LINEAR CLASSIFICATION

We have a feature function \( f(x, y) \) and a score \( \psi_{xy} = \theta^T f(x, y) \)

And return the class with highest score!

\[
\hat{y} = \arg \max_y \theta^T f(x, y)
\]

For each class \( y \in \{ \text{not_beatles, beatles} \} \)

Where did these weights come from? We’ll talk about this in the ML lectures …

(... and also this whole “linear classifier” thing.)
EXPLICIT EXAMPLE

We are interested in classifying documents into one of two classes \( y \in Y = \{ 0, 1 \} = \{ \text{hates\_cats, likes\_cats} \} \)

Document 1: I like cats
Document 2: I hate cats

Now, represent documents with a feature function:

\[
\begin{align*}
\text{f}(x, y = \text{hates\_cats} = 0) &= [x^T, 0^T, 1]^T \\
\text{f}(x, y = \text{likes\_cats} = 1) &= [0^T, x^T, 1]^T
\end{align*}
\]
EXPLICIT EXAMPLE

\[ f(\mathbf{x}, y = 0) = [\mathbf{x}^T, 0^T, 1]^T \]
\[ f(\mathbf{x}, y = 1) = [0^T, \mathbf{x}^T, 1]^T \]

\[ \begin{align*}
x_1^T &= \begin{bmatrix} 1 & 1 & 0 & 1 \end{bmatrix} \\
x_2^T &= \begin{bmatrix} 1 & 0 & 1 & 1 \end{bmatrix}
\end{align*} \]

\[ \begin{array}{c|cccc} l & \text{like} & \text{hate} & \text{cats} \\ \hline y_1 & \text{?} & \text{?} & \text{?} \\ \hline y_2 & \text{?} & \text{?} & \text{?} \\ \hline \end{array} \]

\[ f(\mathbf{x}_1, y = \text{hates}_\text{cats} = 0) = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \]
\[ f(\mathbf{x}_1, y = \text{likes}_\text{cats} = 1) = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix} \]
\[ f(\mathbf{x}_2, y = \text{hates}_\text{cats} = 0) = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \]
\[ f(\mathbf{x}_2, y = \text{likes}_\text{cats} = 1) = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \]

\[ y = 0: \text{hates}_\text{cats} \quad y = 1: \text{likes}_\text{cats} \]
Now, assume we have weights $\theta$ for each feature

$$\theta = \{ \langle l, \text{hates\_cats} \rangle = 0, \langle l, \text{likes\_cats} \rangle = 0, \langle \text{like}, \text{hates\_cats} \rangle = -1, \langle \text{like}, \text{likes\_cats} \rangle = +1, \langle \text{hate}, \text{hates\_cats} \rangle = +1, \langle \text{hate}, \text{likes\_cats} \rangle = -1, \langle \text{cats}, \text{hates\_cats} \rangle = -0.1, \langle \text{cats}, \text{likes\_cats} \rangle = +0.5 \}$$

Write weights as vector that aligns with feature mapping:

<table>
<thead>
<tr>
<th>y=0: hates_cats</th>
<th>y=1: likes_cats</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-1</td>
</tr>
</tbody>
</table>

Parameter vector $\theta^T = (1)$

<table>
<thead>
<tr>
<th>x_1, y = hates_cats = 0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_1, y = likes_cats = 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>x_2, y = hates_cats = 0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>x_2, y = likes_cats = 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Score $\psi$ of an instance $x$ and class $y$ is the sum of the weights for the features in that class:

$$\psi_{xy} = \sum \theta_n f_n(x, y)$$

$$= \theta^T f(x, y)$$

Let's compute $\psi_{x1,y=hates\_cats}$ …

- $\psi_{x1,y=hates\_cats} = \theta^T f(x_1, y = hates\_cats = 0)$
- $= 0*1 + -1*1 + 1*0 + -0.1*1 + 0*0 + 1*0 + -1*0 + 0.5*0 + 1*1$
- $= -1 - 0.1 + 1 = -0.1$

$\theta^T = \begin{bmatrix} 0 & -1 & 1 & -0.1 & 0 & 1 & -1 & 0.5 & 1 \end{bmatrix}$

- $f(x_1, y = 0)$

hates\_cats  likes\_cats

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>hate</td>
<td>hate</td>
<td>hate</td>
<td>hate</td>
<td>like</td>
<td>like</td>
<td>like</td>
</tr>
<tr>
<td>cats</td>
<td>cats</td>
<td>cats</td>
<td>cats</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
EXPLICIT EXAMPLE

Saving the boring stuff:
• \( \psi_{x1,y=hates\_cats} = -0.1; \psi_{x1,y=likes\_cats} = +2.5 \)
• \( \psi_{x2,y=hates\_cats} = +1.9; \psi_{x2,y=likes\_cats} = +0.5 \)

We want to predict the class of each document:

\[
\hat{y} = \text{arg max } \theta^\top f(x, y)
\]

Document 1: \( \text{argmax}\{ \psi_{x1,y=hates\_cats}, \psi_{x1,y=likes\_cats} \} \)
Document 2: \( \text{argmax}\{ \psi_{x2,y=hates\_cats}, \psi_{x2,y=likes\_cats} \} \)
(MINIMUM) EDIT DISTANCE

How similar are two strings?
Many different distance metrics (as we saw earlier when discussing entity resolution)
  • Typically based on the number of edit operations needed to transform from one to the other

Useful in NLP context for spelling correction, information extraction, speech recognition, etc.
NLP IN PYTHON

Two majors libraries for performing basic NLP in Python:

• Natural Language Toolkit (**NLTK**): started as research code, now widely used in industry and research
• **Spacy**: much newer implementation, more streamlined

Pros and cons to both:

• NLTK has more “stuff” implemented, is more customizable
  • This is a blessing and a curse
• Spacy is younger and feature sparse, but can be much faster
• Both are Anaconda packages
import nltk

# Tokenize, aka find the terms in, a sentence
sentence = "A wizard is never late, nor is he early. He arrives precisely when he means to."
tokens = nltk.word_tokenize(sentence)

LookupError:
************************************************
Resource 'tokenizers/punkt/PY3/english.pickle' not found.
Please use the NLTK Downloader to obtain the resource: 
>>> nltk.download()
Searched in:
  - '/Users/spook/nltk_data'
  - '/usr/share/nltk_data'
  - '/usr/local/share/nltk_data'
  - '/usr/lib/nltk_data'
  - '/usr/local/lib/nltk_data'

************************************************

Fool of a Took!
Corpora are, by definition, large bodies of text

- NLTK relies on a large corpus set to perform various functionalities; you can pick and choose:

```python
# Launch a GUI browser of available corpora
nltk.download()
```

# Or download everything at once!
```python
nltk.download("all")
```
import nltk

# Tokenize, aka find the terms in, a sentence
sentence = "A wizard is never late, nor is he early. He arrives precisely when he means to."
tokens = nltk.word_tokenize(sentence)

['A', 'wizard', 'is', 'never', 'late', ',', ',', 'nor', 'is', 'he', 'early', ',', '.', 'He', 'arrives', 'precisely', 'when', 'he', 'means', 'to', '.']

(This will also tokenize words like “o’clock” into one term, and “didn’t” into two term, “did” and “n’t”.)
# Determine parts of speech (POS) tags

tagged = nltk.pos_tag(tokens)
tagged[:10]

[('A', 'DT'), ('wizard', 'NN'), ('is', 'VBZ'),
('never', 'RB'), ('late', 'RB'), ('nor', 'CC'), ('is', 'VBZ'), ('he', 'PRP'), ('early', 'RB')]

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>POS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT</td>
<td>Determiner</td>
</tr>
<tr>
<td>NN</td>
<td>Noun</td>
</tr>
<tr>
<td>VBZ</td>
<td>Verb (3rd person singular present)</td>
</tr>
<tr>
<td>RB</td>
<td>Adverb</td>
</tr>
<tr>
<td>CC</td>
<td>Conjunction</td>
</tr>
<tr>
<td>PRP</td>
<td>Personal Pronoun</td>
</tr>
</tbody>
</table>

Full list: https://cs.nyu.edu/grishman/jet/guide/PennPOS.html
# Find named entities & visualize

```python
text = 
"The Shire was divided into four quarters, the Farthings already referred to. North, South, East, and West; and these again each into a number of folklands, which still bore the names of some of the old leading families, although by the time of this history these names were no longer found only in their proper folklands. Nearly all Tooks still lived in the Tookland, but that was not true of many other families, such as the Bagginses or the Boffins. Outside the Farthings were the East and West Marches: the Buckland (see beginning of Chapter V, Book I); and the Westmarch added to the Shire in S.R. 1462.""

entities = nltk.chunk.ne_chunk(nltk.pos_tag(nltk.word_tokenize(text)))
entities.draw()
```
“fast” is similar to “rapid”
“tall” is similar to “height”

Question answering:

Q: “How tall is Mt. Everest?”
Candidate A: “The official height of Mount Everest is 29,029 feet”
A bottle of *tesgüino* is on the table
Everybody likes *tesgüino*
*Tesgüino* makes you drunk
We make *tesgüino* out of corn.

From context words humans can guess *tesgüino* means
- an alcoholic beverage like beer

Intuition for algorithm:
- Two words are similar if they have similar word contexts.
FOUR KINDS OF VECTOR MODELS

Sparse vector representations
  • Mutual-information weighted word co-occurrence matrices

Dense vector representations:
  • Singular value decomposition (and Latent Semantic Analysis)
  • Neural-network-inspired models (skip-grams, CBOW)
  • Brown clusters
    • Won’t go into these much – basically, classify terms into “word classes” using a particular clustering method
    • Hard clustering due to Brown et al. 1992, embed words in some space and cluster. Generally, better methods out there now …
Model the meaning of a word by embedding in a vector space. The meaning of a word is a vector of numbers

- Vector models are also called “embeddings”.

Contrast: word meaning is represented in many computational linguistic applications by a vocabulary index (“word number 545”)

SHARED INTUITION
REMINDER: TERM-DOCUMENT MATRIX

Each cell: count of term $t$ in a document $d$: $tf_{t,d}$:

- Each document is a count vector in $\mathbb{N}^v$: a column below

<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>1</td>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td>soldier</td>
<td>2</td>
<td>2</td>
<td>12</td>
<td>36</td>
</tr>
<tr>
<td>fool</td>
<td>37</td>
<td>58</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>clown</td>
<td>6</td>
<td>117</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
REMINDER: TERM-DOCUMENT MATRIX

Two documents are similar if their vectors are similar

<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>1</td>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td>soldier</td>
<td>2</td>
<td>2</td>
<td>12</td>
<td>36</td>
</tr>
<tr>
<td>fool</td>
<td>37</td>
<td>58</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>clown</td>
<td>6</td>
<td>117</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
THE WORDS IN A TERM-DOCUMENT MATRIX

Each word is a count vector in $\mathbb{N}^D$: a row below

<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>1</td>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td>soldier</td>
<td>2</td>
<td>2</td>
<td>12</td>
<td>36</td>
</tr>
<tr>
<td>fool</td>
<td>37</td>
<td>58</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>clown</td>
<td>6</td>
<td>117</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
THE WORDS IN A TERM-DOCUMENT MATRIX

Two words are similar if their vectors are similar

<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>1</td>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td>soldier</td>
<td>2</td>
<td>2</td>
<td>12</td>
<td>36</td>
</tr>
<tr>
<td>fool</td>
<td>37</td>
<td>58</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>clown</td>
<td>6</td>
<td>117</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
**TERM-CONTEXT MATRIX FOR WORD SIMILARITY**

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

<table>
<thead>
<tr>
<th></th>
<th>aardvark</th>
<th>computer</th>
<th>data</th>
<th>pinch</th>
<th>result</th>
<th>sugar</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>apricot</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>pineapple</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>digital</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>information</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
THE WORD-WORD OR WORD-CONTEXT MATRIX

Instead of entire documents, use smaller contexts
  • Paragraph
  • Window of 4 words

A word is now defined by a vector over counts of context words
  • Instead of each vector being of length $D$
  • Each vector is now of length $|V|$

The word-word matrix is $|V| \times |V|$, not $D \times D$
sugar, a sliced lemon, a tablespoonful of their enjoyment. Cautiously she sampled her first well suited to programming on the digital for the purpose of gathering data and apricot pineapple computer information preserve or jam, a pinch each of, and another fruit whose taste she likened In finding the optimal R-stage policy from necessary for the study authorized in the

|               | aardvark | computer | data | pinch | result | sugar | ...
|---------------|----------|----------|------|-------|--------|-------|------
| apricot       | 0        | 0        | 0    | 1     | 0      | 1     | ...
| pineapple     | 0        | 0        | 0    | 1     | 0      | 1     | ...
| digital       | 0        | 2        | 1    | 0     | 1      | 0     | ...
| information   | 0        | 1        | 6    | 0     | 4      | 0     | ...
|               |          |          |      |       |        |       | ...

[DJ]
WORD-WORD MATRIX

We showed only 4x6, but the real matrix is 50,000 x 50,000

- So it’s very sparse
  - Most values are 0.
- That’s OK, since there are lots of efficient algorithms for sparse matrices.

The size of windows depends on your goals

- The shorter the windows, the more syntactic the representation
  - 1-3 very syntacticy
- The longer the windows, the more semantic the representation
  - 4-10 more semantic
MEASURING SIMILARITY

Given 2 target words v and w
• Need a way to measure their similarity.

Most measure of vectors similarity are based on the:
• Dot product or inner product from linear algebra

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

• High when two vectors have large values in same dimensions.
• Low (in fact 0) for orthogonal vectors with zeros in complementary distribution
PROBLEM WITH DOT PRODUCT

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

Dot product is longer if the vector is longer. Vector length:

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

Vectors are longer if they have higher values in each dimension
That means more frequent words will have higher dot products
That’s bad: we don’t want a similarity metric to be sensitive to word frequency
SOLUTION: COSINE

Just divide the dot product by the length of the two vectors!

\[
\frac{\vec{a} \cdot \vec{b}}{|\vec{a}| |\vec{b}|}
\]

This turns out to be the cosine of the angle between them!

\[
\frac{\vec{a} \cdot \vec{b}}{|\vec{a}| |\vec{b}|} = \cos \theta
\]
SIMILARITY BETWEEN DOCUMENTS

Given two documents x and y, represented by their TF-IDF vectors (or any vectors), the cosine similarity is:

\[
\text{similarity}(x, y) = \frac{x^\top y}{|x| \times |y|}
\]

Formally, it measures the cosine of the angle between two vectors x and y:

- \(\cos(0^\circ) = 1\), \(\cos(90^\circ) = 0\)

Similar documents have high cosine similarity; dissimilar documents have low cosine similarity.
EXAMPLE

$\cos(v, w) = \frac{v \cdot w}{|v||w|} = \frac{v}{|v|} \cdot \frac{w}{|w|} = \frac{\sum_{i=1}^{n} v_i w_i}{\sqrt{\sum_{i=1}^{n} v_i^2} \sqrt{\sum_{i=1}^{n} w_i^2}}$

Which pair of words is more similar?

$\cosine(\text{apricot}, \text{information}) = \frac{2 + 0 + 0}{\sqrt{2 + 0 + 0} \sqrt{1 + 36 + 1}} = \frac{2}{\sqrt{2}\sqrt{38}} = .23$

$\cosine(\text{digital}, \text{information}) = \frac{0 + 6 + 2}{\sqrt{0 + 1 + 4} \sqrt{1 + 36 + 1}} = \frac{8}{\sqrt{38}\sqrt{5}} = .58$

$\cosine(\text{apricot}, \text{digital}) = \frac{0 + 0 + 0}{\sqrt{1 + 0 + 0} \sqrt{0 + 1 + 4}} = 0$

<table>
<thead>
<tr>
<th></th>
<th>large</th>
<th>data</th>
<th>computer</th>
</tr>
</thead>
<tbody>
<tr>
<td>apricot</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>digital</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>information</td>
<td>1</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>
How similar are two strings?

Many different distance metrics (as we saw earlier when discussing entity resolution)

- Typically based on the number of edit operations needed to transform from one to the other

Useful in NLP context for spelling correction, information extraction, speech recognition, etc.
Language Models
LANGUAGE MODELING

Assign a probability to a sentence

• Machine Translation:
  • \( P(\text{high winds tonite}) > P(\text{large winds tonite}) \)

• Spell Correction
  • The office is about fifteen \textit{minuets} from my house
    • \( P(\text{about fifteen minutes from}) > P(\text{about fifteen minuets from}) \)

• Speech Recognition
  • \( P(\text{I saw a van}) \gg P(\text{eyes awe of an}) \)

• + Summarization, question-answering, etc., etc.!!
LANGUAGE MODELING

Goal: compute the probability of a sentence or sequence of words:
• \( P(W) = P(w_1,w_2,w_3,w_4,w_5\ldots w_n) \)

Related task: probability of an upcoming word:
• \( P(w_5|w_1,w_2,w_3,w_4) \)

A model that computes either of these:
• \( P(W) \) or \( P(w_n|w_1,w_2\ldots w_{n-1}) \) is called a language model.

(We won’t talk about this much further in this class.)
BIGRAM MODEL

Condition on the previous word:

\[ P(w_i | w_1 w_2 ... w_{i-1}) \approx P(w_i | w_{i-1}) \]

texaco, rose, one, in, this, issue, is, pursuing, growth, in, a, boiler, house, said, mr., gurria, mexico, 's, motion, control, proposal, without, permission, from, five, hundred, fifty, five, yen

outside, new, car, parking, lot, of, the, agreement, reached

this, would, be, a, record, november
EXAMPLE

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

\[
P(I | < s > ) = \frac{2}{3} \quad P(Sam | < s > ) = \frac{1}{3} \quad P(am | I) = \frac{2}{3}
\]

\[
P( < /s > | Sam) = \frac{1}{2} \quad P(Sam | am) = \frac{1}{2} \quad P(do | I) = \frac{1}{3}
\]

Some bigram probabilities
N-GRAM MODELS

We can extend to trigrams, 4-grams, 5-grams
In general this is an insufficient model of language
  • because language has long-distance dependencies:

  • “The computer which I had just put into the machine room on the fifth floor crashed.”

But we can often get away with N-gram models