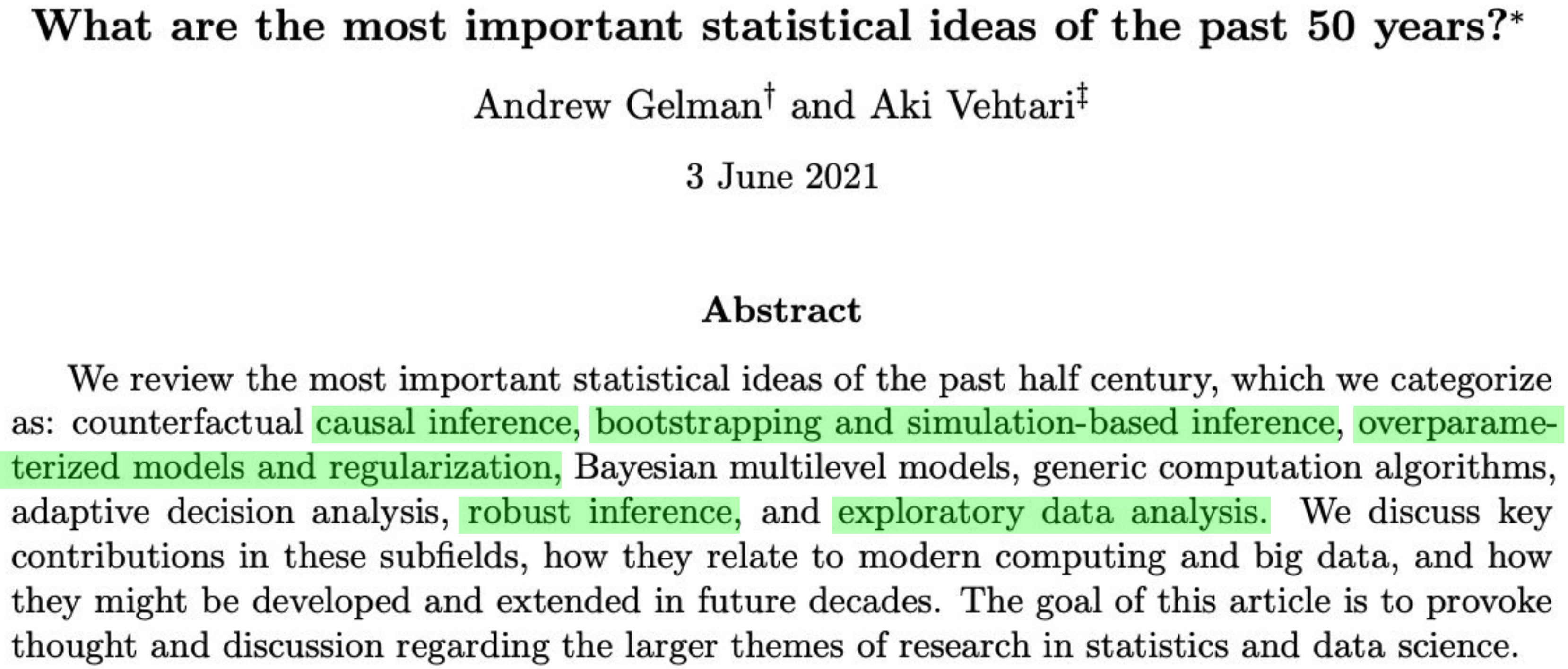
Applied Data Analysis (CS401)

Lectures 10+11
Handling text data

22 Nov 2023
+ 29 Nov 2023

Robert West

Looking back at ADA so far…



What are the most important statistical ideas of the past 50 years?[*]

Andrew Gelman[†] and Aki Vehtari[‡]

3 June 2021

Abstract

We review the most important statistical ideas of the past half century, which we categorize as: counterfactual causal inference, bootstrapping and simulation-based inference, overparameterized models and regularization, Bayesian multilevel models, generic computation algorithms, adaptive decision analysis, robust inference, and exploratory data analysis. We discuss key contributions in these subfields, how they relate to modern computing and big data, and how they might be developed and extended in future decades. The goal of this article is to provoke thought and discussion regarding the larger themes of research in statistics and data science.

https://arxiv.org/abs/2012.00174

Announcements

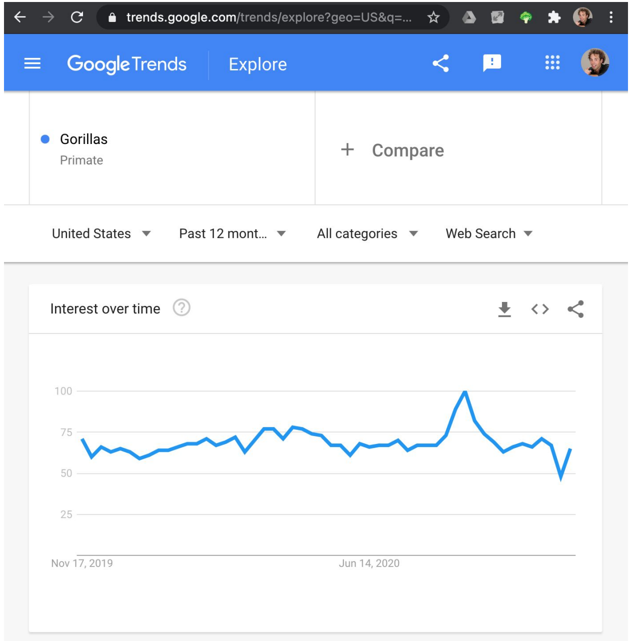
- To all ADAmericans: Happy Thanksgiving!
- Milestone P2 being graded; feedback to be released next week
- Homework H2 due on Fri 1 Dec
- Friday's lab session:
  - Quiz 9
  - Homework H2 office hours (in person in BCH 2201, **not** on Zoom!)
  - Alumnus Niccolò Stefanini (ML scientist @ Expedia) will give "report from the trenches" (a.k.a. the real world of data science)

Give us feedback on this lecture here:
https://go.epfl.ch/ada2023-lec10-feedback

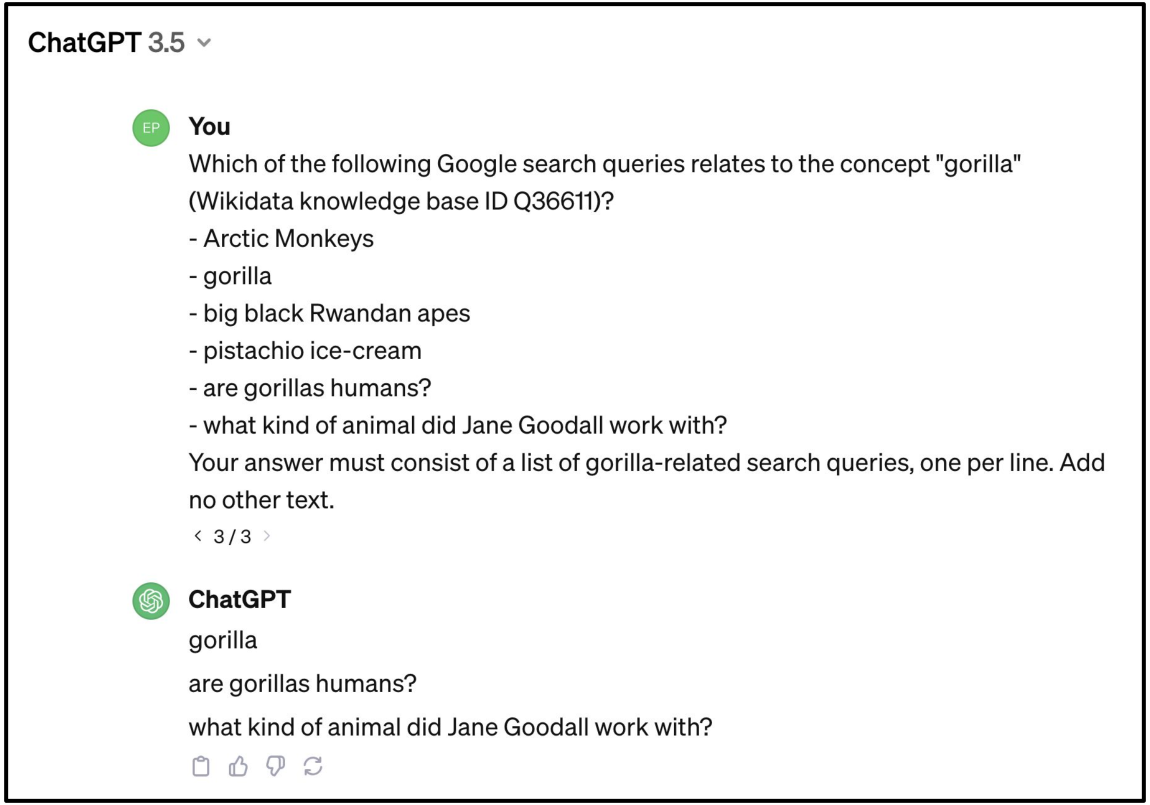
- What did you (not) like about this lecture?
- What was (not) well explained?
- On what would you like more (fewer) details?
- …

Textual data

- Much modern data is unstructured text
  - Web
  - Social media
  - News
  - Several ADA project datasets…
- Frequently, "clean" datasets can be derived from "dirty" textual data
  - e.g., search queries are short texts;
    Google Trends time series for concepts
    (e.g., Q36611 *Gorilla*) are obtained by aggregating
    all search queries referring to the concept
    (e.g., "gorilla", "big black Rwandan apes", "are gorillas humans?")



Nov 2022: the dawn of a new era



You

Which of the following Google search queries relates to the concept "gorilla" (Wikidata knowledge base ID Q36611)?

- Arctic Monkeys
- gorilla
- big black Rwandan apes
- pistachio ice-cream
- are gorillas humans?
- what kind of animal did Jane Goodall work with?

Your answer must consist of a list of gorilla-related search queries, one per line. Add no other text.

ChatGPT

gorilla

are gorillas humans?

what kind of animal did Jane Goodall work with?

Outline

- 4 typical tasks on text data:
  - Document retrieval
  - Document classification
  - Sentiment analysis
  - Topic detection
- How to phrase these tasks as machine learning problems
- How to preprocess text so it can be fed to ML algorithms
- Next lecture: pointers to miscellaneous more advanced topics
- ADA spirit: show you what’s there; give you basic feel
  - For more, take classes on NLP and information retrieval

Typical task 1: document retrieval

- Given:
  - Document collection (a.k.a. corpus)
  - Query document (can be short query string)
- Task:
  - Rank all docs in collection by similarity to query
- An old problem (e.g., libraries)
- Document retrieval is the core task solved by Web search engines (“10 blue links”)

Document retrieval

- Straightforward approach: neighbor search (as in kNN)
- Define a distance function between documents
- Given query *q*, find the *k* docs with smallest distance to *q*
- *k* = 10, docs sorted by distance, blue links, ads ⟶ G
- The hard part: craft/learn a distance function (and scale it to the Web…)

Typical task 2: document classification

- Given:
  - Document *d*
  - Set of classes (e.g., topics: news, sports, tech, music, romance)
- Task:
  - Decide to which one of the classes document *d* belongs
- Example scenario:
  - Find gangster movies in CMU Movie Summary Corpus

Document classification

- Supervised learning
- Obtain a large collection of documents
- Label each doc with the class it belongs to
- Represent docs as feature vectors
- Train a supervised classifier based on the labeled docs:
    - e.g., kNN, logistic regression, decision tree, random forest, boosted decision trees, neural network, …

Typical task 3: sentiment analysis

- Given:
    - Document *d* (e.g., product review)
- Task:
    - “Sentiment” score capturing how positive/negative *d* is
- Example scenarios:
    - Infer what people think about a product from text only (i.e., without explicitly given ratings)
    - Historical opinion analysis; e.g., how has people’s attitude toward certain politicians changed over time?

Sentiment analysis

- Supervised learning
  - Regression
  - Classification
- Same setup as for document classification:
  - Label a training set with ground-truth sentiment scores
  - Represent documents as feature vectors
  - Train supervised model: kNN, linear/logistic regression, …

Typical task 4: topic detection

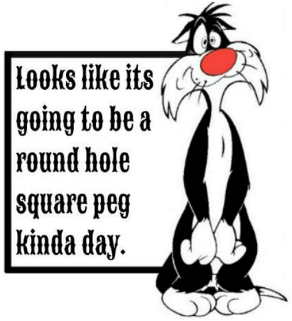
- Given:
  - Unlabeled document collection
- Task:
  - Determine a set of prevalent topics in the docs
  - Determine for each document to which topics it belongs
- Example scenario:
  - Detection of trending topics in social media (e.g., Twitter)
  - Detection of distinct viewpoints on a political subject
  - Exploratory analysis of a large doc collection

Topic detection

- Clustering
- Represent documents as feature vectors
- Run hierarchical or point-assignment clustering algorithm
  - Hierarchical: agglomerative or divisive
  - Point-assignment: e.g., k-means, DBSCAN
- Alternative: matrix factorization (cf. next lecture)

Feature vectors

- Nearly all ML methods work with feature vectors
  - E.g., previous slides: document retrieval; document classification; sentiment analysis; topic detection
- Text is not immediately a feature vector
  - Variable length
  - Even for fixed length (e.g., tweet…): Positions don't correspond to meaningful features


Looks like its going to be a round hole square peg kinda day.

Feature vectors

- Need to transform arbitrarily long string to fixed-length vector
  - Traditional and vetted: bag of words
  - More recent: *learn* a mapping from strings to vectors (buzzword: “text embedding”)

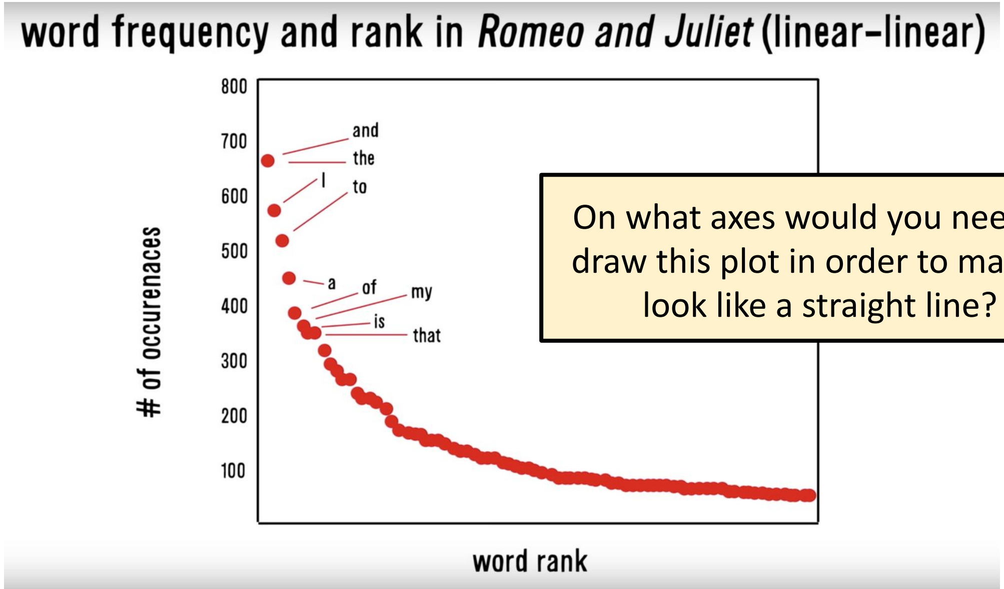
Bag of words

Tom Mitchell (CMU)

- Bag == multiset
  - "multi-": keep multiplicity of words
  - "-set": don't keep order of words
  - E.g., document "what you see is what you get"
    → bag of words {get:1, is:1, see:1, what:2, you:2}
- To have fixed-length representation for all documents:
  - Vector with one entry for each unique word in vocabulary
  - Bag-of-word vectors are very high-dimensional (typically 1e5 or 1e6) and very sparse
  - E.g., above: [0…0 1 0…0 1 0…0 1 0…0 2 0…0 2 0…0]

An extra reason for sparsity: Zipf’s law

A famous power law



On what axes would you need to draw this plot in order to make it look like a straight line?

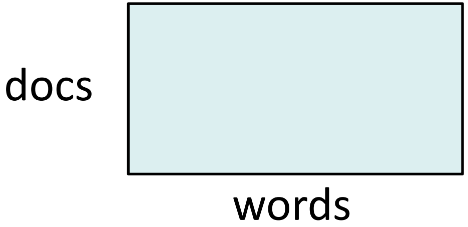
The probability of observing a word scales inversely with its frequency rank:

$p(w_i) \propto 1/i$ (where w_i is the i-th most frequent word)

POLLING TIME

- Scan QR code or go to https://web.speakup.info/room/join/66626

Bag-of-words matrix



- Combine document vectors as rows in a matrix
  - One row per doc
  - One column per word in vocabulary
- This matrix is huge!
  - E.g., Wikipedia: 6M docs, 2M words → 12 trillion entries
- Solution: use a sparse matrix format
  - Triples: (doc_idx, word_idx, count)
  - E.g., Wikipedia, assuming 2000 words per article on avg.: 12 billion non-zero entries (fits in memory)
- With matrix representation, you're ready to use any ML model

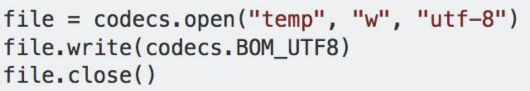
… or are you really?

- In theory, yes
- In practice: “garbage in, garbage out”
- Be careful when mapping raw text to bag-of-words matrix!
    - Character encoding
    - Language identification
    - Tokenization
    - Stopword removal
    - Word normalization
- Tweaking the matrix a bit can lead to much better performance
    - Reweight/normalize rows and/or columns of matrix

Bag of tricks for bags of words

Character encoding

- Mapping from (abstract) characters to bytes
- Old school: ASCII, Latin-1
- New school: Unicode (e.g., UTF-8, UTF-16, UTF-32)
- E.g., W → 0x57 (one byte)
- Reading text from file:
  - Need to read with encoding that was used to write file
  - Especially important for non-English text: à, ê, ü, ß, …
- Writing to file: Always use UTF-8 or UTF-16; hard-code the output format!



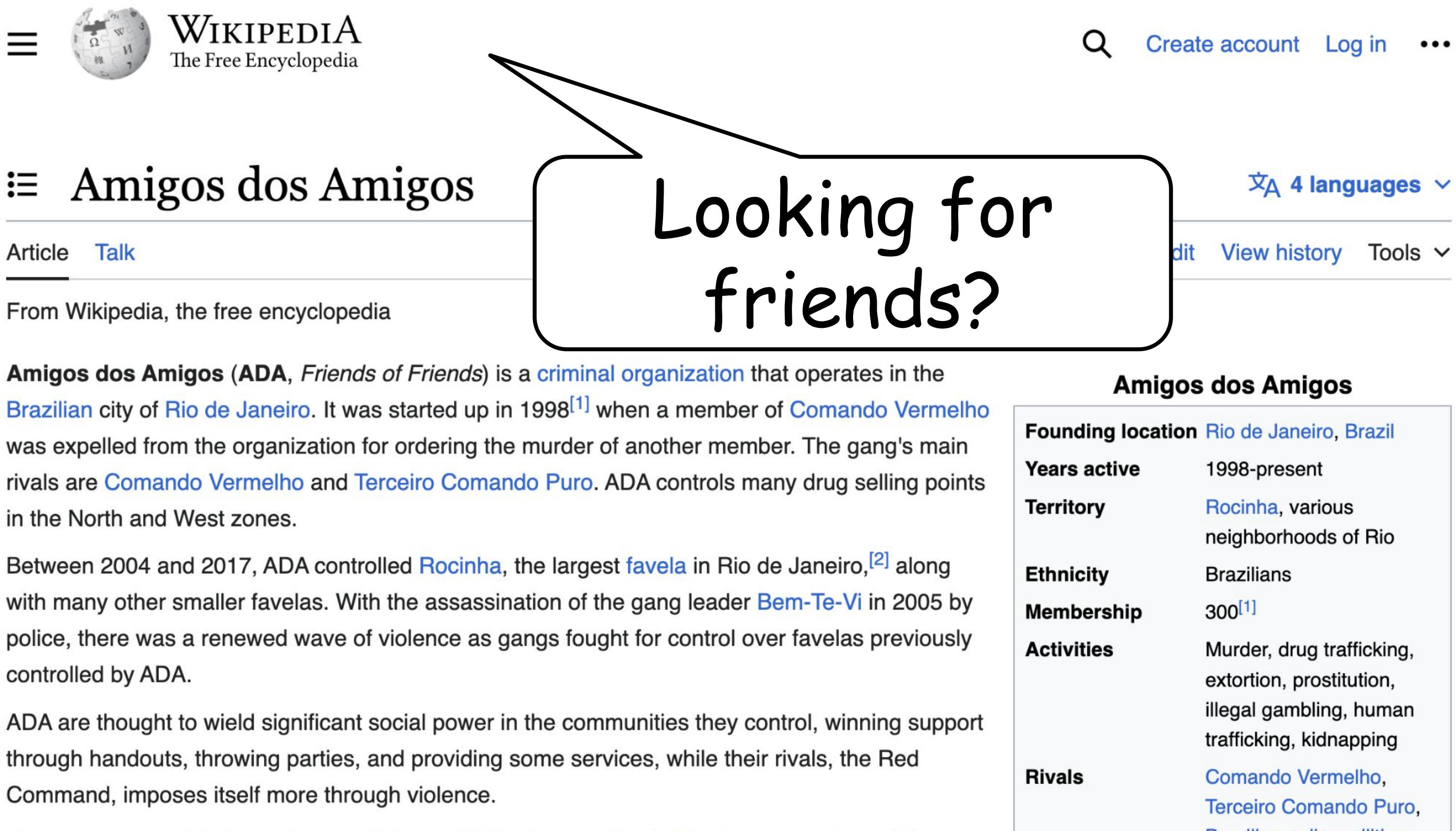
```
file = codecs.open("temp", "w", "utf-8")
file.write(codecs.BOM_UTF8)
file.close()
```

- Otherwise, your future self will be very angry at you (example in speaker notes 👇)

Language identification

- Typically, you're interested in text from a single language
- Increasingly, content is multilingual (e.g., Twitter, Wikipedia)
- Ideally, language code is specified (e.g., headers in HTML; JSON field in Twitter API results)
- But not always…
- There are good libraries (e.g., Python, Java)
  - Most commonly based on letter trigrams (e.g., "eau", "ghi", "ijs", "sch", "eiß", "ção")
  - Much harder if you messed up character encoding…

Commercial break



Amigos dos Amigos

From Wikipedia, the free encyclopedia

Amigos dos Amigos (**ADA**, *Friends of Friends*) is a criminal organization that operates in the Brazilian city of Rio de Janeiro. It was started up in 1998[1] when a member of Comando Vermelho was expelled from the organization for ordering the murder of another member. The gang's main rivals are Comando Vermelho and Terceiro Comando Puro. ADA controls many drug selling points in the North and West zones.

Between 2004 and 2017, ADA controlled Rocinha, the largest favela in Rio de Janeiro,[2] along with many other smaller favelas. With the assassination of the gang leader Bem-Te-Vi in 2005 by police, there was a renewed wave of violence as gangs fought for control over favelas previously controlled by ADA.

ADA are thought to wield significant social power in the communities they control, winning support through handouts, throwing parties, and providing some services, while their rivals, the Red Command, imposes itself more through violence.

Amigos dos Amigos

Founding location	Rio de Janeiro, Brazil
Years active	1998-present
Territory	Rocinha, various neighborhoods of Rio
Ethnicity	Brazilians
Membership	300[1]
Activities	Murder, drug trafficking, extortion, prostitution, illegal gambling, human trafficking, kidnapping
Rivals	Comando Vermelho, Terceiro Comando Puro,

Tokenization

- Maps character string into sequence of tokens (≈ words)
- E.g., “Hello! How are you?” ⟶ Hello_!_How_are_you_?
- Tempting to do this yourself by splitting at whitespaces and punctuation
- But many corner cases:
  - “Hello, Mr. President! How are you?! :-)”
    ⟶ Hello_,_Mr._President_!_How_are_you_?!_:-)
- Don’t do it yourself, use libraries instead; e.g.,
  - Python: spaCy, nltk; Java: Stanford CoreNLP
  - Rule-based, deterministic, fast

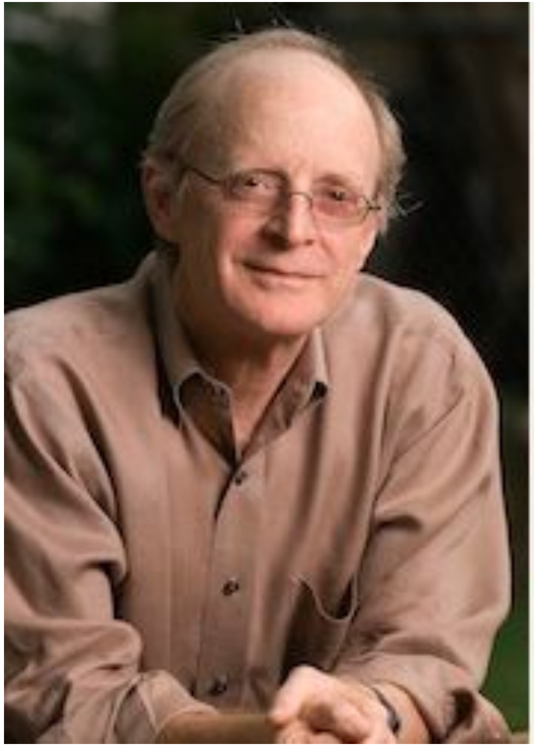
Tokenization

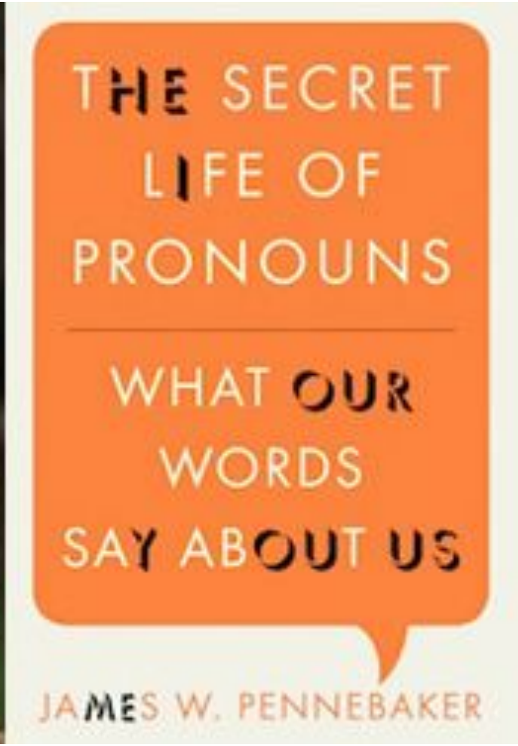
- Optimal tokenizer different for different languages (e.g., Swedish “Sankt Peter” → “S:t Peter”), but English tokenizer often good enough
- Tokenization relatively straightforward in English
- Hard in, e.g., Chinese: no whitespace between words
- Compound words, e.g. in German:
    - Advanced models can split “Donaudampfschifffahrtskapitän” into “Donau dampf schiff fahrts kapitän”
    - But what to keep together…? “Schiff fahrt” or “Schifffahrt”?

Stopword removal

- Very frequent, “small” words carry little information for most tasks and can “drown out” information contained in real content words
- E.g., “a”, “the”, “is”, “you”, “I”, punctuation marks
- Many stopword lists online, but be careful!
    - Different tasks require removing different stopwords
    - Good heuristic: remove words appearing in at least *p*% of all documents (but what should *p* be…?)
    - Sometimes stopword removal hurts!
        - Author identification, psychological modeling; punctuation can be useful as well: e.g., “!!!”, “:-)”

Don’t throw out the baby with the bathwater!





Word normalization: casefolding

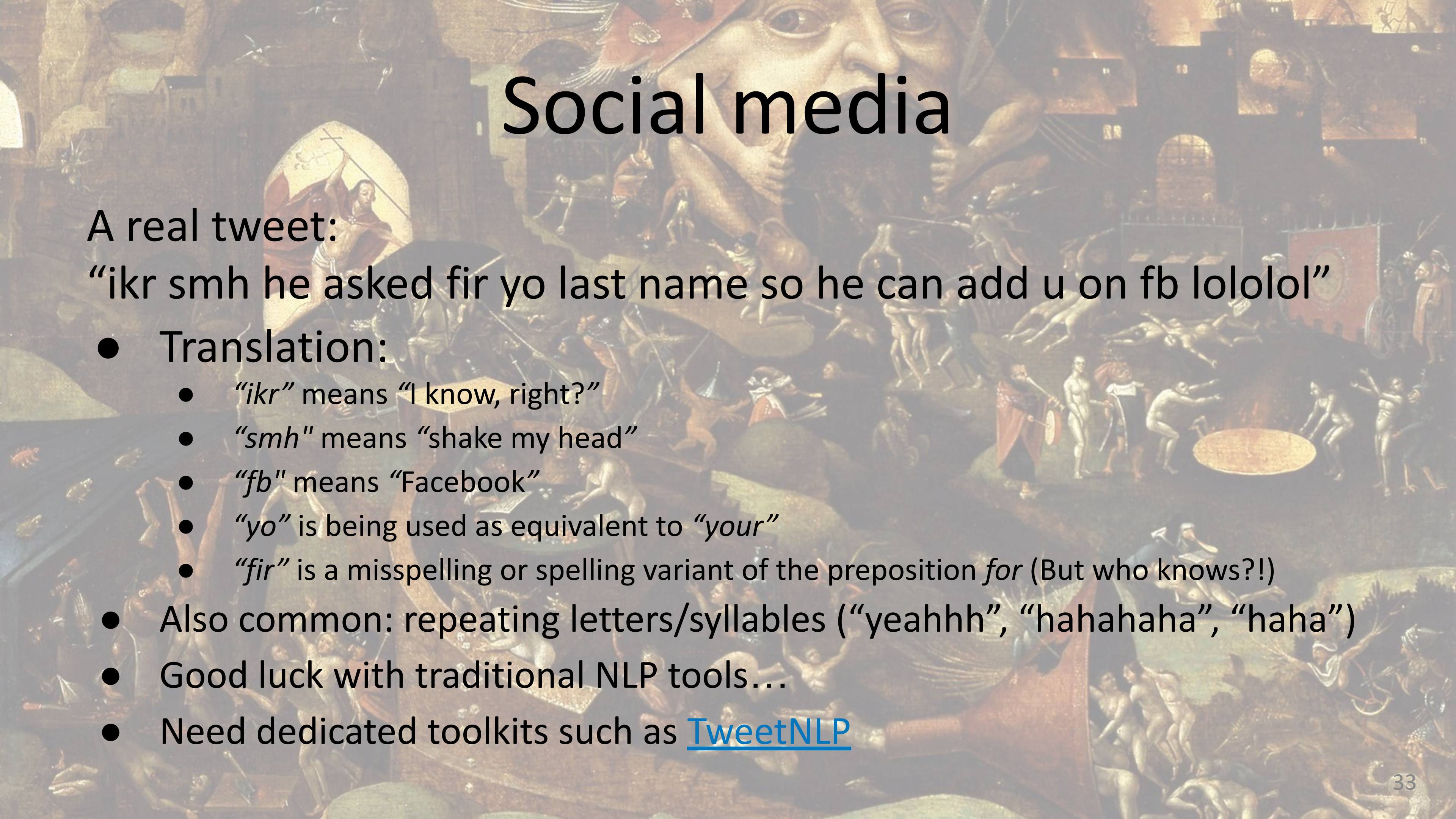
- E.g., “I love yams. Yams are yummy.”
- Should “yams” and “Yams” really be different features?
- Simple solution: make everything lower-case (“casefolding”)
- But then: “I’d rather have an apple than an Apple.”
- Hand-code exceptions?
- In practice (especially when dataset is large), typically best to **not** do casefolding
- But when dataset is small, might help because less sparsity

Word normalization: Stemming

- Map different forms of same word to same, normalized form, by stripping affixes
- E.g., “walking”, “walks”, “walked” → “walk”
  “business”, “busy” → “busi”
- Typically done in hacky, heuristic way (e.g., Porter stemmer)
- Pro: decreases sparsity in bag-of-words matrix
- Con: discards information
  - E.g., “business” vs. “busy”; “operating” (as in “operating system”)
- In English (esp. with big data) typically not done anymore
- Still very useful in morphologically richer languages (e.g., German, Finnish, Bantu languages)

Word normalization: Lemmatization

- Lemmatization == stemming++
- Map tokens to lexicon entries
- E.g., “U.S.A.”, “US” → “United States”
  “Grüße”, “Gruesse” → “Grüße”
  “You **lie** in the grass” vs. “You **lie** to me”
- Frequently omitted, as it requires complete lexicon and complex mapping rules
- Especially hard for non-English



Social media

A real tweet:
“ikr smh he asked fir yo last name so he can add u on fb lololol”

- Translation:
  - *“ikr”* means “I know, right?”
  - *“smh"* means “shake my head”
  - *“fb"* means “Facebook”
  - *“yo”* is being used as equivalent to *“your”*
  - *“fir”* is a misspelling or spelling variant of the preposition *for* (But who knows?!)
- Also common: repeating letters/syllables (“yeahhh”, “hahahaha”, “haha”)
- Good luck with traditional NLP tools…
- Need dedicated toolkits such as TweetNLP

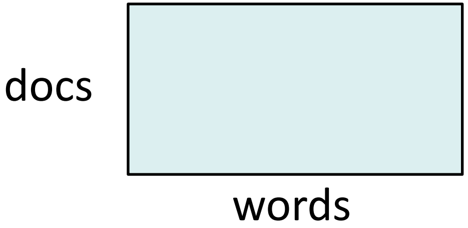
Tokens vs. n-grams

- So far: bag-of-words matrix
  - Rows: documents
  - Columns: tokens (a.k.a. unigrams, or 1-grams)
- Frequently, longer sequences belong together
  - E.g., “United States”, “operating system”
- Brute-force approach: use $n > 1$
  - E.g., all bigrams ($n = 2$), all trigrams ($n = 3$)
  - Using all 5-grams can beat neural networks (Table 1 here)
  - Problem: combinatorial explosion

Tokens vs. n-grams

- Smarter:
  - Feature selection ("multi-word expressions", "phrase extraction")
  - Simple approach for bigrams: keep bigram if *mutual information* between constituent tokens is large
  - How to generalize to $n > 2$?
    - Frequent itemset/sequence mining
    - Wikipedia anchor texts
    - Compressive feature learning (link)

Postprocessing the BOW matrix



docs

words

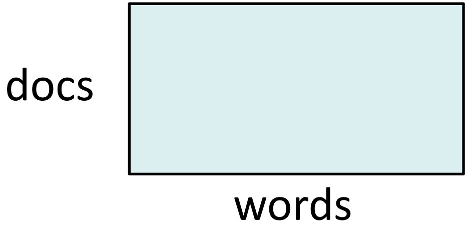
Inverse document frequency

- Not all words equally informative
- This is the reason for removing stopwords (“a”, “the”, “is”, …)
- Beyond discarding stopwords, want to give less weight to more common words
    - E.g., “per” vs. “perceptron”
- Standard way: **IDF = inverse document frequency**
    - docfreq(*w*): number of documents that contain word *w*
    - *N*: overall number of documents
    - idf(*w*) = –log(docfreq(*w*) / *N*) = log(*N*) – log(docfreq(*w*))

Inverse document frequency

- idf(*w*) = –log(docfreq(*w*) / *N*)
- Interpretation: information content (in terms of #bits) of event “randomly drawing a document that contains *w*”
- Beyond this theoretical justification, IDF weighting has been shown to work well in practice

TF-IDF matrix



- tf(*d*, *w*): term frequency of word *w* in doc *d*
  - This is what the bag of words captures
  - E.g., document “what you see is what you get”
    → bag of words {get:1, is:1, see:1, what:2, you:2}
- idf(*w*): inverse doc freq of *w* (computed on entire corpus)
- TF-IDF matrix:
  - Entry in row *d* and column *w* has value
    tf(*d*, *w*) * idf(*w*)
  - Amounts to multiplying column *w* with constant idf(*w*)

Row normalization of TF-IDF matrix

- Longer docs have more non-zero entries
- Interpreted as vectors, longer docs have longer vectors
- This may throw off ML algorithms
  - Long vectors far away from short vectors
  - Dot product: random vector has higher dot product with longer vector
- Fix: normalize doc vectors, i.e., rows of TF-IDF matrix
  - L2-normalization: all rows have Euclidean distance 1 from origin (all data points lie on a unit sphere)
  - L1-normalization: all rows sum to 1, i.e., can be interpreted as distribution
- How to know which one is better?

Column normalization

- IDF-scaling may be seen as column normalization
- Additionally, it may help to apply any of the normalization techniques we discussed in lecture 8 (“Applied ML”)
  - Min-max scale
  - Standardize: subtract mean; divide by standard deviation
- How to know which one (if any) to use?

Bag of tricks for bags of words

Stay tuned!

Next week: Part 2

Applied Data Analysis (CS401)

ada

Lectures 10+11

Handling text data

22 Nov 2023

+ 29 Nov 2023

EPFL

Robert West

dlab

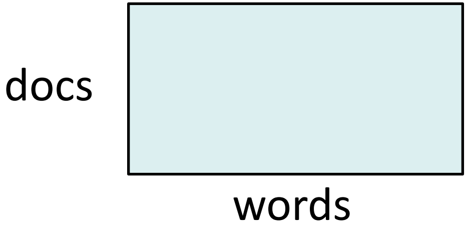
Announcements

- Homework H2 due this Fri 1 Dec 23:59
    - Reminder: We won’t answer questions asked during final 24h
- Projects:
    - Milestone P2 grades have been released
    - Start working on Milestone P3 after Friday!
- Friday’s lab session:
    - Exercises on handling text

Recap

Let me open my **bag of tricks for bags of words** for you! But only if you were good children...

Reminder: bag-of-words matrix

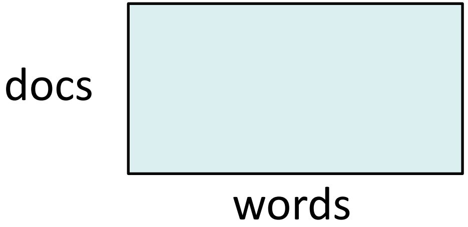


docs

words

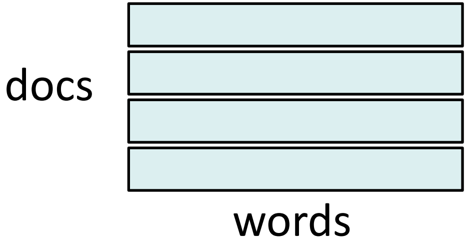
Revisiting the 4 typical tasks

- Document retrieval
- Document classification
- Sentiment analysis
- Topic detection



- TF-IDF matrix to the rescue
  - Entry for doc d, word w:
    tf(d, w) * idf(w)

Typical task 1: document retrieval



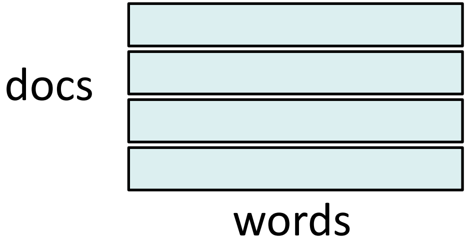
- Nearest-neighbor method in spirit of kNN
- Compare query doc *q* to all documents in the collection (i.e., rows of the TF-IDF matrix)
- Rank docs from collection in increasing order of distance
- Distance metrics
  - Typically cosine distance (= 1 – cosine similarity)
  - Recall: cosine similarity of *q* and *v* = <*q*/|*q*|, *v*/|*v*|>
  - If rows are L2-normalized, may simply take dot products <*q*, *v*>

Typical task 1: document retrieval

- This is just the most basic approach
- For efficiency
  - Start by filtering documents by presence of query terms (use efficient full-text index)
  - Hugely narrows down set of documents to be ranked
- Google et al. do much more…
  - Query-independent relevance: PageRank
  - Boost recent results
  - Personalization, contextualization
  - …

Typical task 2: document classification

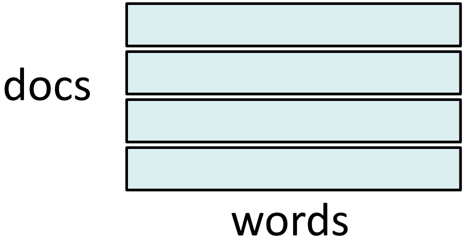
- Use TF-IDF matrix as feature matrix for supervised methods (cf. lecture 7)
- Often more features (words) than documents
- What’s the danger with this?
- High model capacity can lead to overfitting (high variance)
- Potential solutions:
  - Use more data (i.e., more labeled training docs)
  - Decrease model capacity:
    - Feature selection
    - Regularization (two slides from now)
    - Dimensionality reduction (a few slides from now)
  - Use ensemble methods such as random forests



Typical task 3: sentiment analysis

- When treated as classification:
  Ctrl-C Ctrl-V previous slide

- When treated as regression:
  Pretty much the same (most supervised methods work for both classification and regression)



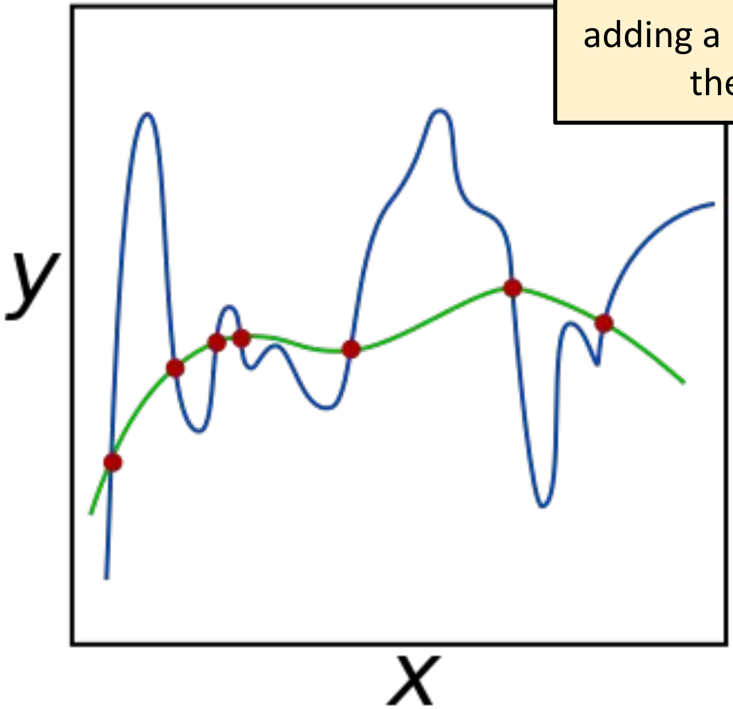
Regularization

- E.g., linear regression:
  Find weight vector $\boldsymbol{\beta}$ that minimizes $$\sum_{i=1}^{n} (y_i - \mathbf{x}_i^\top \beta)^2$$

  ($\mathbf{x}_i$: feature vector of *i*-th data point; y_i: label of *i*-th data point, e.g., sentiment [1–5 stars] expressed in document *i*)
- If one word *j* appears only in docs with sentiment 5, we can obtain very small training error on these docs by making β_j large enough
- But doesn't generalize to unseen test data!
- Remedy: penalize very large positive and very large negative weights:

$$\text{minimize} \quad \sum_{i=1}^{n} (y_i - \mathbf{x}_i^\top \beta)^2 + \boxed{\lambda \sum_{j=1}^{p} \beta_j^2}$$

Regularization

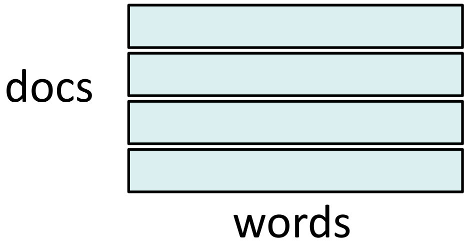


Which curve resulted from adding a regularization term to the loss function?

POLLING TIME

- Scan QR code or go to https://web.speakup.info/room/join/66626

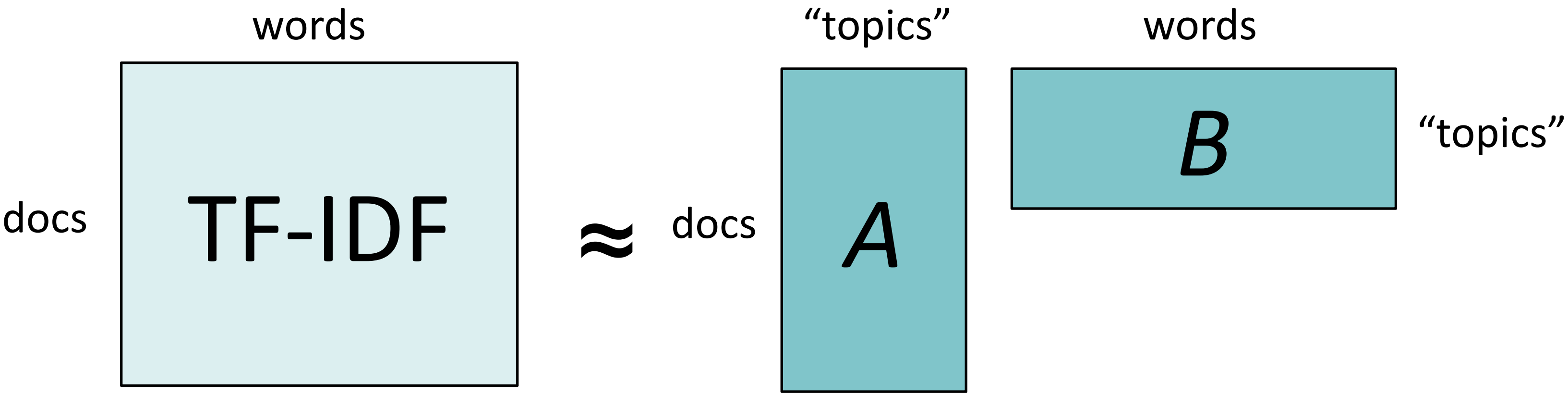
Typical task 4: topic detection



- Cluster rows of TF-IDF matrix (each row a data point)
- Manually inspect clusters and label them with descriptive names (e.g., “news”, “sports”, “romance”, “tech”, “politics”)
- In principle, may use k-means, k-medoids, etc.
- But can be difficult if dimensionality is large (#words ≫ #docs)
    - “Curse of dimensionality”
    - Many outliers

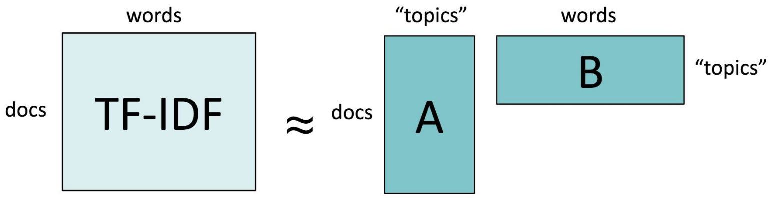
Typical task 4: topic detection

- Alternative approach: **matrix factorization**



- Assume docs and words have representation in (latent) "topic space"
- (IDF-weighted) word frequency modeled as dot product of doc's vectors and word's vectors in topic space
- #topics ≪ #words (→ "dimensionality reduction"): D*W → (D+W)*T
- Topics interpretable in doc space (*A*'s cols) and word space (*B*'s rows)

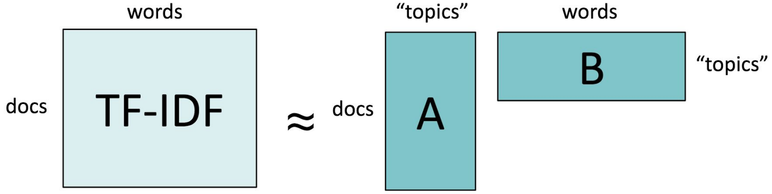
Typical task 4: topic detection



- Optimization problem:
  - Find *A*, *B* such that *AB* is as close to TF-IDF matrix as possible
  - That is, minimize $\sum_{d=1}^{N} \sum_{w=1}^{M} (T_{dw} - A_d \; B_w)^2$

    where *T* is TF-IDF matrix, A_d is *d*-th row of *A*, and B_w is *w*-th column of *B*
- This is called **latent semantic analysis (LSA)**

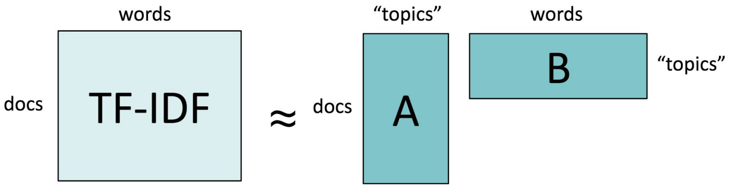
Typical task 4: topic detection



You already know how to efficiently compute this, from your linear algebra class: **singular-value decomposition** (SVD)

- $T = USV^{\mathrm{T}}$
- Freebie: columns of *U* and *V* are orthonormal bases (yay!)
- *S* is diagonal (with values in decreasing order) and captures "importance" of topic (amount of variation in corpus w.r.t. topic)
- If you want *k* topics, keep only the first *k* columns of *U* and *V*, and the first *k* rows and columns of *S*
  $\rightarrow U', S', V'$
- E.g., $A = U', B = S'V'^{\mathrm{T}}$ or $A = U'S', B = V'^{\mathrm{T}}$

Typical task 4: topic detection



- Recall potential problem with clustering and classification and regression: "curse of dimensionality"
- Matrix factorization via LSA solves these problems for you:
  - Use *A* instead of original TF-IDF matrix
  - That is, cluster (or learn to classify or regress) in topic space, rather than word space

- Topic representation from LSA is simply a vector, not a probability distribution over topics
- Probabilistic: LDA = Latent Dirichlet Allocation (p.t.o.)

Commercial break

ada

ça roule pour moi

LDA: probabilistic topic modeling

- **L**atent **D**irichlet **A**llocation (*not* Latent Discriminant Analysis!)
- Document := bag of words
- Topic := probability distribution over words
- Each document has a (latent) distribution over topics
- "Generative story" for generating a doc of length *n*:

```
$d$ := sample a topic distribution for the doc (⟵ "Dirichlet")
for $i = 1, \ldots, n$
    $t$ := sample a topic from topic distribution $d$
    $w$ := sample a word from topic $t$
    Add $w$ to the bag of words of the doc to be generated
```

Topics

gene	0.04
dna	0.02
genetic	0.01
. . .	

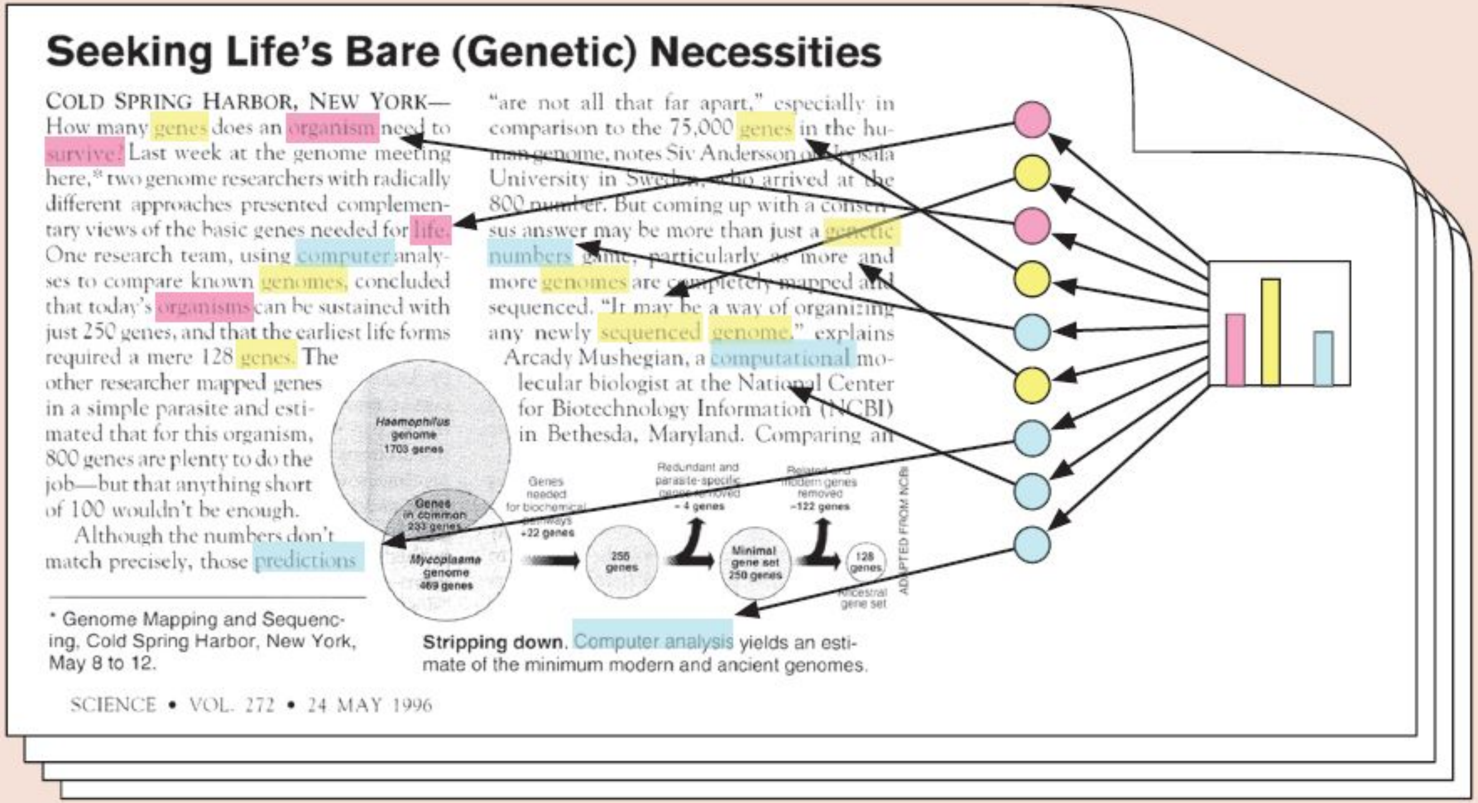
life	0.02
evolve	0.01
organism	0.01
. . .	

brain	0.04
neuron	0.02
nerve	0.01
. . .	

data	0.02
number	0.02
computer	0.01
. . .	

Documents

Topic proportions and assignments



Seeking Life's Bare (Genetic) Necessities

COLD SPRING HARBOR, NEW YORK— How many genes does an organism need to survive? Last week at the genome meeting here,* two genome researchers with radically different approaches presented complementary views of the basic genes needed for life. One research team, using computer analyses to compare known genomes, concluded that today's organisms can be sustained with just 250 genes, and that the earliest life forms required a mere 128 genes. The other researcher mapped genes in a simple parasite and estimated that for this organism, 800 genes are plenty to do the job—but that anything short of 100 wouldn't be enough.

Although the numbers don't match precisely, those predictions "are not all that far apart," especially in comparison to the 75,000 genes in the human genome, notes Siv Andersson of Uppsala University in Sweden, who arrived at the 800 number. But coming up with a consensus answer may be more than just a genetic numbers game, particularly as more and more genomes are completely mapped and sequenced. "It may be a way of organizing any newly sequenced genome," explains Arcady Mushegian, a computational molecular biologist at the National Center for Biotechnology Information (NCBI) in Bethesda, Maryland. Comparing an

Haemophilus genome 1703 genes
Genes in common 233 genes
Mycoplasma genome 469 genes
Genes needed for biochemical pathways +22 genes
256 genes
Redundant and parasite-specific genes removed −4 genes
Minimal gene set 250 genes
Related and modern genes removed −122 genes
128 genes
Ancestral gene set
ADAPTED FROM NCBI

Stripping down. Computer analysis yields an estimate of the minimum modern and ancient genomes.

* Genome Mapping and Sequencing, Cold Spring Harbor, New York, May 8 to 12.

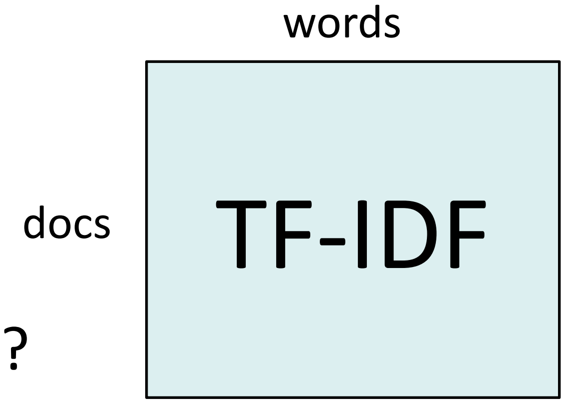
SCIENCE • VOL. 272 • 24 MAY 1996

Topic inference in LDA

- LDA is unsupervised (topics come out “magically”)
- Input:
  - Docs represented as bags of words
  - Number *K* of topics
- Output:
  - *K* topics (distributions over words)
  - For each doc: distribution over *K* topics
- How is this done?
  - Find distributions (i.e., topics, docs) that maximize the likelihood of the observed documents (maximum likelihood)

Question:

- “Which of these word pairs is more closely related?”
  - (car, bus)
  - (car, astronaut)
- How to quantify this?
- Detour:
  - How to quantify closeness of two docs?
  - E.g., cosine of **rows** of TF-IDF matrix
- Retour:
  - How to quantify closeness of two words?
  - E.g., cosine of **cols** of TF-IDF matrix



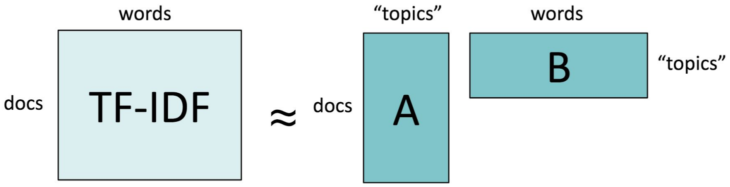
words

docs

TF-IDF

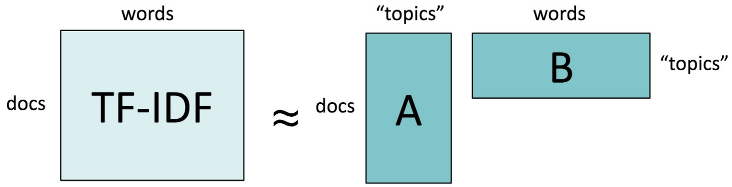
Sparsity in TF-IDF matrix

- Two docs (i.e., rows of TF-IDF matrix)
  - “Do you love men?”
  - “Adorest thou the likes of Adam?”
  - Cosine of row vectors of TF-IDF matrix == 0
- Same problem when comparing two words (i.e., cols of matrix)
- Solution:
  - Move from sparse to dense vectors
  - But how?
  - Latent semantic analysis (LSA)!
    - Use columns of *B* as dense vectors representing words



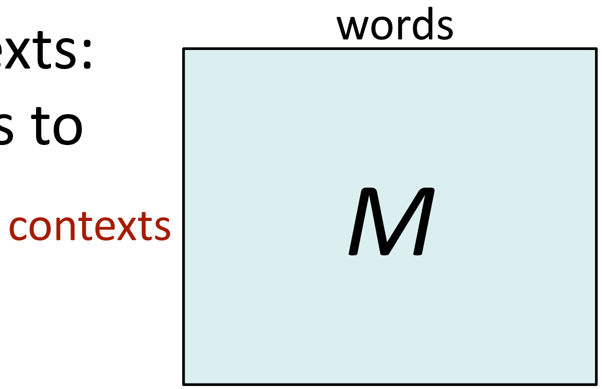
“Word vectors”

- Columns of TF-IDF matrix (sparse) or of word-by-topic matrix B (dense)



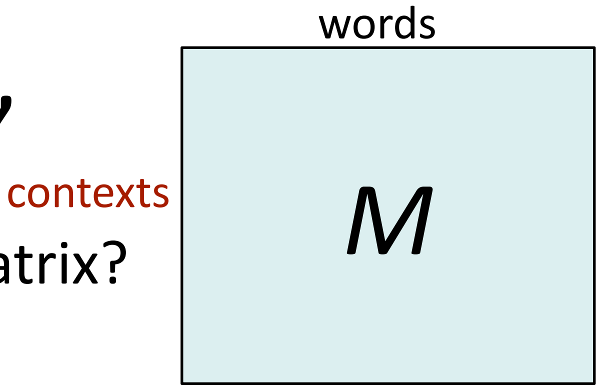
words
docs
TF-IDF
≈
“topics”
docs
A
words
B
“topics”

- Problem:
  - Entire doc treated as one bag of words
  - All information about word proximity, syntax, etc., is lost
- Solution:
  - Instead of full docs, consider local contexts: windows of *L* (e.g., 3) consecutive words to left and right of the target word
  - Rows of matrix: not docs, but contexts

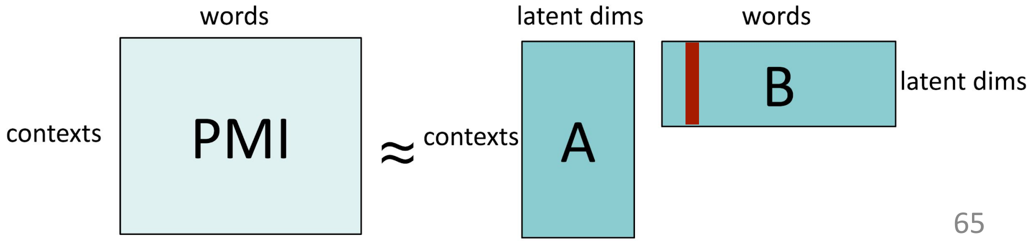


words
contexts
M

“Word vectors”



- What to use as entries of word/context matrix?
- Straightforward: same as TF-IDF, but with contexts as “pseudo-docs”: $M[c,w]$ = TF-IDF(c,w)
- May use any other measures of statistical association
- E.g., pointwise mutual information (PMI):
  $M[c,w] = \text{PMI}(c,w) = \log \frac{\Pr(c,w)}{\Pr(c)\ \Pr(w)}$
  “How much more likely are c and w to occur together than if they were independent?”
- word2vec: factor PMI matrix and use columns of B as word vectors



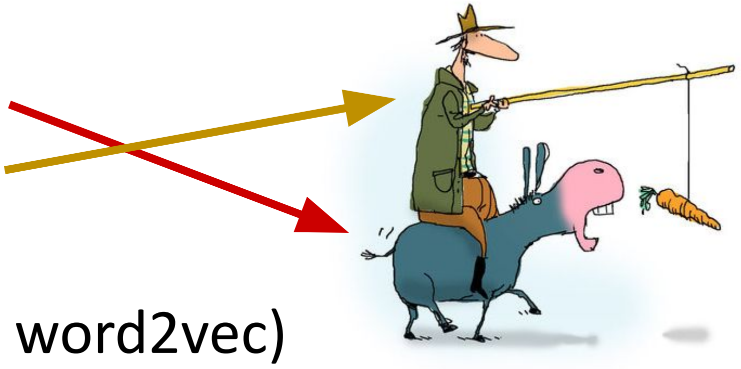
Beyond bags of words

From words to texts

- Word vectors represent, well, words
- How to represent larger units, such as sentences, paragraphs, docs?
- Typical approach: take sum/average of word vectors
- Note: this is roughly also what bags of words are (when using “one-hot” encoding for words, i.e., vector with exactly one 1, rest 0)
- More recently: **learn** vectors for longer units
  - Cr5, sent2vec
  - Convolutional neural networks
  - Recurrent neural networks, e.g., LSTM, ELMo
  - Transformer-based models, e.g., BERT (next slides), GPT-*

Contextualized word vectors

- Motivating example:
  - “My ass likes carrots”
  - vs. “He’s such an ass”

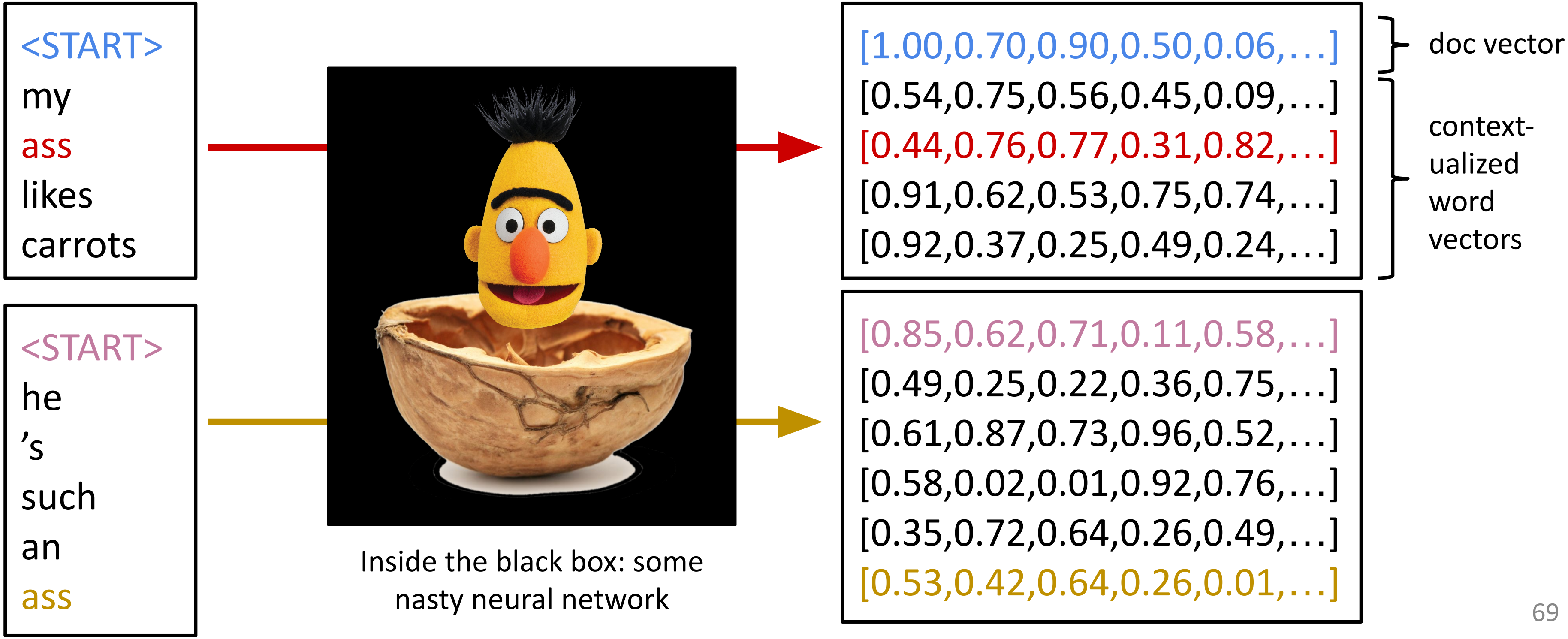


- Classic word vectors (e.g., word2vec) cannot distinguish these two cases; same vector used for both instances of “ass”

- Solution: contextualized word vectors
  - E.g., BERT

BERT in a nutshell

- Introduced in 2018 by Google Research



<START>
my
ass
likes
carrots

<START>
he
’s
such
an
ass

Inside the black box: some nasty neural network

[1.00,0.70,0.90,0.50,0.06,…]
[0.54,0.75,0.56,0.45,0.09,…]
[0.44,0.76,0.77,0.31,0.82,…]
[0.91,0.62,0.53,0.75,0.74,…]
[0.92,0.37,0.25,0.49,0.24,…]

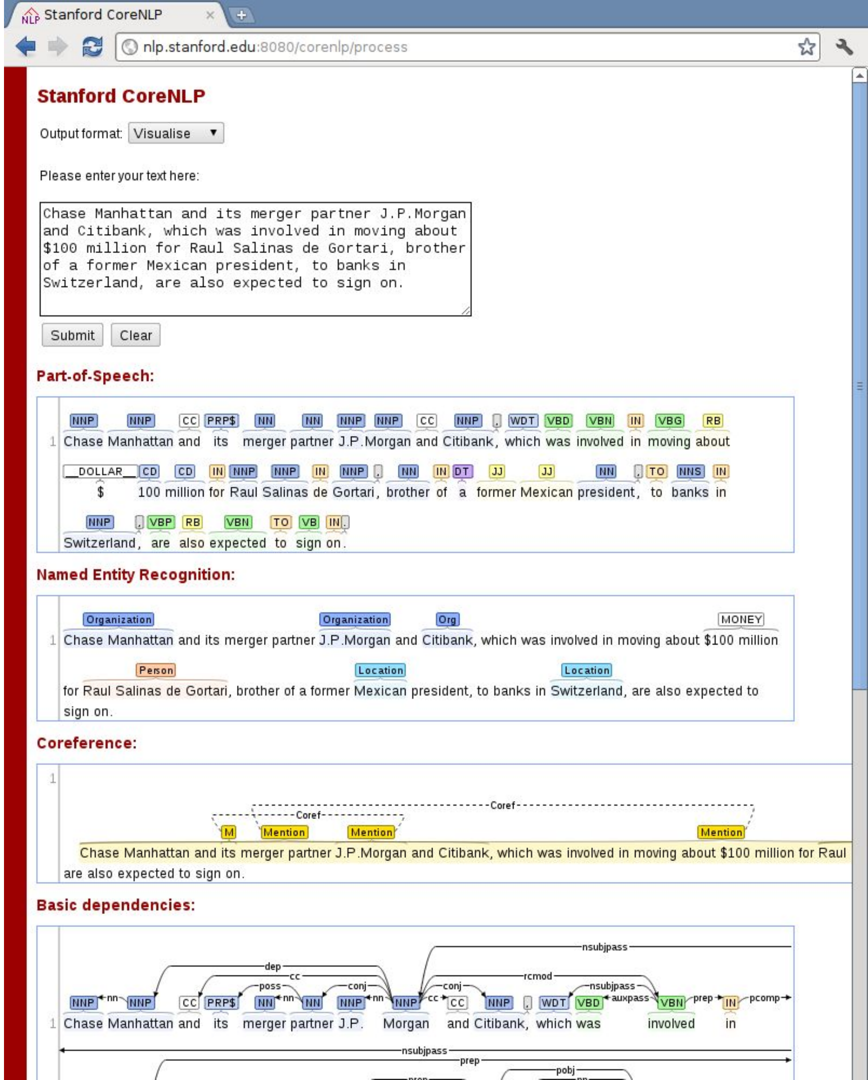
doc vector

context-ualized word vectors

[0.85,0.62,0.71,0.11,0.58,…]
[0.49,0.25,0.22,0.36,0.75,…]
[0.61,0.87,0.73,0.96,0.52,…]
[0.58,0.02,0.01,0.92,0.76,…]
[0.35,0.72,0.64,0.26,0.49,…]
[0.53,0.42,0.64,0.26,0.01,…]

NLP pipeline

- Tokenization
- Sentence splitting
- Part-of-speech (POS) tagging
- Named-entity recognition (NER)
- Coreference resolution
- Parsing
  - Shallow parsing (a.k.a. chunking)
  - Constituency parsing
  - Dependency parsing



Stanford CoreNLP

nlp.stanford.edu:8080/corenlp/process

Stanford CoreNLP

Output format: Visualise

Please enter your text here:

```
Chase Manhattan and its merger partner J.P.Morgan
and Citibank, which was involved in moving about
$100 million for Raul Salinas de Gortari, brother
of a former Mexican president, to banks in
Switzerland, are also expected to sign on.
```

Submit Clear

Part-of-Speech:

NNP NNP CC PRP$ NN NN NNP NNP CC NNP , WDT VBD VBN IN VBG RB
1 Chase Manhattan and its merger partner J.P.Morgan and Citibank , which was involved in moving about
$ CD CD IN NNP NNP IN NNP , NN IN DT JJ JJ NN , TO NNS IN
$ 100 million for Raul Salinas de Gortari , brother of a former Mexican president , to banks in
NNP , VBP RB VBN TO VB IN .
Switzerland , are also expected to sign on .

Named Entity Recognition:

1 Chase Manhattan and its merger partner J.P.Morgan and Citibank, which was involved in moving about $100 million for Raul Salinas de Gortari, brother of a former Mexican president, to banks in Switzerland, are also expected to sign on.

Coreference:

1 Chase Manhattan and its merger partner J.P.Morgan and Citibank, which was involved in moving about $100 million for Raul are also expected to sign on.

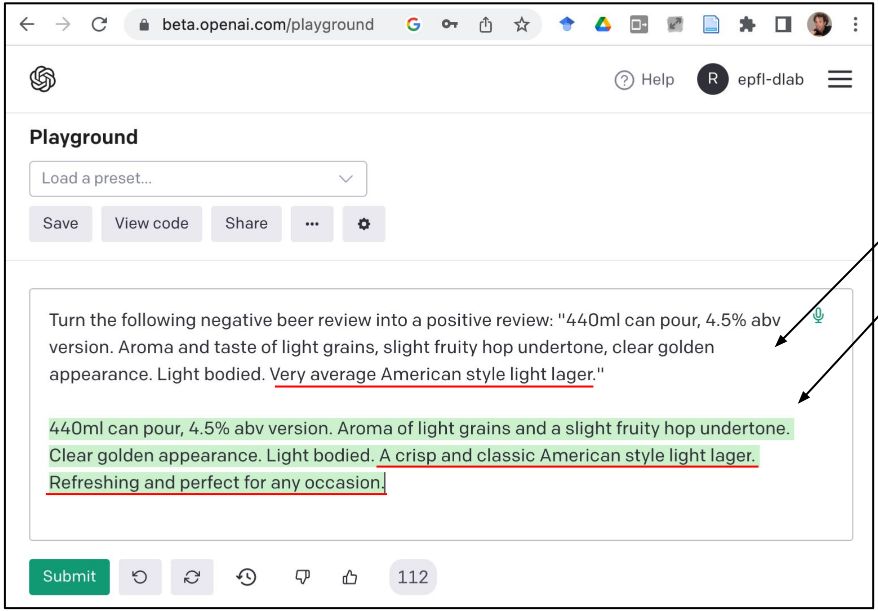
Basic dependencies:

nsubjpass
dep
cc
rcmod
poss
conj
conj
nsubjpass
nn
nn
nn
cc
auxpass
prep
pcomp
NNP NNP CC PRP$ NN NN NNP NNP CC NNP , WDT VBD VBN IN
1 Chase Manhattan and its merger partner J.P. Morgan and Citibank , which was involved in
nsubjpass
prep
pobj

NLP pipeline

- Implemented by Stanford CoreNLP, nltk, spaCy, etc.
- Sequential model
  - Fixed order of steps
  - Early errors will propagate downstream
  - Fixed order not optimal for all cases (e.g., syntax usually done before semantics, but semantics might be useful for inferring syntax)
- Hence, current research: learn all tasks jointly (early example)
- To learn how all this magic is implemented
  - Take CS-431 (Intro to NLP), CS-552 (Modern NLP)

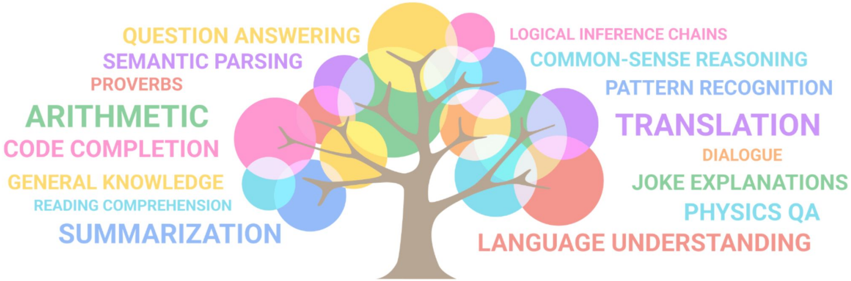
Today’s trend: generative language models



Turn the following negative beer review into a positive review: "440ml can pour, 4.5% abv version. Aroma and taste of light grains, slight fruity hop undertone, clear golden appearance. Light bodied. Very average American style light lager."

440ml can pour, 4.5% abv version. Aroma of light grains and a slight fruity hop undertone. Clear golden appearance. Light bodied. A crisp and classic American style light lager. Refreshing and perfect for any occasion.

- E.g., OpenAI GPT-*, ChatGPT
- Input: text
- Output: text
- Many NLP tasks can be formulated in this framework, by “prompting” the language model with the right input



Give us feedback on this lecture here:
https://go.epfl.ch/ada2023-lec10-feedback

- What did you (not) like about this lecture?
- What was (not) well explained?
- On what would you like more (fewer) details?
- …