# PRINCIPLES OF DATA SCIENCE

JOHN P DICKERSON

**Lecture #9 – 10/24/2018**

**CMSC641**
**Wednesdays**
**7:00pm – 9:30pm**

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

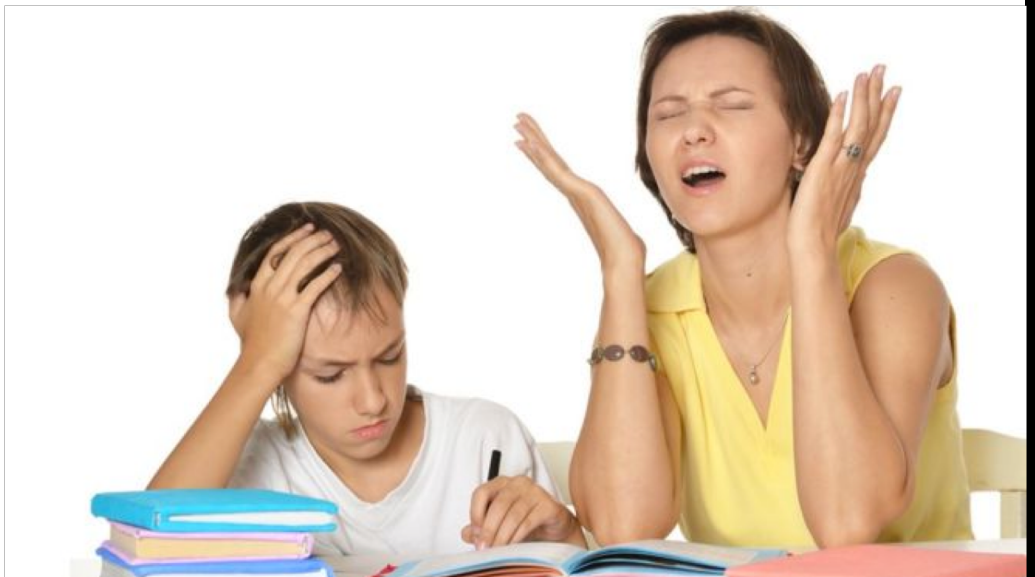
# ANNOUNCEMENTS

**Mini-Project #2 was due today!**

**Mini-Project #3 is not out yet!  But will be soon …**

- It is linked to from ELMS; will be available at:
  https://github.com/umddb/cmsc641-fall2018/tree/master/project3
- Deliverable is a .ipynb file submitted to ELMS
- Due Wednesday, November 14th

**After this: final tutorials!**



# THE MINI-TUTORIAL

**In lieu of a final exam, you'll create a mini-tutorial that:**

- Identifies a raw data source
- Processes and stores that data
- Performs exploratory data analysis & visualization
- Derives insight(s) using statistics and ML
- Communicates those insights as actionable text

**Individual or group project**

**Will be hosted publicly online (GitHub Pages) and will strengthen your portfolio.**

# READY-MADE DATASET REPOSITORIES

https://www.data.gov/

- US-centric agriculture, climate, education, energy, finance, health, manufacturing data, …

https://cloud.google.com/bigquery/public-data/

- BigQuery (Google Cloud) public datasets (bikeshare, GitHub, Hacker News, Form 990 non-profits, NOAA, …)

https://www.kaggle.com/datasets

- Microsoft-owned, various (Billboard Top 100 lyrics, credit card fraud, crime in Chicago, global terrorism, world happiness, …)

https://aws.amazon.com/public-datasets/

- AWS-hosted, various (NASA, a bunch of genome stuff, Google Books n-grams, Multimedia Commons, …)

# NEW DATASET IDEAS

**Fraternal Order of Police vs Black Lives Matter**

**Linking finance data to ${anything_else}**

**Something having to do with Pokémon statistics?**

**Look through http://www.alexa.com/topsites and scrape something interesting!**

**University of Maryland-related, or College Park-related, stuff**

- Check out http://umd.io/ – open source project; maybe your data collection and cleaning scripts can be added to this!

**Honestly, pretty much anything! Just document everything.**

Reproducibility!

# FINAL TUTORIAL

**Deliverable: URL of your own GitHub Pages site hosting an .ipynb/.html export of your final tutorial**

- https://pages.github.com/ – make a GitHub account, too!
- https://github.com/blog/1995-github-jupyter-notebooks-3

**The project itself:**

- ~1500+ words of Markdown prose
- ~150+ lines of Python
- Should be viewable as a **static webpage** – that is, if I (or anyone else) opens the link up, everything should render and I shouldn't have to run any cells to generate output

# FINAL TUTORIAL RUBRIC

**I will grade on a scale of 1-10:**

**Motivation:** Does the tutorial make the reader believe the topic is important (a) in general and (b) with respect to data science?

**Understanding:** After reading the tutorial, does the reader understand the topic?

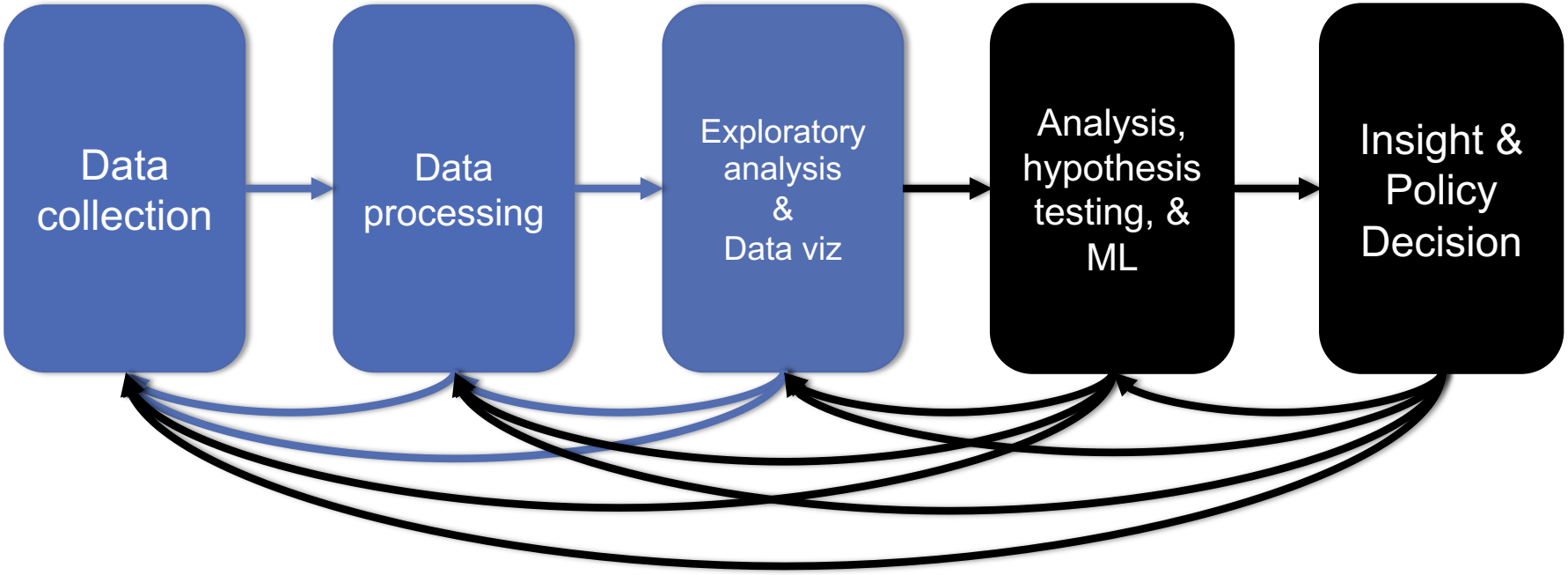
**Further resources:** Does the tutorial "call out" to other resources that would help the reader understand basic concepts, deep dive, related work, etc?

**Prose:** Does the prose in the Markdown portion of the .ipynb add to the reader's understanding of the tutorial?

**Code:** Does the code help solidify understanding, is it well documented, and does it include helpful examples?

**Subjective Evaluation:** If somebody linked to this tutorial from Hacker News, would people actually read the whole thing?

# WRAP-UP FROM LAST LECTURE …



# CONTINUING FROM LAST CLASS …

**Words words words!**

- Free text and natural language processing in data science
- Bag of words and TF-IDF
- N-Grams and language models
- Information extraction & sentiment mining

**Thanks to: Zico Kolter (CMU), Dan Jurafsky (Stanford), & Marine Carpuat's 723 (UMD)**




# TEXT CLASSIFICATION

**Input:**

- A document $w$
- A set of classes $Y = \{y_1, y_2, \ldots, y_J\}$
- A training set of $m$ hand-labeled documents
  $\{(w_1, y_1), (w_2, y_2), \ldots, (w_m, y_m)\}$

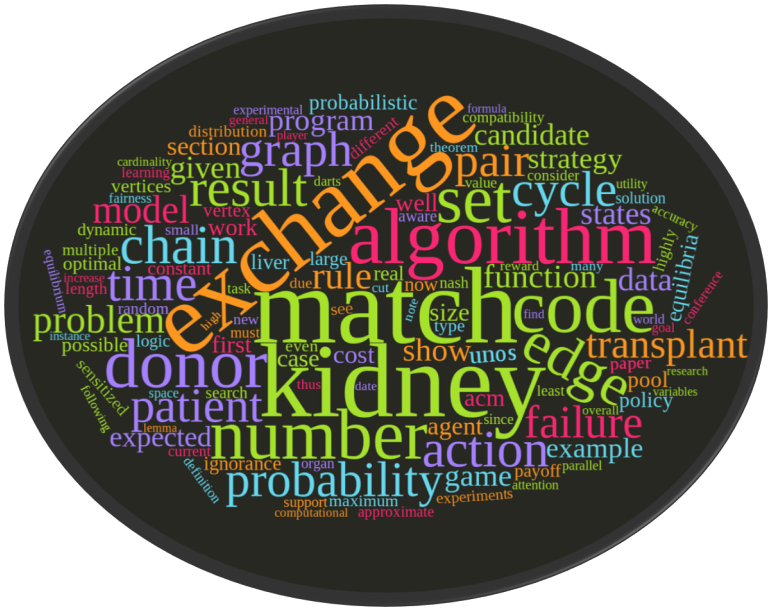
**Output:**

- A learned classifier $w \rightarrow y$

**This is an example of supervised learning**

# REPRESENTING A DOCUMENT "IN MATH"

**Simplest method: bag of words**




**Represent each document as a vector of word frequencies**

- Order of words does not matter, just #occurrences

# BAG OF WORDS EXAMPLE

**the quick brown fox jumps over the lazy dog**

**I am he as you are he as you are me**

**he said the CMSC641 is 510 more CMSCs than the CMSC131**

| | the | CMSC641 | you | he | I | quick | dog | me | CMSCs | … | than |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Document 1 | 2 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | | 0 |
| Document 2 | 0 | 0 | 2 | 2 | 1 | 0 | 0 | 1 | 0 | … | 0 |
| Document 3 | 2 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | | 1 |

# TERM FREQUENCY

**Term frequency: the number of times a term appears in a specific document**

- $tf_{ij}$: frequency of word *j* in document *i*

**This can be the raw count (like in the BOW in the last slide):**

- $tf_{ij} \in \{0,1\}$ if word *j* appears or doesn't appear in doc *i*
- $\log(1 + tf_{ij})$ – reduce the effect of outliers
- $tf_{ij} / \max_j tf_{ij}$ – normalize by document i's most frequent word

**What can we do with this?**

- Use as features to learn a classifier $w \rightarrow y$ …!

# DEFINING FEATURES FROM TERM FREQUENCY

**Suppose we are classifying if a document was written by The Beatles or not (i.e., binary classification):**

- Two classes $y \in Y = \{ 0, 1 \} = \{$ not_beatles, beatles $\}$

**Let's use $tf_{ij} \in \{0,1\}$, which gives:**

| | the | CMSC641 | you | he | I | quick | dog | me | CMSCs | ... | than |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $x_1^T =$ | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | | 0 |
| $x_2^T =$ | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | … | 0 |
| $x_3^T =$ | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | | 1 |

$y_1 = 0$

$y_2 = 1$

$y_3 = 0$

**Then represent documents with a feature function:**

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

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

# LINEAR CLASSIFICATION

**We can then define weights $\boldsymbol{\theta}$ for each feature**

$\boldsymbol{\theta}$ = { <CMSC641, not_beatles> = +1,
<CMSC641, beatles> = -1,
<walrus, not_beatles> = -0.3,
<walrus, beatles> = +1,
<the, not_beatles> = 0,
<the, beatles>, 0, … }

**Write weights as vector that aligns with feature mapping**

**Score $\psi$ of an instance $x$ and class $y$ is the sum of the weights for the features in that class:**

$$\psi_{\mathbf{x}y} = \Sigma\, \theta_n\, f_n(\mathbf{x}, y)$$

$$= \boldsymbol{\theta}^{\mathsf{T}}\, \mathbf{f}(\mathbf{x}, y)$$

# 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!

Compute the score of the document for that class

$$\hat{y} = \arg\max_{y} \theta^T \mathbf{f}(\mathbf{x}, y)$$

For each class y ∈ { not_beatles, beatles }

(… and also this whole "linear classifier" thing.)

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

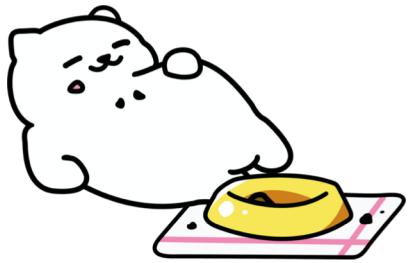
# EXPLICIT EXAMPLE

**We are interested in classifying documents into one of two classes $y \in Y = \{ 0, 1 \} = \{$ hates_cats, likes_cats$\}$**

**Document 1: I like cats**

**Document 2: I hate cats**

| | I | like | hate | cats |
|---|---|---|---|---|
| $x_1^T =$ | 1 | 1 | 0 | 1 |
| $x_2^T =$ | 1 | 0 | 1 | 1 |



$y_1 = ?$

$y_2 = ?$

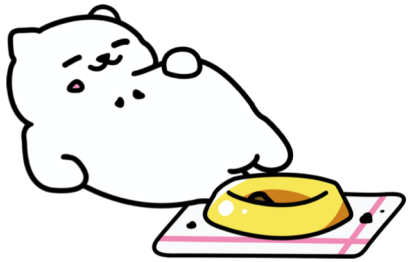
**Now, represent documents with a feature function:**

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

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

# EXPLICIT EXAMPLE

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



| | I | like | hate | cats |
|---|---|---|---|---|
| $x_1^T =$ | 1 | 1 | 0 | 1 |
| $x_2^T =$ | 1 | 0 | 1 | 1 |

$y_1 = ?$

$y_2 = ?$

*y=0: hates_cats* *y=1: likes_cats* (1)

| | I | like | hate | cats | I | like | hate | cats | .. |
|---|---|---|---|---|---|---|---|---|---|
| $f(\mathbf{x_1}, y = \text{hates\_cats} = 0) =$ | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| $f(\mathbf{x_1}, y = \text{likes\_cats} = 1) =$ | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| $f(\mathbf{x_2}, y = \text{hates\_cats} = 0) =$ | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| $f(\mathbf{x_2}, y = \text{likes\_cats} = 1) =$ | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |

# EXPLICIT EXAMPLE

**Now, assume we have weights $\theta$ for each feature**

$\theta$ = { <I, hates_cats> = 0, <I, likes_cats> = 0,
<like, hates_cats> = -1, <like, likes_cats> = +1,
<hate, hates_cats> = +1, <hate, likes_cats> = -1,
<cats, hates_cats> = -0.1, <cats, likes_cats = +0.5> }

**Write weights as vector that aligns with feature mapping:**

| | *y=0: hates_cats* | | | | *y=1: likes_cats* | | | | (1) |
|---|---|---|---|---|---|---|---|---|---|
| Parameter vector $\theta^T$ = | 0 | -1 | 1 | -0.1 | 0 | 1 | -1 | 0.5 | 1 |
| | I | like | hate | cats | I | like | hate | cats | - |
| f($\mathbf{x}_1$, y = hates_cats = 0) = | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| f($\mathbf{x}_1$, y = likes_cats = 1) = | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| f($\mathbf{x}_2$, y = hates_cats = 0) = | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| f($\mathbf{x}_2$, y = likes_cats = 1) = | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |

# EXPLICIT EXAMPLE

**Score $\psi$ of an instance *x* and class *y* is the sum of the weights for the features in that class:**

$$\psi_{\mathbf{x}y} = \Sigma\, \theta_n\, f_n(\mathbf{x}, y)$$

$$= \boldsymbol{\theta}^{\mathsf{T}}\, \mathbf{f}(\mathbf{x}, y)$$

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

- $\psi_{\mathbf{x1},y=hates\_cats} = \boldsymbol{\theta}^{\mathsf{T}}\, \mathbf{f}(\mathbf{x}_1, y = \text{hates\_cats} = 0)$
- $= 0*1 + -1*1 + 1*0 + -0.1*1 + 0*0 + 1*0 + -1*0 + 0.5*0 + 1*1$
- $= -1 - 0.1 + 1 = \mathbf{-0.1}$

$\boldsymbol{\theta}^{\mathsf{T}} =$

| 0 | -1 | 1 | -0.1 | 0 | 1 | -1 | 0.5 | 1 |
|---|---|---|---|---|---|---|---|---|

●

| | | |
|---|---|---|
| 1 | I | |
| 1 | like | |
| 0 | hate | *hates_cats* |
| 1 | cats | |
| 0 | I | |
| 0 | like | *likes_cats* |
| 0 | hate | |
| 0 | cats | |
| 1 | – | (1) |

$\mathbf{f}(\mathbf{x}_1, y = 0)$

# EXPLICIT EXAMPLE

**Saving the boring stuff:**

- $\psi_{\mathbf{x1},y=hates\_cats} = -0.1$; $\psi_{\mathbf{x1},y=likes\_cats} = +2.5$ Document 1: I like cats
- $\psi_{\mathbf{x2},y=hates\_cats} = +1.9$; $\psi_{\mathbf{x2},y=likes\_cats} = +0.5$ Document 2: I hate cats

**We want to predict the class of each document:**

$$\hat{y} = \arg\max_{y} \theta^{\top} \mathbf{f}(\mathbf{x}, y)$$

**Document 1: argmax{ $\psi_{\mathbf{x1},y=hates\_cats}$, $\psi_{\mathbf{x1},y=likes\_cats}$ } ????????**

**Document 2: argmax{ $\psi_{\mathbf{x2},y=hates\_cats}$, $\psi_{\mathbf{x2},y=likes\_cats}$ } ????????**

# INVERSE DOCUMENT FREQUENCY

**Recall:**

- $tf_{ij}$: frequency of word *j* in document *i*

**Any issues with this ??????????**

- Term frequency gets overloaded by common words

**Inverse Document Frequency (IDF): weight individual words negatively by how frequently they appear in the corpus:**

$$\text{idf}_j = \log\left(\frac{\#\text{documents}}{\#\text{documents with word } j}\right)$$

**IDF is just defined for a word j, not word/document pair j, i**

# INVERSE DOCUMENT FREQUENCY

| | the | CMSC641 | you | he | I | quick | dog | me | CMSCs | ... | than |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Document 1 | 2 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | | 0 |
| Document 2 | 0 | 0 | 2 | 2 | 1 | 0 | 0 | 1 | 0 | ... | 0 |
| Document 3 | 2 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | | 1 |

$$\text{idf}_{\text{the}} = \log\left(\frac{3}{2}\right) = 0.405$$

$$\text{idf}_{\text{you}} = \log\left(\frac{3}{1}\right) = 1.098$$

$$\text{idf}_{\text{CMSC320}} = \log\left(\frac{3}{1}\right) = 1.098$$

$$\text{idf}_{\text{he}} = \log\left(\frac{3}{2}\right) = 0.405$$

# TF-IDF

**How do we use the IDF weights?**

**Term frequency inverse document frequency (TF-IDF):**

- TF-IDF score: $tf_{ij}$ x $idf_j$

| | the | CMSC641 | you | he | I | quick | dog | me | CMSCs | … | than |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Document 1 | 0.8 | 0 | 0 | 0 | 0 | 1.1 | 1.1 | 0 | 0 | | 0 |
| Document 2 | 0 | 0 | 2.2 | 0.8 | 1.1 | 0 | 0 | 1.1 | 0 | … | 0 |
| Document 3 | 0.8 | 1.1 | 0 | 0.4 | 0 | 0 | 0 | 0 | 1.1 | | 1.1 |

**This ends up working better than raw scores for classification and for computing similarity between documents.**

# TOKENIZATION

**First step towards text processing**

**For English, just split on non-alphanumerical characters**

- Need to deal with cases like: I'm, or France's, or Hewlett-Packard
- Should "San Francisco" be one token or two?

**Other languages introduce additional issues**

- L'ensemble → one token or two?
- German noun compounds are not segmented
    - Lebensversicherungsgesellschaftsangestellter
- Chinese/Japanese more complicated because of white spaces

# OTHER BASIC TERMS

**Lemmatization**

- Reduce inflections or variant forms to base form
  - am, are, is → be
  - car, cars, car's, cars' → car
- the boy's cars are different colors → the boy car be different color

**Morphology/Morphemes**

- The small meaningful units that make up words
- Stems: The core meaning-bearing units
- Affixes: Bits and pieces that adhere to stems
  - Often with grammatical functions

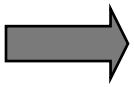
# STEMMING

**Reduce terms to their stems in information retrieval**

**Stemming is crude chopping of affixes**

- language dependent
- e.g., **automate(s)**, **automatic**, **automation** all reduced to **automat**.

*for example compressed and compression are both accepted as equivalent to compress*.



for exampl compress and compress ar both accept as equival to compress

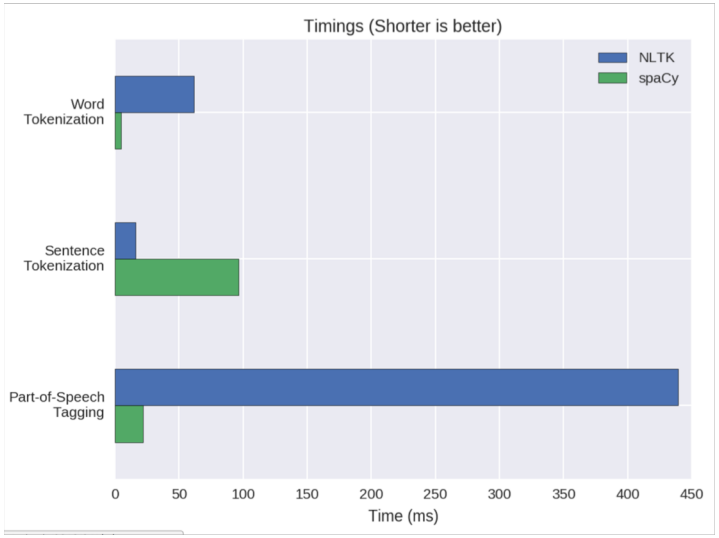
# NLP IN PYTHON


**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

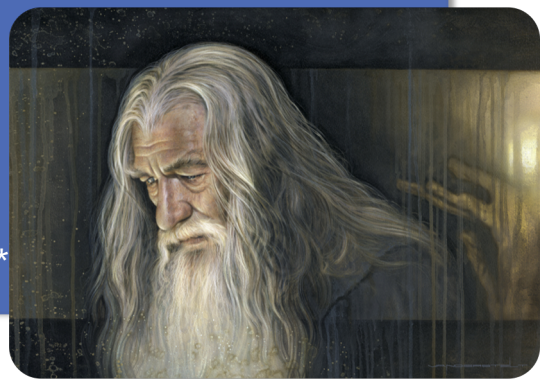


# NLTK EXAMPLES

```
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'
    - ''
**********************************************************************
```



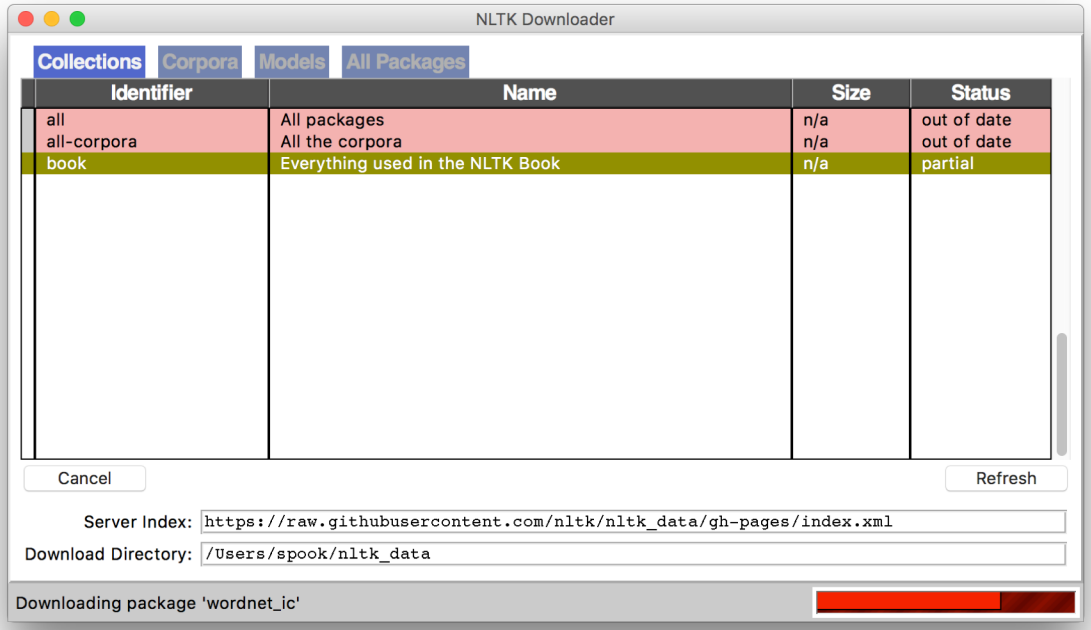
*Fool of a Took!*

# NLTK EXAMPLES

**Corpora are, by definition, large bodies of text**

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

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




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

# NLTK EXAMPLES

| | | | |
|---|---|---|---|
| ptb | Penn Treebank | 6.1 KB | not installed |
| punkt | Punkt Tokenizer Models | 13.0 MB | installed |
| qc | Experimental Data for Question Classification | 122.5 KB | not installed |

```
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".)**

# NLTK EXAMPLES

```
# 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')]
```

| Abbreviation | POS |
|---|---|
| DT | Determiner |
| NN | Noun |
| VBZ | Verb (3rd person singular present) |
| RB | Adverb |
| CC | Conjunction |
| PRP | Personal Pronoun |

Full list: https://cs.nyu.edu/grishman/jet/guide/PennPOS.html

# NLTK EXAMPLES

```
# Find named entities & visualize
entities = nltk.chunk.ne_chunk( nltk.pos_tag(
nltk.word_tokenize("""
    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.draw()
```

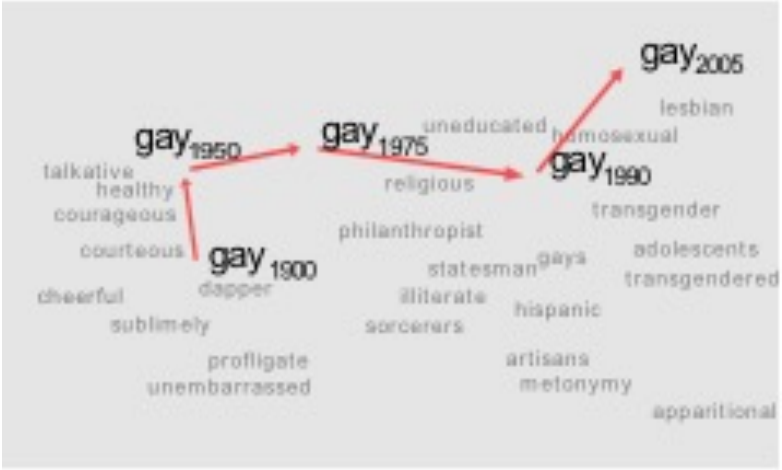
# BRIEF ASIDE: VECTOR SEMANTICS OF DOCS/TERMS

**"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 29029 feet"***




[Kulkarni, Al-Rfou, Perozzi, Skiena 2015]

*Many thanks to Dan Jurafsky here!*

# DISTRIBUTIONAL MODELS OF MEANING

**Distributional models of meaning**
**= vector-space models of meaning**
**= vector semantics**

**Intuitions: Zellig Harris (1954):**

- "oculist and eye-doctor … occur in almost the same environments"
- "If A and B have almost identical environments we say that they are synonyms."

**Firth (1957):**

- "You shall know a word by the company it keeps!"

[DJ]

# INTUITION OF DISTRIBUTIONAL WORD SIMILARITY

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.

[DJ]

# 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 …

[DJ]

# SHARED INTUITION

**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”)**

[DJ]

# 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

| | As You Like It | Twelfth Night | Julius Caesar | Henry V |
|---|---|---|---|---|
| battle | 1 | 1 | 8 | 15 |
| soldier | 2 | 2 | 12 | 36 |
| fool | 37 | 58 | 1 | 5 |
| clown | 6 | 117 | 0 | 0 |

[DJ]

# REMINDER: TERM-DOCUMENT MATRIX

**Two documents are similar if their vectors are similar**

| | As You Like It | Twelfth Night | Julius Caesar | Henry V |
|---|---|---|---|---|
| battle | 1 | 1 | 8 | 15 |
| soldier | 2 | 2 | 12 | 36 |
| fool | 37 | 58 | 1 | 5 |
| clown | 6 | 117 | 0 | 0 |

[DJ]

# THE WORDS IN A TERM-DOCUMENT MATRIX

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

| | As You Like It | Twelfth Night | Julius Caesar | Henry V |
|---|---|---|---|---|
| battle | 1 | 1 | 8 | 15 |
| soldier | 2 | 2 | 12 | 36 |
| fool | 37 | 58 | 1 | 5 |
| clown | 6 | 117 | 0 | 0 |

[DJ]

# THE WORDS IN A TERM-DOCUMENT MATRIX

**Two words are similar if their vectors are similar**

| | As You Like It | Twelfth Night | Julius Caesar | Henry V |
|---|---|---|---|---|
| battle | 1 | 1 | 8 | 15 |
| soldier | 2 | 2 | 12 | 36 |
| fool | 37 | 58 | 1 | 5 |
| clown | 6 | 117 | 0 | 0 |

[DJ]

# TERM-CONTEXT MATRIX FOR WORD SIMILARITY

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

| | 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]

# 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*|x|*V*|, not *D*x*D***

[DJ]

# WORD-WORD MATRIX
# SAMPLE CONTEXTS ± 7 WORDS

sugar, a sliced lemon, a tablespoonful of **apricot** preserve or 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 | 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 semanticy

[DJ]

# 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

[DJ]

# 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**

[DJ]

# 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!**

$$\vec{a} \cdot \vec{b} = |\vec{a}||\vec{b}| \cos\theta$$

$$\frac{\vec{a} \cdot \vec{b}}{|\vec{a}||\vec{b}|} = \cos\theta$$

# SIMILARITY BETWEEN DOCUMENTS

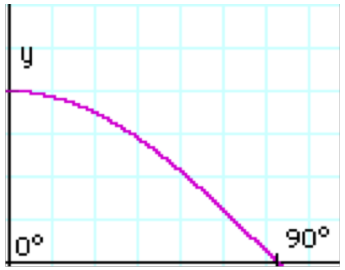
**Given two documents x and y, represented by their TF-IDF vectors (or any vectors), the cosine similarity is:**

$$\text{similarity}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x}^\intercal \mathbf{y}}{|\mathbf{x}| \times |\mathbf{y}|}$$

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

- $\cos(0^o) = 1$, $\cos(90^o) = 0$ ??????????

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



# EXAMPLE

| | large | data | computer |
|---|---|---|---|
| apricot | 2 | 0 | 0 |
| digital | 0 | 1 | 2 |
| information | 1 | 6 | 1 |

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

Which pair of words is more similar?

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

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

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

# (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.**

# BRIEF ASIDE: N-GRAMS

**n-gram: Contiguous sequence of n tokens/words etc.**

- Unigram, bigram, trigram, "four-gram", "five-gram", …

**Figure 1 *n*-gram examples from various disciplines**

| Field | Unit | Sample sequence | 1-gram sequence | 2-gram sequence | 3-gram sequence |
|---|---|---|---|---|---|
| Vernacular name | | | unigram | bigram | trigram |
| Order of resulting Markov model | | | 0 | 1 | 2 |
| Protein sequencing | amino acid | … Cys-Gly-Leu-Ser-Trp … | …, Cys, Gly, Leu, Ser, Trp, … | …, Cys-Gly, Gly-Leu, Leu-Ser, Ser-Trp, … | …, Cys-Gly-Leu, Gly-Leu-Ser, Leu-Ser-Trp, … |
| DNA sequencing | base pair | …AGCTTCGA… | …, A, G, C, T, T, C, G, A, … | …, AG, GC, CT, TT, TC, CG, GA, … | …, AGC, GCT, CTT, TTC, TCG, CGA, … |
| Computational linguistics | character | …to_be_or_not_to_be… | …, t, o, _, b, e, _, o, r, _, n, o, t, _, t, o, _, b, e, … | …, to, o_, _b, be, e_, _o, or, r_, _n, no, ot, t_, _t, to, o_, _b, be, … | …, to_, o_b, _be, be_, e_o, _or, or_, r_n, _no, not, ot_, t_t, _to, to_, o_b, _be, … |
| Computational linguistics | word | … to be or not to be … | …, to, be, or, not, to, be, … | …, to be, be or, or not, not to, to be, … | …, to be or, be or not, or not to, not to be, … |

# LANGUAGE MODELING

**Assign a probability to a sentence**

- Machine Translation:
  - P(**high** winds tonite) > P(**large** winds tonite)
- Spell Correction
  - The office is about fifteen **minuets** from my house
    - P(about fifteen **minutes** from) > P(about fifteen **minuets** from)
- Speech Recognition
  - P(I saw a van) >> P(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.)**

# SIMPLEST CASE: UNIGRAM MODEL

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i)$$

Some automatically generated sentences from a unigram model

```
fifth, an, of, futures, the, an, incorporated, a,
a, the, inflation, most, dollars, quarter, in, is,
mass

thrift, did, eighty, said, hard, 'm, july, bullish

that, or, limited, the
```

# BIGRAM MODEL

**Condition on the previous word:**

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-1})$$

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

# 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**

# MOVING ON …

**Words words words!**

- Free text and natural language processing in data science
- Bag of words and TF-IDF
- N-Grams and language models
- Information extraction & sentiment mining

**Thanks to Amol Deshpande (UMD)**



DuckDuckGo

# INFORMATION EXTRACTION (IE)

**Information extraction (IE) systems**

- Find and understand limited relevant parts of texts
- Gather information from many pieces of text
- Produce a structured representation of relevant information:
  - relations (in the database sense), a.k.a.,
  - a knowledge base
- Goals:
  - Organize information so that it is useful to people
  - Put information in a semantically precise form that allows further inferences to be made by computer algorithms

# INFORMATION EXTRACTION (IE)

**IE systems extract clear, factual information**

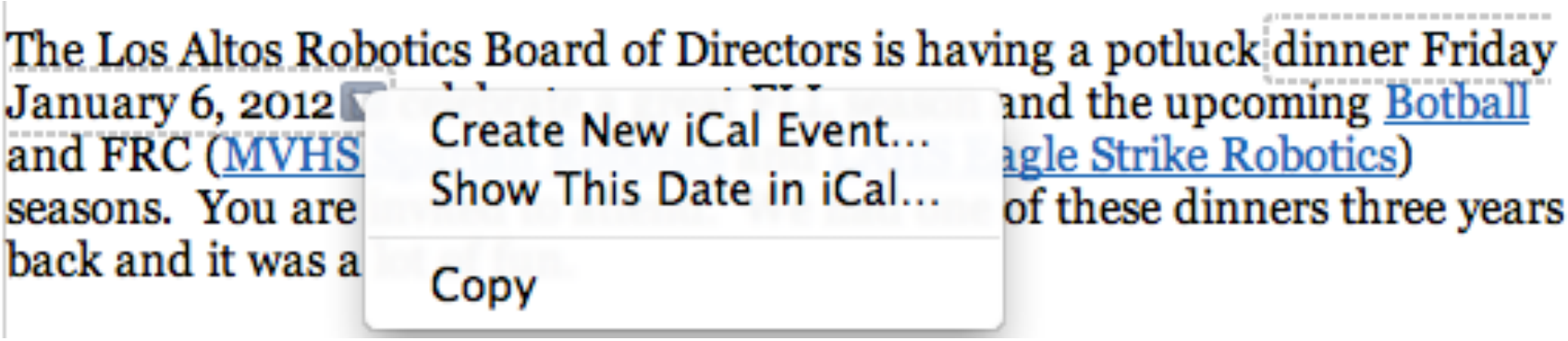
- Roughly: Who did what to whom when?

**E.g.,**

- Gathering earnings, profits, board members, headquarters, etc. from company reports
  - The headquarters of BHP Billiton Limited, and the global headquarters of the combined BHP Billiton Group, are located in Melbourne, Australia.
  - headquarters("BHP Biliton Limited", "Melbourne, Australia")
- Learn drug-gene product interactions from medical research literature

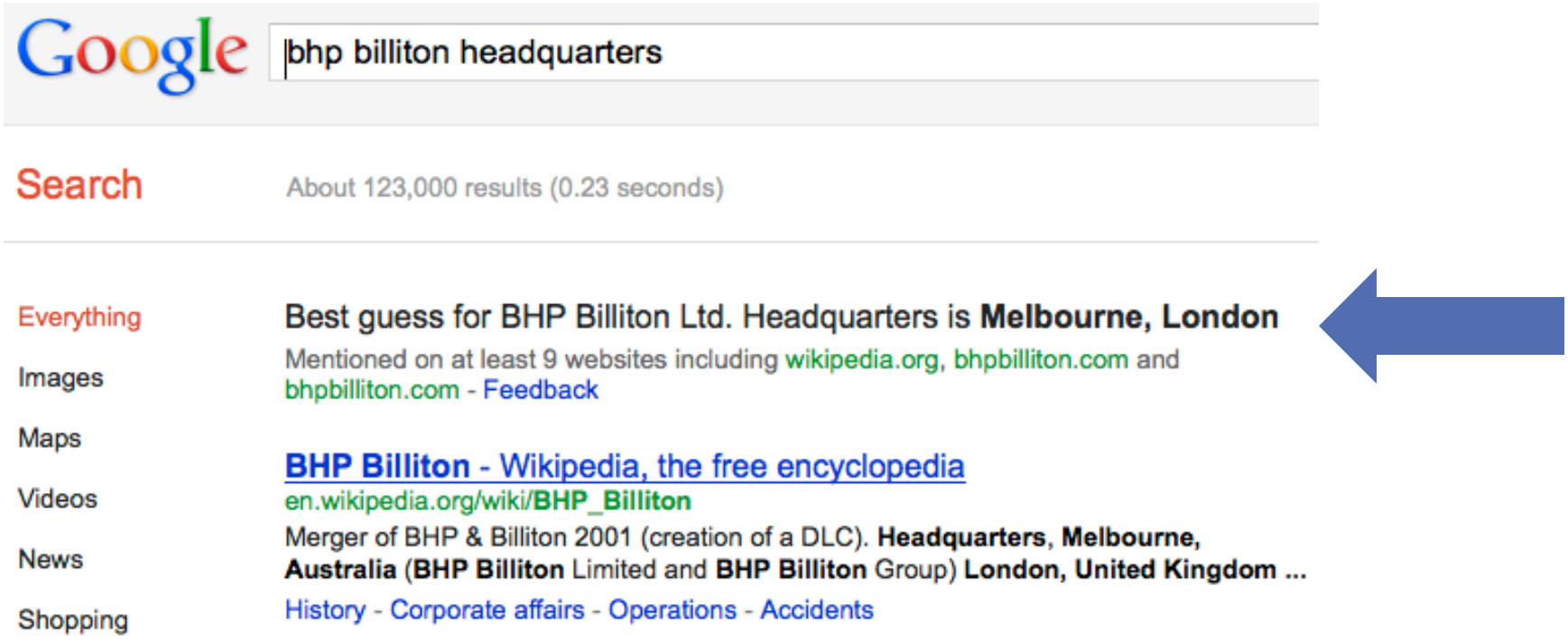
# LOW-LEVEL INFORMATION EXTRACTION

**Is now available and popular in applications like Apple or Google mail, and web indexing**




**Often seems to be based on regular expressions and name lists**

# LOW-LEVEL INFORMATION EXTRACTION

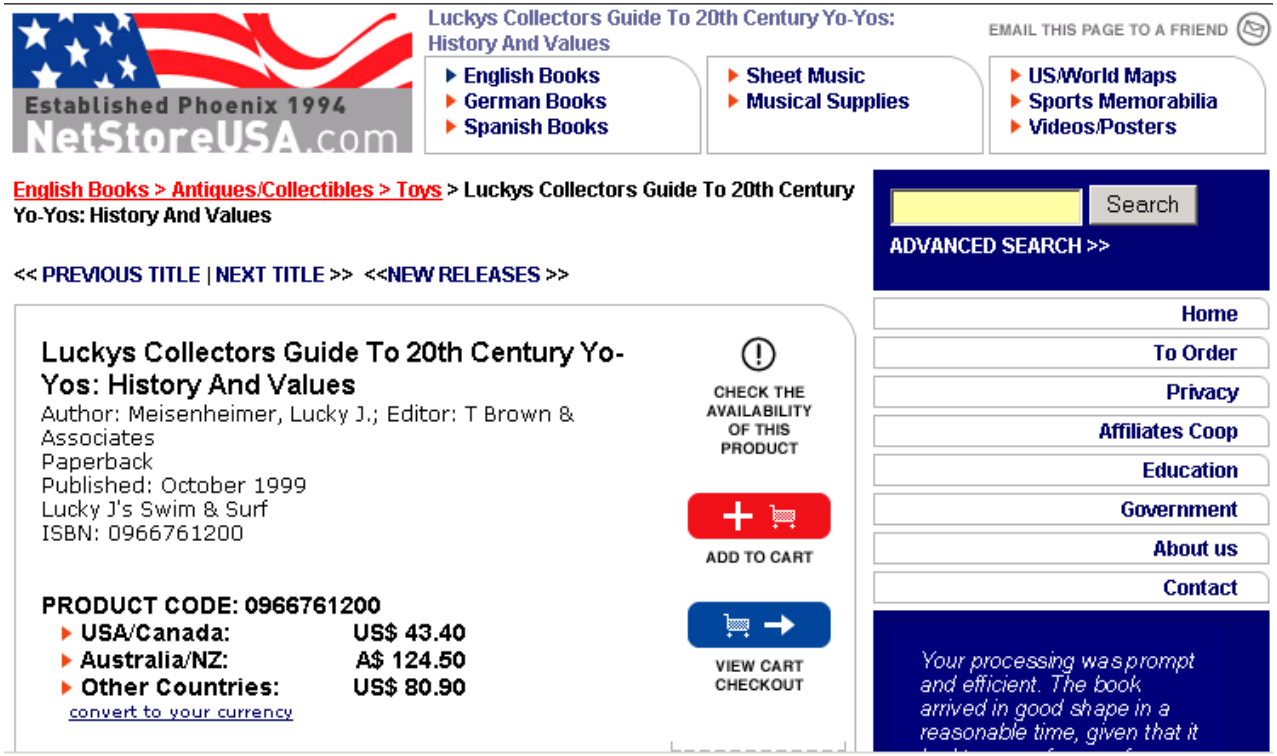


# WHY IS IE HARD ON THE WEB?

A book, Not a toy

Title

Need this price



# WHY DOESN'T TEXT SEARCH (IR) WORK?

**What you search for in real estate advertisements:**

**Town/suburb. You might think easy, but:**

- Real estate agents: Coldwell Banker, Mosman
- Phrases: Only 45 minutes from Parramatta
- Multiple property ads have different suburbs in one ad

**Money: want a range not a textual match**

- Multiple amounts: was $155K, now $145K
- Variations: offers in the high 700s [but not rents for $270]

**Bedrooms: similar issues: br, bdr, beds, B/R**

# NAMED ENTITY RECOGNITION (NER)

**A very important sub-task: find and classify names in text**

In 1917, Einstein applied the general theory of relativity to model the large-scale structure of the universe. He was visiting the United States when Adolf Hitler came to power in 1933 and did not go back to Germany, where he had been a professor at the Berlin Academy of Sciences. He settled in the U.S., becoming an American citizen in 1940. On the eve of World War II, he endorsed a letter to President Franklin D. Roosevelt alerting him to the potential development of "extremely powerful bombs of a new type" and recommending that the U.S. begin similar research. This eventually led to what would become the Manhattan Project. Einstein supported defending the Allied forces, but largely denounced using the new discovery of nuclear fission as a weapon. Later, with the British philosopher Bertrand Russell, Einstein signed the Russell–Einstein Manifesto, which highlighted the danger of nuclear weapons. Einstein was affiliated with the Institute for Advanced Study in Princeton, New Jersey, until his death in 1955.

Tag colours:

LOCATION TIME PERSON ORGANIZATION MONEY PERCENT DATE

# NAMED ENTITY RECOGNITION (NER)

**The uses:**

- Named entities can be indexed, linked off, etc.
- Sentiment can be attributed to companies or products
- A lot of IE relations are associations between named entities
- For question answering, answers are often named entities.

**Concretely:**

- Many web pages tag various entities, with links to bio or topic pages, etc.
  - Reuters' OpenCalais, Evri, AlchemyAPI, Yahoo's Term Extraction, …
- Apple/Google/Microsoft/… smart recognizers for document content