Applied Data Analysis (CS401)

Lecture 10
Handling text data
Part I
19 Nov 2025

EPFL

Maria Brbić

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

- Milestone P2 feedback to be released **today**
- Homework due on **Wed Nov 26th 23:59**
- Friday's lab session:
  - Exercise on handling text data: Part I (Exercise 9)

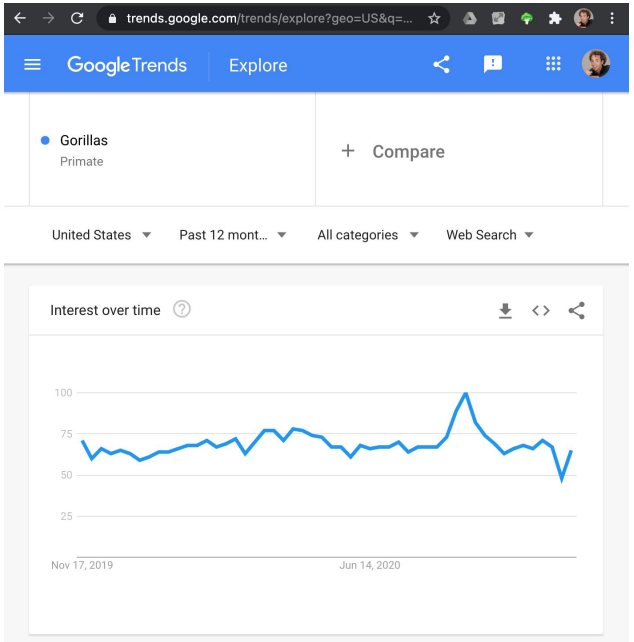
Feedback

Give us feedback on this lecture here:
https://go.epfl.ch/ada2025-lec10-feedback

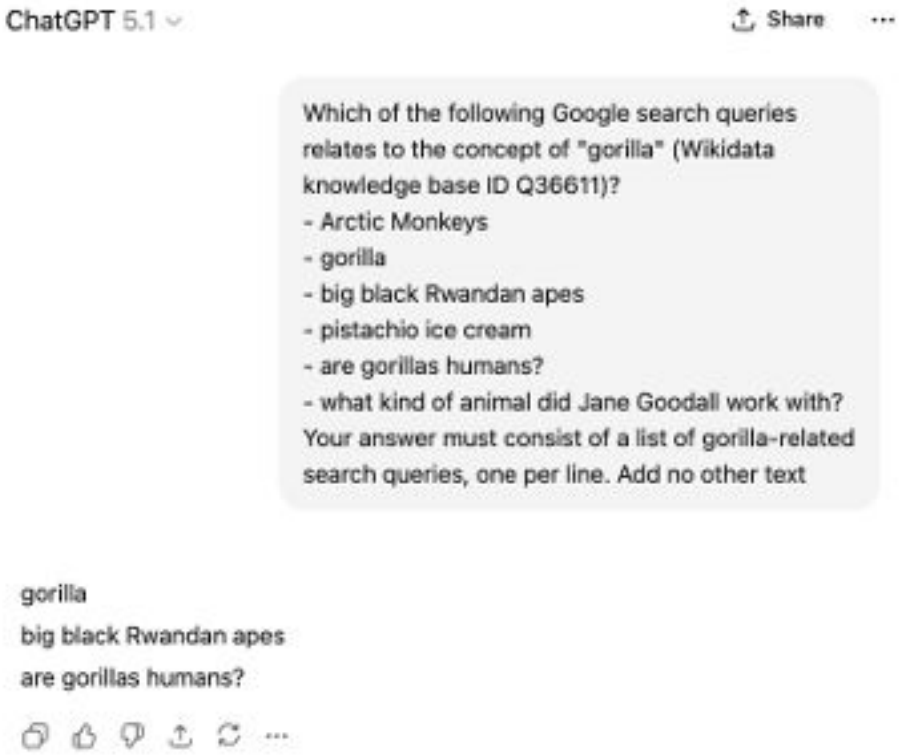
- What did you (not) like about this lecture?
- What was (not) well explained?
- On what would you like more (fewer) details?
- …

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



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 →
- 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

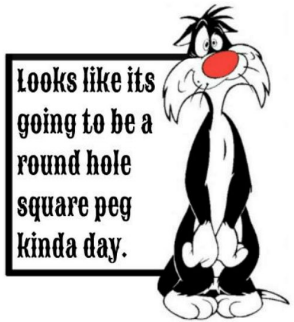
- **Given:**
  - Unlabeled document collection
- **Task:**
  - Determine a set of prevalent topics in the docs
  - Determine for each document to which topics it belongs
- Example scenario:
  - Detection of trending topics in social media (e.g., Twitter)
  - Detection of distinct viewpoints on a political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



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”)

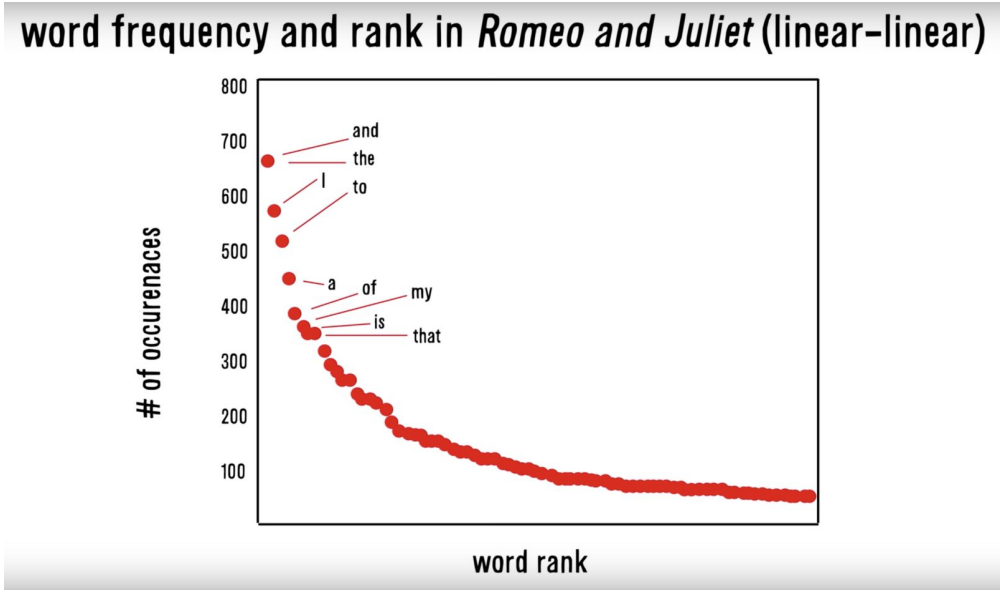
Bag of words

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

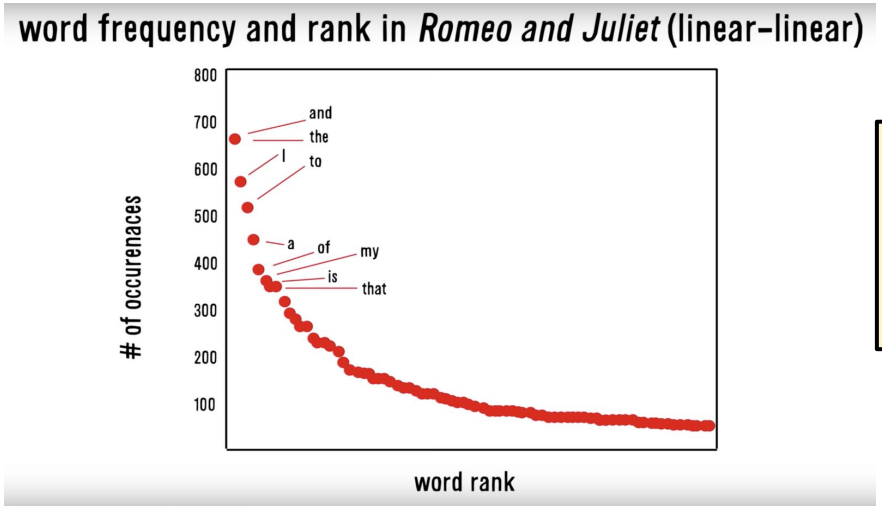


The probability of observing a word scales inversely with its frequency rank:

$$p(w_i) \propto 1/i$$

(where w_i is the i-th most frequent word)

Poll time



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

POLLING TIME

- Scan QR code or go to https://go.epfl.ch/ada2025-lec10-poll

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, ć, …
- 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", "prv")
  - Much harder if you messed up character encoding…

Commercial break

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

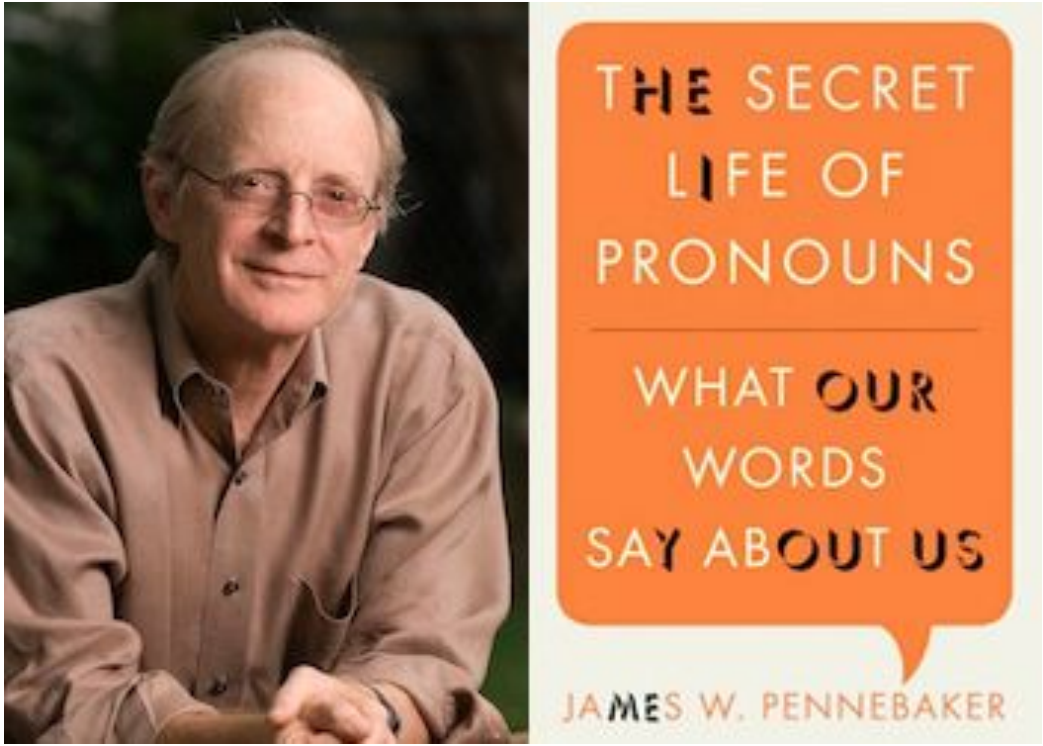
Tokenization

- 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 lolololol”

- 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

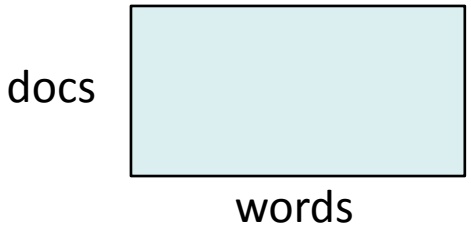
Tokens vs. n-grams

- 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



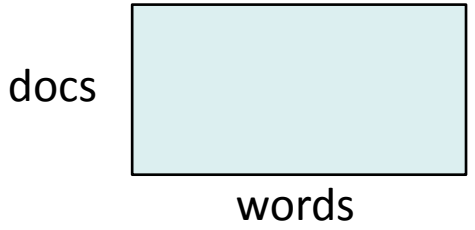
Inverse document frequency

- 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

Feedback

Give us feedback on this lecture here:
https://go.epfl.ch/ada2025-lec10-feedback

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