of historical text to see meanings shift

~30 million books, 1850-1990, Google Books data

Embeddings reflect cultural bias!

Bolukbasi, Tolga, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam T. Kalai. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." In NeurIPS, pp. 4349-4357. 2016.

Ask “Paris : France :: Tokyo : x”
  ◦ x = Japan

Ask “father : doctor :: mother : x”
  ◦ x = nurse

Ask “man : computer programmer :: woman : x”
  ◦ x = homemaker

Algorithms that use embeddings as part of e.g., hiring searches for programmers, might lead to bias in hiring
Historical embedding as a tool to study cultural biases


• Compute a **gender or ethnic bias** for each adjective: e.g., how much closer the adjective is to "woman" synonyms than "man" synonyms, or names of particular ethnicities
  • Embeddings for **competence** adjective (*smart, wise, brilliant, resourceful, thoughtful, logical*) are biased toward men, a bias slowly decreasing 1960-1990
  • Embeddings for **dehumanizing** adjectives (barbaric, monstrous, bizarre) were biased toward Asians in the 1930s, bias decreasing over the 20th century.
• These match the results of old surveys done in the 1930s
Properties of Embeddings