POLITÉCNICA
"Ingeniamos el futuro"

CAMPUS
DE EXCELENCIA
INTERNACIONAL

Madrid Summer School on
Advanced Statistics and Data Mining

Module C9 :: Text Mining and NLP

1st - 5th of July 2019

Florian Leitner
Selerity, Inc.; Data Catalytics, SL; IE University
leitner@datacatytics.com

Overview

- Monday
  - Concepts & Terminology
  - Information Retrieval
  - Text Classification
- Tuesday
  - Evaluation Metrics
  - Unsupervised Methods
    - Sentences and Collocations
    - Locality Sensitive Hashing
    - Latent Semantic Indexing*
    - Latent Dirichlet Allocation*
- Wednesday*
  - Representation Learning
    - Embeddings
- Thursday
  - Open Information Extraction
    - Text Summarization
    - Keyword Extraction
    - Dynamic Graphical Models
    - Named Entity Recognition+
- Friday+
  - Natural Language Processing
    - Dependency Parsing
  - Review and Outlook
    - A SOTA deep classifier for text

The "one single" book recommendation

- **Speech and Language Processing**
  - Dan Jurafsky and James H. Martin
  - https://www.cs.colorado.edu/~martin/slp.html
- 3rd edition in the making
  - Will be covering the new deep learning aspects, too
  - chapter drafts available from: https://web.stanford.edu/~jurafsky/slp3/

Concepts & Terminology

Madrid Summer School on
Advanced Statistics and Data Mining

Florian Leitner
Selerity, Inc.; Data Catalytics, SL; IE University
leitner@datacatytics.com

Applications of text mining and language processing

Text mining

(Web) Search engines

Information retrieval

Spam filtering

Document classification

Twitter brand monitoring

Opinion mining ("sentiment" analysis)

Finding similar items (Amazon)

Content-based recommender systems

Event detection in e-mail

Information extraction

Spelling correction

Statistical language modeling

Siri (Apple) and Google Now

Language understanding

Website translation (Google)

Machine translation

Chat bots, fake news/reviews

Language generation

Watson in Jeopardy! (IBM)

Question answering

Language processing

Ambiguity

Lexical A.

Part-of-Speech tagging

The robot **wheels** out the iron.

Named entity recognition

Is **Paris** really good for you?

Structural A.

Semantic ambiguity

They saw her duck.

Syntactic ambiguity

She ate the hot-dog with relish.

Metaphors

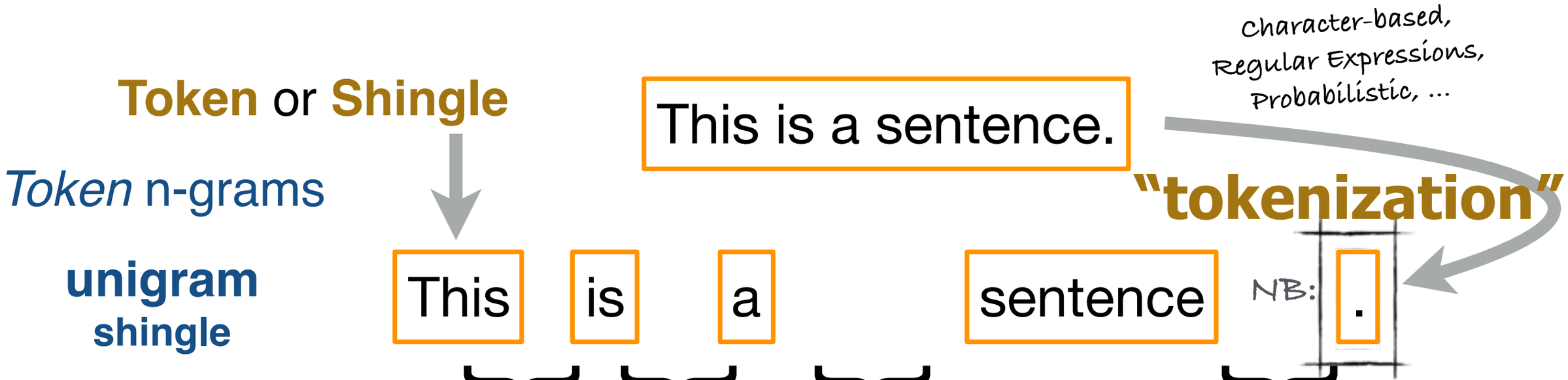
Anaphora resolution

Carl and Bob were fighting:
"**You** should shut up,"
Carl told **him**.

Paraphrasing

Unemployment is on the rise.
vs
The economy is slumping.

Tokens and n-grams



Character n-grams

all **trigrams** of "sentence":
[sen, ent, nte, ten, enc, nce]

Beware: the terms "shingle", "token" and "n-gram" are not used consistently…

The parts of speech (PoS)

I	ate	the	pizza	with	green	peppers	.
PRP	VB	DT	NN	IN	JJ	NN	.

- The Parts of Speech:
  - noun: NN, verb: VB, adjective: JJ, adverb: RB, preposition: IN, personal pronoun: PRP, …
  - e.g. the full **Penn Treebank PoS tagset** contains 48 tags:
  - 34 grammatical tags (i.e., "real" parts-of-speech) for words
  - one for cardinal numbers ("CD"; i.e., a series of digits)
  - 13 for [mathematical] "SYM" and currency symbols ($, €, …), various types of punctuation, as well as for opening/closing parenthesis and quotes

Linguistic morphology

- [Verbal] **Inflections**
  - conjugation (Indo-European languages)
  - tense ("availability" and use varies across languages)
  - modality (subjunctive, conditional, imperative)
  - voice (active/passive)
  - …
- **Contractions**
  - don't say you're in a man**'s** world…

not a contraction: possessive s

- **Declensions**
  - on nouns, pronouns, adjectives, determiners
  - case (nominative, accusative, dative, ablative, genitive, …)
  - gender (female, male, neuter)
  - number forms (singular, plural, dual)
  - possessive pronouns (I→my, you→your, she→her, it→its, … car)
  - reflexive pronouns (for myself, yourself, …)
  - …

Stemming & Lemmatization

➡ "token normalization"

a.k.a. token "regularization"
(although that is technically the wrong wording)

- Stemming
  - produced by "**stemmers**"
  - produces a word's "stem"
  - am → am
  - the going → the go
  - having → hav
  - fast and simple (pattern-based)
  - **Snowball**; **Lovins**; **Porter**
- Lemmatization
  - produced by "**lemmatizers**"
  - produces a word's "lemma"
  - am → be
  - the going → the going
  - having → have
  - requires: a dictionary for the mappings and the **PoS tags (!)**
  - **LemmaGen**; **morpha**; **SpaCy; BioLemmatizer**; **geniatagger**

Collocations (and idioms)

A **Collocation** is an expression consisting of two or more words that correspond to some conventional way of saying things.

"Collocations of a given word are statements of the habitual or customary places of that word." -Firth (1957: 181)

Collocations are **non-compositional** (i.e., the meaning of the expression cannot be understood [viz., composed] from its parts): meaning is added to the term (making it a special version of the linguistic concept of a **term**).

strong tea, weapons of mass destruction, to make up, the rich and powerful, stiff breeze, broad daylight, white wine, ...

collocation test: Is a literal word-by-word translation to another language possible? (if not, its probably a collocation)

Manning and Schütze. Foundations of Statistical Natural Language Processing. 2000

Converting tokens to numbers

- **Tokenization** splits text into words, punctuation, and symbols (tokens), and tokens can be combined by detecting **collocations**: A **term** therefore can refer to either a token or a collocation.
- **Indexing** groups equal terms and counts them: a **bag of words**.
  - Counting terms within a document: (token or) **term frequency TF**
  - Counting the number of documents with a given term: **document frequency DF**
  - Overall count of a term in a document collection: **corpus frequency CF**
- **Inference** can be used learn unsupervised vector representations of character n-grams, terms, sentences, paragraphs or documents.
- The remaining question then becomes: How to make computations with these numbers (the indexed terms or vector representations)?

Zipf's law: The Pareto distribution of terms

Axioms

Word frequency is inversely proportional to its rank (ordered by counts)

$$f \propto 1 \div r$$

$$k = E[f \times r]$$

E = Expected value →

the "true" mean →

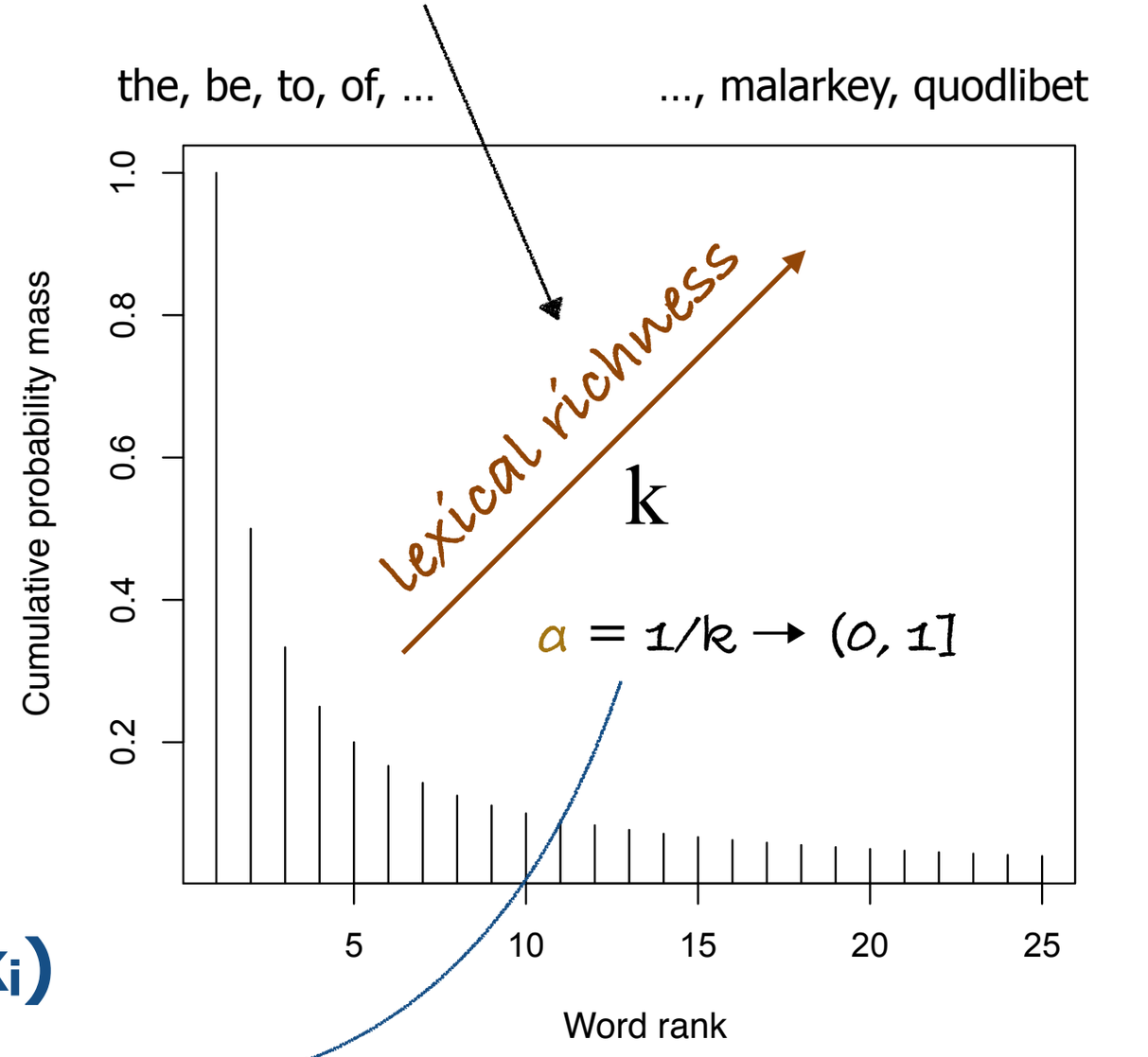
dim. reduction

Words of lower rank "clump" within a region of a document

Word frequency is inversely proportional to its length

Almost all words are rare

This is what makes language modeling hard!

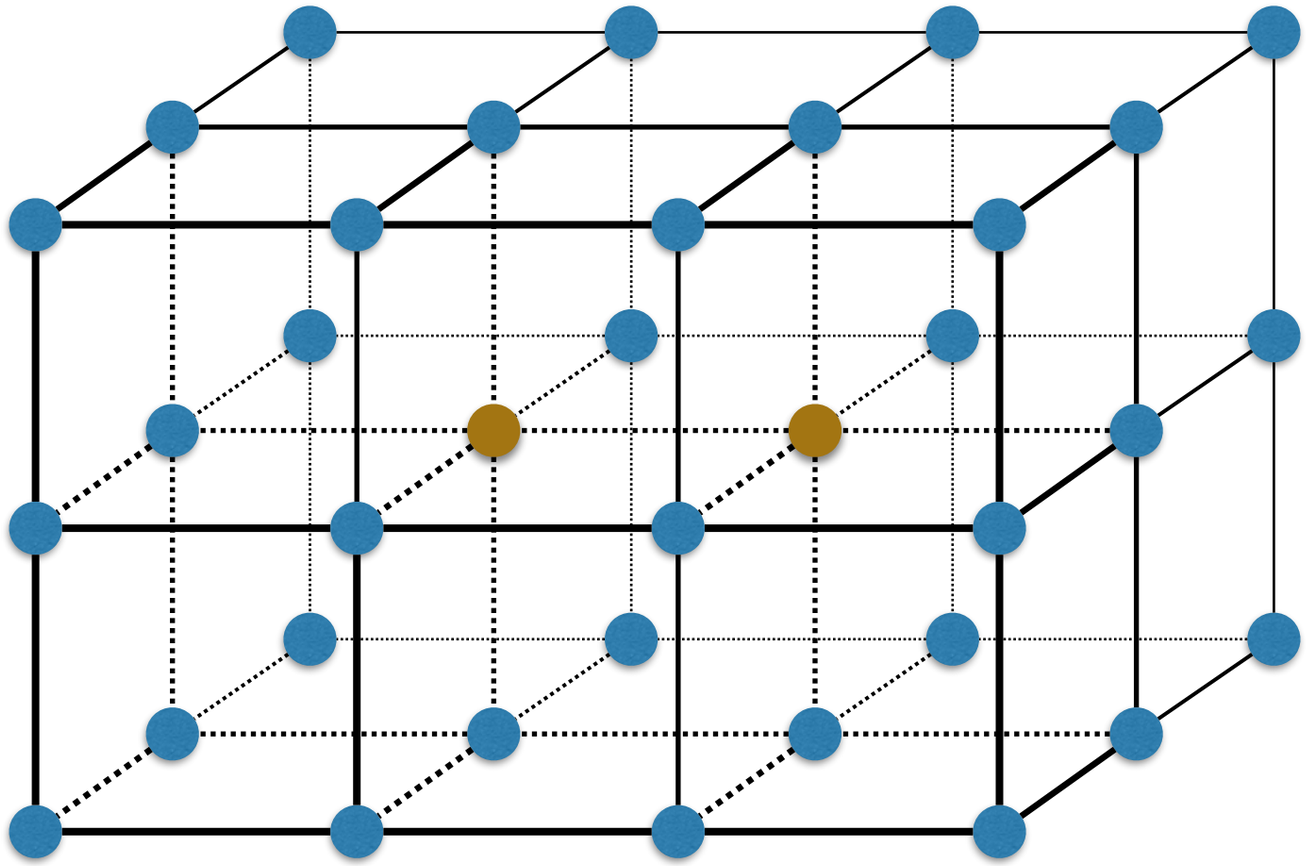


"power law"

$count(w_i) \propto P(w_i) = pow(\alpha, rank_i)$

The curse of dimensionality

(RE Bellman, 1961) [inventor of dynamic programming]



- $\mathbf{p} \gg \mathbf{n}$ (far more tokens/features than texts/documents)
- Inverted indices are (discrete) **sparse** matrices.
- Even with millions of training examples, **unseen** tokens will keep coming up during evaluation or in production.
- In a high-dimensional hypercube, most instances are closer to the **face of the cube** ("nothing", outside) than their nearest neighbor.

Dimensionality reduction

✓ The remedy to "the curse" (aka. the "blessing of non-uniformity")

- Feature **extraction** (compression): PCA/LDA (projection), factor analysis (regression), compression, auto-encoders, word embeddings, representation learning, …
- Feature **selection** (elimination): LASSO (regularization), SVM (support vectors), Bayesian nets (structure learning), locality sensitive hashing, random projections/forests, …

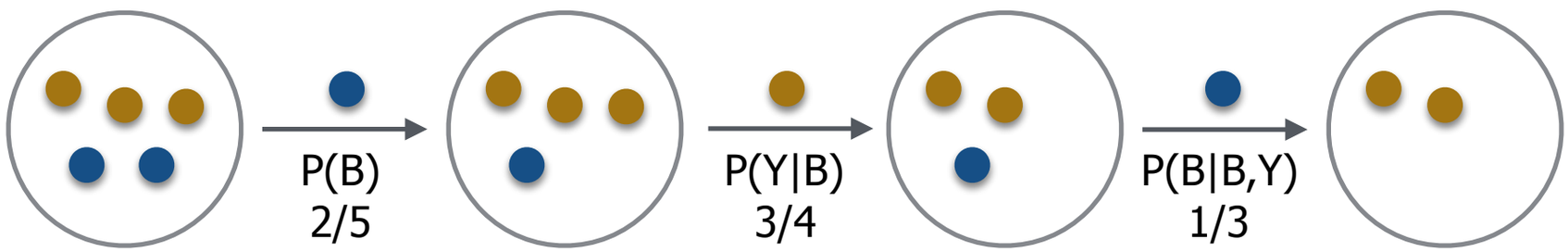
The chain rule of conditional probability

$$P(A,B,C,D) = P(A) \times P(B|A) \times P(C|A,B) \times P(D|A,B,C)$$

Notation: i-1 is an index, not the power!

$$P(w_1, ..., w_n) = \prod_{i=1}^{n} P(w_i|w_1, ..., w_{i-1}) = \prod_{i=1}^{n} P(w_i|w_1^{i-1})$$

joint — conditional — ...to w_{i-1} from w_1...



NB: without replacement!

$$P(B,Y,B) = P(B) \times P(Y|B) \times P(B|B,Y)$$
$$1/10 = 2/5 \times 3/4 \times 1/3$$

NB: the $\sum$ of all possible trigram combinations will be 1!

Information Retrieval

Madrid Summer School on
Advanced Statistics and Data Mining

Florian Leitner
Selerity, Inc.; Data Catalytics, SL; IE University
leitner@datacatytics.com

Information retrieval: Incentives and applications

- Decision support, question answering, recommendation.
- Retrieving text based on queries.
  - Search engines, percolation (querying future [unindexed] data streams)
- Easiest to set up and possibly the most studied TM application.
  - Advanced IR systems will use many of the unsupervised and representational methods we will only learn about later.

The inverted index

factors, normalization (len[text]), probabilities, and n-grams

Text 1: He that not wills to the end neither
wills to the means.

Text 2: If the mountain will not go to Moses,
then Moses must go to the mountain.

tokens	Text 1	Text 2
end	1	0
go	0	2
he	1	0
if	0	1
means	1	0
Moses	0	2
mountain	0	2
must	0	1
not	1	1
that	1	0
the	2	2
then	0	1
to	2	2
will	2	1

unigrams	T1	T2	p(T1)	p(T2)
end	1	**0**	0.09	0.00
go	**0**	2	0.00	0.13
he	1	**0**	0.09	0.00
if	**0**	1	0.00	0.07
means	1	**0**	0.09	0.00
Moses	**0**	2	0.00	0.13
mountain	**0**	2	0.00	0.13
must	**0**	1	0.00	0.07
not	1	1	0.09	0.07
that	1	**0**	0.09	0.00
the	2	2	0.18	0.13
then	**0**	1	0.00	0.07
to	2	2	0.18	0.13
will	2	1	0.18	0.07
SUM	11	15	1.00	1.00

bigrams	Text 1	Text 2
end, neither	1	**0**
go, to	**0**	2
he, that	1	**0**
if, the	**0**	1
Moses, must	**0**	1
Moses, then	**0**	1
mountain, will	**0**	1
must, go	**0**	1
not, go	**0**	1
not, will	1	**0**
that, not	1	**0**
the, means	1	**0**
the, mountain	**0**	2
then, Moses	**0**	1
to, Moses	**0**	1
to, the	**2**	**1**
will, not	**0**	1
will, to	2	**0**

Document similarity

- Similarity measures
  - **Cosine similarity (of document/text vectors)**
  - **Correlation coefficients**
- Word vector normalization
  - **TF-IDF**
- Dimensionality reduction/clustering
  - **Locality sensitive hashing**
  - **Latent semantic indexing**
  - **Latent Dirichlet allocation**
  - **Neural document embeddings**

Document (word) vectors

Collections of vectorized texts: **Inverted index**

Text 1: He that not wills to the end neither wills to the means.

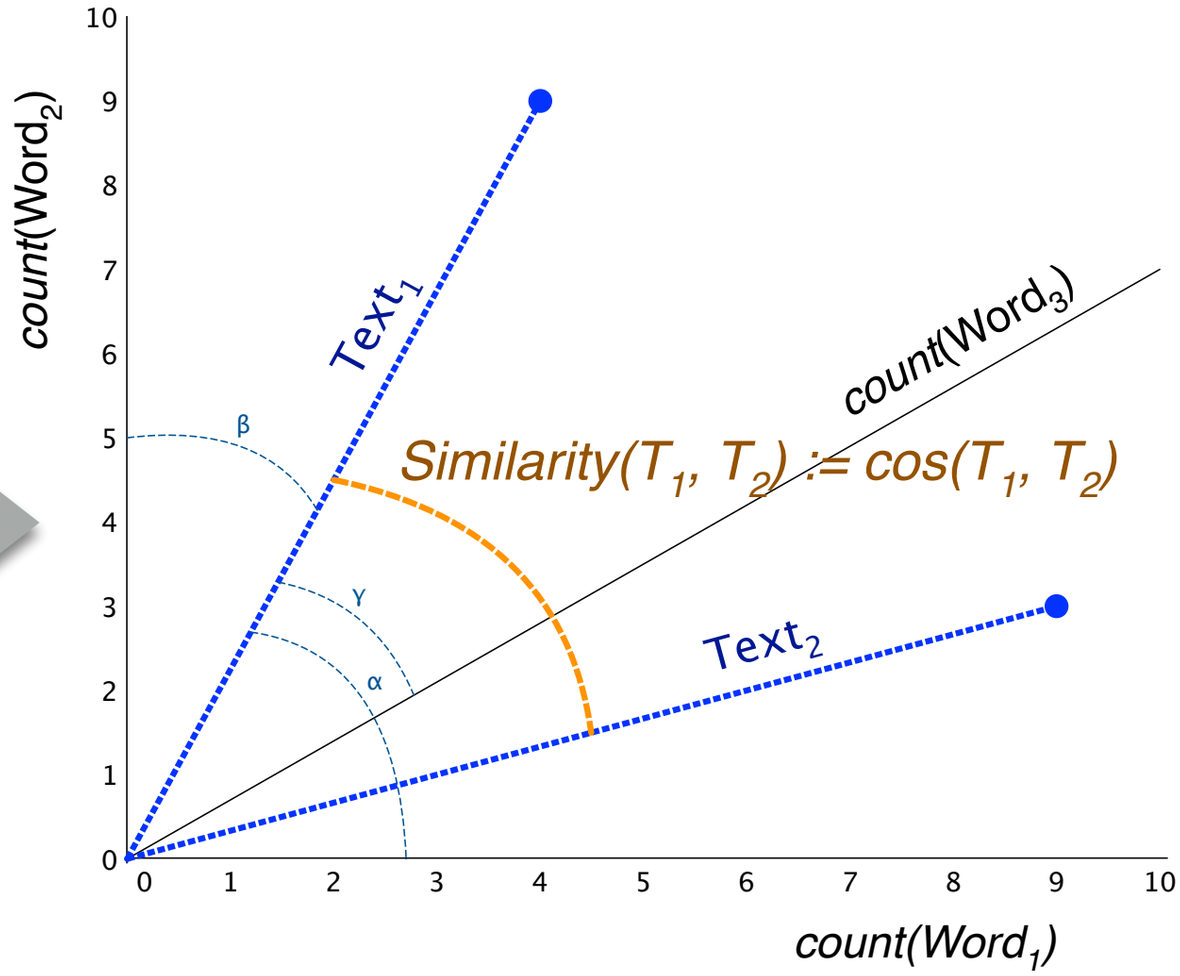
Text 2: If the mountain will not go to Moses, then Moses must go to the mountain.

tokens	Text 1	Text 2
end	1	0
go	0	2
he	1	0
if	0	1
means	1	0
Moses	0	2
mountain	0	2
must	0	1
not	1	1
that	1	0
the	2	2
then	0	1
to	2	2
will	2	1

↑ word vector ↑ word vector

each token/word is a dimension!

Comparing word vectors: **Cosine similarity**



Terrier Sphinx Xapian INDRI Lucene

Cosine similarity

$$sim(\vec{x},\vec{y}) = cos(\vec{x},\vec{y}) = \frac{\vec{x}\cdot\vec{y}}{|\vec{x}||\vec{y}|} = \frac{\sum x_i y_i}{\sqrt{\sum x_i^2}\sqrt{\sum y_i^2}}$$

- Define a similarity score between two document vectors (or the query vector in Information Retrieval)
- **Euclidian** vector **distance** is **length dependent**
- The **cosine** [angle] between two vectors **is not**

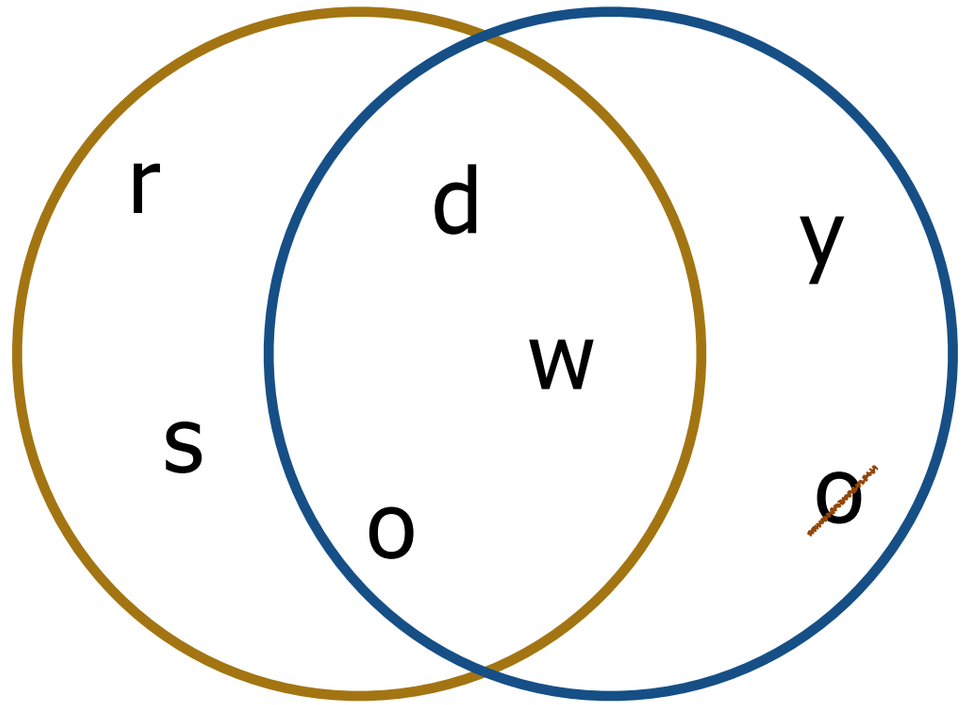
can be dropped if using unit vectors ("length-normalized" a.k.a. "cosine normalization") only dot-product is now left: extremely efficient ways to compute on modern CPUs

$$sim(T_1,T_2) = \frac{1*1+2*2+2*2+2*1}{\sqrt{(5+3*4)} * \sqrt{(5+5*4)}} = 0.5336$$

tokens	Text 1	Text 2
end	1	0
go	0	2
he	1	0
if	0	1
means	1	0
Moses	0	2
mountain	0	2
must	0	1
not	1	1
that	1	0
the	2	2
then	0	1
to	2	2
will	2	1

Jaccard's [set] similarity

"words" vs "woody"



Typical "mistake": Jaccard cannot distinguish between ABC and ABCABCABC!

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{3}{7} = 0.43$$

6

0.5

Alternative similarity coefficients

- **Spearman's rank correlation coefficient** ρ (r[ho])
  - Ranking is done by term frequency (**TF**; count)
  - Critique: sensitive to ranking differences that are likely to occur with high-frequency words (e.g., "the", "a", …) ➡ use the **log** of the term count, rounded to two significant digits
    - NB that this is not relevant when only short documents (e.g. titles) with low TF counts are compared
- **Pearson's chi-square test** χ^2
  - Directly on the TFs (counts) - intuition:
    Are the TFs "random samples" from the same base distribution?
  - Usually, χ^2 should be preferred over ρ (Kilgarriff & Rose, 1998)
    - NB that both measures have no inherent normalization of document size
      - preprocessing might be necessary!

Term frequency times inverse document frequency (TF-IDF)

- **Motivation and background**
- The problem
  - **Frequent terms** contribute most to a document vector's direction, but **not all** terms are **relevant** ("the", "a", …).
- The goal
  - **Separate** important terms from frequent, but **irrelevant terms** in the collection.
- The idea
  - Frequent **terms** appearing **in all documents** tend to be less important **versus** frequent terms in just a **few documents**. → Zipf's Law!
  - Also **dampens** the effect of **topic-specific noun phrases** or an **author's bias** for a specific set of adjectives

Term frequency times inverse document frequency (TF-IDF)

- **tf.idf**(w) := tf(w) × idf(w)
  - tf: (document-specific) term frequency
  - idf: inverse (global) document frequency

- $\mathbf{tf_{natural}}(w) := \text{count}(w)$
  - $\text{tf}_{\text{natural}}$: n. of times term w occurs in a document
- $\mathbf{tf_{log}}(w) := \log(\text{count}(w) + 1)$
  - tf_{log}: the TF is smoothed by taking its log

- $\mathbf{idf_{natural}}(w) := N / \sum^{N} \{w_i > 0\}$
  - $\text{idf}_{\text{natural}}$: n. documents divided by n. documents in which term w occurs
- $\mathbf{idf_{log}}(w) := \log(N / \sum^{N} \{w_i > 0\})$
  - idf_{log}: the IDF is smoothed by taking its log
  - where N is the **number of documents**, w_i the **count of word** w in document i, and $\{w_i>0\}$ is 1 if document i has word w or 0 otherwise

TF-IDF in information retrieval

- Document vectors = tf_{log}
- Query vector = $tf_{log}\ idf_{log}$
  - Terms are counted on each individual document & the query
- Cosine vector length normalization for TF-IDF scores:
  - Document W normalization
  - Query Q normalization
- IDF is calculated over the indexed collection of all documents

→ i.e. here the doc. vectors do not use IDF weighting because the QV uses IDF, too, and gets multiplied with the DV.

$$\sqrt{\sum_{w \in W} tf_{log}(w)^2}$$

i.e., create unit-vectors (see Cosine similarity)

$$\sqrt{\sum_{q \in Q} (tf_{log}(q) \times idf_{log}(q))^2}$$

TF-IDF query score: An example

	Collection		Query Q: "best TM tutorial"				Document D			Similarity
Term	df	idf_{log}	tf	tf_{log}	tf.idf	norm	tf	tf_{log}	tf.1	cos(Q,D)
best	3.5E+05	1.46	1	0.30	0.44	0.21	**0**	0.00	0.00	0.00
text	2.4E+03	3.62	1	0.30	1.09	0.53	10	1.04	0.06	0.03
mining	2.8E+02	4.55	1	0.30	1.37	0.67	8	0.95	0.06	0.04
tutorial	5.5E+03	3.26	1	0.30	0.98	0.48	3	0.60	0.04	0.02
data	9.2E+05	1.04	**0**	0.00	0.00	0.00	10	1.04	0.06	0.00
…	…	…	…	…	0.00	0.00	…	16.00	…	0.00
Sums	1.0E+07		4		2.05		~355	16.11		**0.09**

√ of ∑ of ² ÷ √ of ∑ of ² ÷

3 out of hundreds/…/millions of unique words match (Jaccard < 0.03)

Example idea from: Manning et al. Introduction to Information Retrieval. 2009 free PDF!

TF-IDF in Python

- SciKit-Learn has a "bells and whistles" implementation via the TfidfVectorizer API.
- Details in the Practical 07, Document classification.

Text Classification

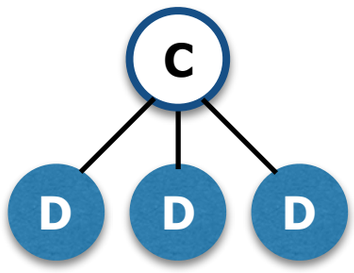
Madrid Summer School on
Advanced Statistics and Data Mining

Florian Leitner
Selerity, Inc.; Data Catalytics, SL; IE University
leitner@datacatytics.com

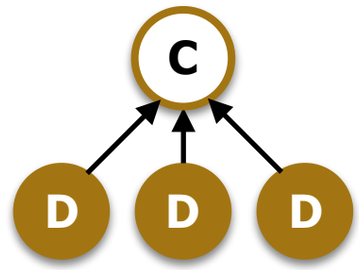
Text classification: Incentives and applications

- Assign one or more "labels" (classes) to a collection of "texts".

- Spam filtering
- Marketing and politics (**opinion mining**)
- Topic assignment (clustering or, here: classification)
- …



Generative vs. discriminative models



- Generative models describe how the [hidden] labels generated the [observed] input as **joint probabilities**: $\mathrm{P}(class, data)$
  - They learn a data distribution for each individual class.
  - Examples: Markov Chain, **Naïve Bayes**, Latent Dirichlet Allocation, Hidden Markov Model, …
  - Graphical models for detecting outliers or when there is a need to update models (change)
- Discriminative models predict (discriminate) the [hidden] labels **conditioned** on the [observed] input: $\mathrm{P}(class \mid data)$
  - They ("only") learn the data boundaries between classes.
  - Ex.: Logistic Regression (**MaxEnt**), SVM, Conditional Random Field, Random Forest, …
- Both can identify the most likely labels and their likelihoods
- **Only generative models**:
  - Likelihood of an input (data) for a particular label
  - Most likely input data and their likelihoods

$$P(H|D) = \frac{P(H) \times P(D|H)}{P(D)}$$

The maximum a-posteriori (MAP) estimator

- Issue: Predict the class $c \in C$ for a given document $d \in D$
- Solution: MAP, the "perfect" generative Bayesian estimator:

$$C_{MAP}(d) = \underset{c \in C}{argmax}\ P(c|d) = \underset{c \in C}{argmax}\ \frac{P(d|c)P(c)}{P(d)}$$

the posterior

redundant computation

- Problem: d is really a set $\{w_1, \ldots, w_n\}$ of dependent terms V
  - exponential parameterization: one for each possible combination of V and C

Multinomial naïve Bayes classification

- A simplification of the MAP Estimator
  - $\text{count}(w)$ is a discrete, **multinomial** variable (unigrams, bigrams, etc.)
  - Reduce space by making a **strong independence assumption** ("naïve")

Naive assumption: "each word/token/term is independent of all others"

$$C_{MAP}(d) = \underset{c \in C}{argmax}\ P(d|c)P(c) \approx \underset{c \in C}{argmax}\ P(c) \prod_{w \in W} P(w|c)$$

"bag of words/tokens/terms"

- Easy **parameter estimation**

$$\hat{P}(w_i|c) = \frac{count(w_i, c) + 1}{|V| + \sum_{w \in V} count(w, c)}$$

$\text{count}(w_i, c)$: the total count of word i in all documents of class c [in our training set]

  - V is the entire **vocabulary** (collection of unique words/tokens/terms…) in D
  - Re. the "+1" term: Laplacian ("add-one") smoothing; → $|V|$ in the denom.

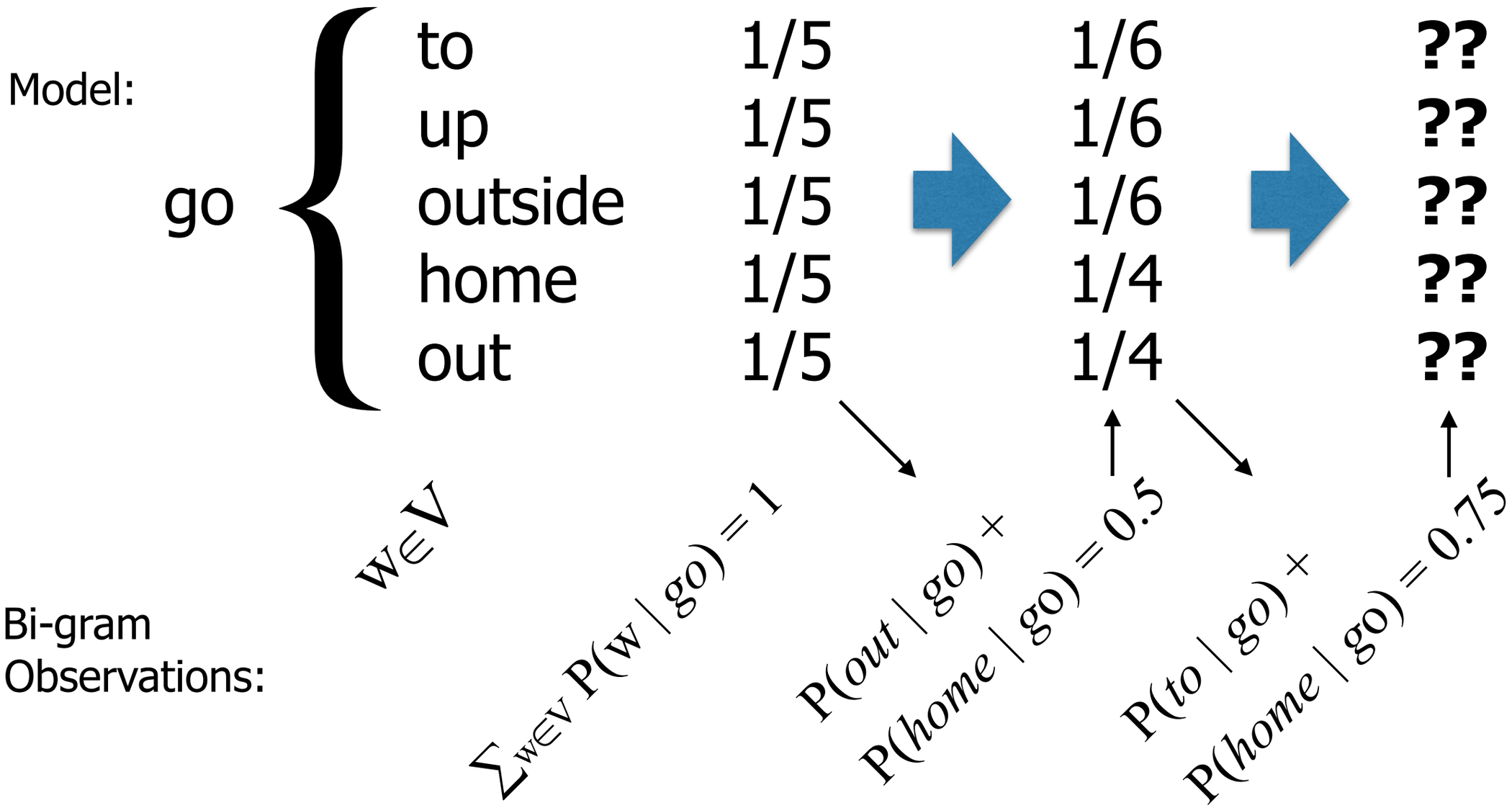
Multinomial naïve Bayes: Practical aspects

- Can gracefully handle **unseen words** (add-one smoothing)
- Has **low space requirements**: $|V| + |C|$ floats → sum $\prod$ using logs!
- **Irrelevant** ("stop") **words** cancel each other out for large $|V|$
- Opposed to SVM or Nearest Neighbor, it is very **fast**
  - (Locality Sensitive Hashing is another very efficient approach)
  - But fast approaches tend to result in lower accuracies (➡ good "**baselines**")
- **Each class** has its own **n-gram language model**
- **Logarithmic damping** ($\log(count)$) or **TF-IDF** typically improves classification

$$\hat{P}(w_i|c) = \frac{log(count(w_i, c) + 1)}{log(|V| + \sum_{w \in V} count(w, c))}$$

The Maximum Entropy (MaxEnt) classifier: Intuition

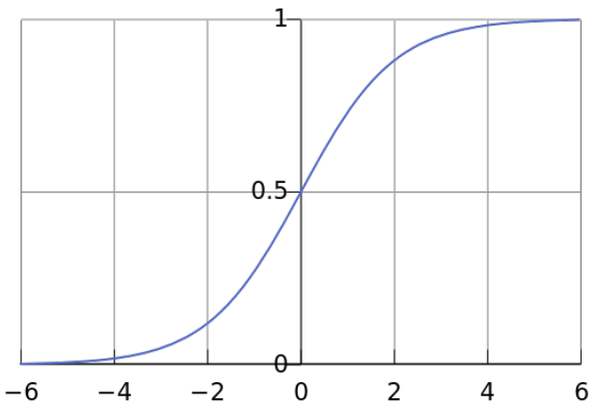
The principle of maximum entropy



Bi-gram Observations:

Supervised MaxEnt classification

$$p(x) = \frac{1}{1 + exp(-(\lambda_0 + \lambda_1 x))}$$



logistic function p

Image Source: WikiMedia Commons, Qef

ln

$$ln \frac{p(x)}{1 - p(x)} = \lambda_0 + \lambda_1 x$$

exp

$$\frac{p(x)}{1 - p(x)} = exp(\lambda_0 + \lambda_1 x)$$

odds-ratio!

- a.k.a. **multinomial logistic regression**
  (multinomial: two labels or more, i.e., multi-class)
- **Does not assume independence of features**
- Can model **mixtures of** binary, discrete, and real **features**
- Training data are **per-feature-label-pair probab.s**: $\mathrm{P}(\mathrm{F}, \mathrm{L})$
  - I.e., $\mathrm{count}(f_j, l_j) \div \sum_{i=1}^{N} \mathrm{count}(f_i, l_i)$

➡ words → very sparse training data (zero or few examples)

- Model parameters are commonly learned using gradient descent
  - Expensive if compared to Naïve Bayes, but efficient optimizers exist (**L-BFGS**)

Mixed type feature probabilities with indicator functions

- Some definitions:
  - The observable joint prob. of y (label) with x (word) over N (instances) is:

$$\hat{P}(x, y) = count(x, y) \div N$$

  - An **indicator function** ("**feature**") is defined as a binary valued function that returns 1 iff class and data match the **indicated** requirements (**constraints**):

$$f(x, y) = \begin{cases} 1 \; if \; y = c_i \wedge x = w_i \\ 0 \; otherwise \end{cases}$$

Indicator functions are "on-off switches"

real/discrete/binary features are all the same, it simply is a constraint match

  - Therefore, the probability of a feature with respect to the observed distribution of X and Y is:

$$\hat{P}(f_i, X, Y) = E_{\hat{P}}[f_i] = \sum_{(x,y) \in (X,Y)} \hat{P}(x, y) f_i(x, y)$$

"for each feature, label pair"

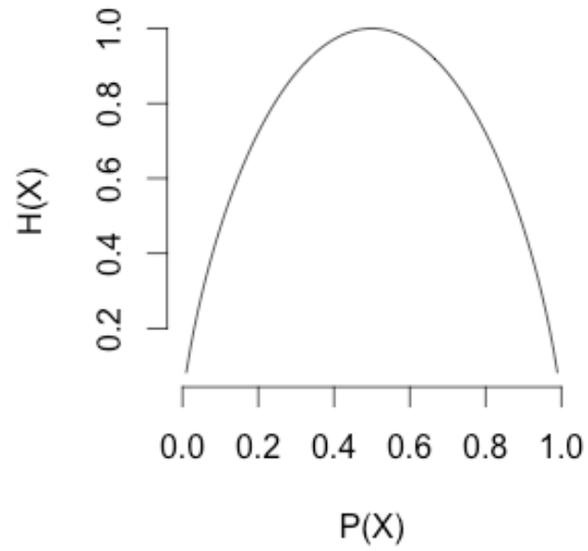
Indicator function examples

- Examples of indicator functions (a.k.a. **feature functions**)
  - Assume we wish to classify the general polarity (positive, negative) of product reviews:
- $f(c, w) := \{c = \text{POSITIVE} \wedge w = \text{“}great\text{”}\}$
  - Equally, for classifying words in a text, say to detect proper names, we could create a feature:
- $f(c, w) := \{c = \text{NAME} \wedge \text{isCapitalized}(w)\}$
- Note that while we can have multiple classes, we cannot require more than **one class** in the whole match condition of a single indicator (feature) function.

NB: typical text mining models can have a million or more features:
unigrams + bigrams + trigrams + counts + dictionary matches + …

Shannon's entropy (reminder)

- Answers the questions:
  - "How much information (bits) will I gain when I see w_n?"
  - "How predictable is w_n from its past?"



Bernoulli process: $H(X) = -P(X)log_2[P(X)] - (1 - P(X))log_2[1 - P(X)]$

- Each outcome provides $-log_2 P$ bits of information ("surprise").
  - Claude E. Shannon, 1948
  - The more probable an event (outcome), the lower its entropy.
  - A certain event (p=1 or 0) has zero entropy ("no surprise").

$$H(X) = -\sum P(x_i)\ log_2\ P(x_i)$$

Maximizing the conditional entropy

- The **conditioned** (on X) version of Shannon's **entropy** H:

(for our domain, Y is the class and X the term)

$$H(Y|X) = \sum_{x \in X} P(x)\ H(Y|X = x)$$

vs. NB: summing, not multiplying!

H(X) = -∑P(x) log₂ P(x)

(swapped nom/denom to remove the minus)

$$= -\sum_{x \in X} P(x) \sum_{y \in Y} P(y|x)\ log_2\ P(y|x)$$

$$= \sum_{x,y \in X,Y} P(x,y)\ log_2 \frac{P(x)}{P(x,y)}$$

Applying the chain rule

$$P(x,y) = P(x)\ P(y|x)$$

- MaxEnt **training** then is about **selecting** the **model** $\mathrm{p}*$ that maximizes the conditional entropy H:

$$p^* = \underset{p \in P}{argmax}\ H(P) = \underset{p \in P}{argmax}\ H(Y|X)$$

Feature weights and the MaxEnt model

- In a **linear** model, we'd use weights ("lambdas") that identify the most relevant features of our model, i.e., we use the following MAP to select a class:

$$\underset{y \in Y}{argmax} \sum \lambda_i f_i(X, y)$$

Linear model: Sum → features not independent

high dimensionality makes linear model OK-ish

- To do multinomial **logistic** regression, expand with a **linear combination**:

$$\underset{y \in Y}{argmax} \frac{exp(\sum \lambda_i f_i(X, y))}{\sum_{y \in Y} exp(\sum \lambda_i f_i(X, y))}$$

"exponential model"

- Next: **Estimate** the λ weights (parameters) that **maximize** the conditional **likelihood** of this logistic model (**MLE**)

Maximum Entropy (finally…)

- MaxEnt is all about selecting the "maximal" model p*:

$$p^* = \underset{p \in P}{argmax} \; -\sum_{x \in X} p(x) \sum_{y \in Y} p(y|x) \; log_2 \; p(y|x)$$

Find a classifier that maximizes the conditional entropy by selecting …

- That obeys a **conditional** equality **constraint** on the feature probabilities:

$$\sum_{x \in X} P(x) \sum_{y \in Y} P(y|x) \; f(x,y) = \sum_{x \in X, y \in Y} P(x,y) \; f(x,y)$$

… a conditional probability model that matches the observed joint probabilities.

- Using, e.g., **Langrange multipliers**, one can calculate the optimal λ weight parameters of the **model** that maximize the entropy of this probability:

$$p^*(y|X) = \frac{exp(\sum \lambda_i f_i(X,y))}{\sum_{y \in Y} exp(\sum \lambda_i f_i(X,y))}$$

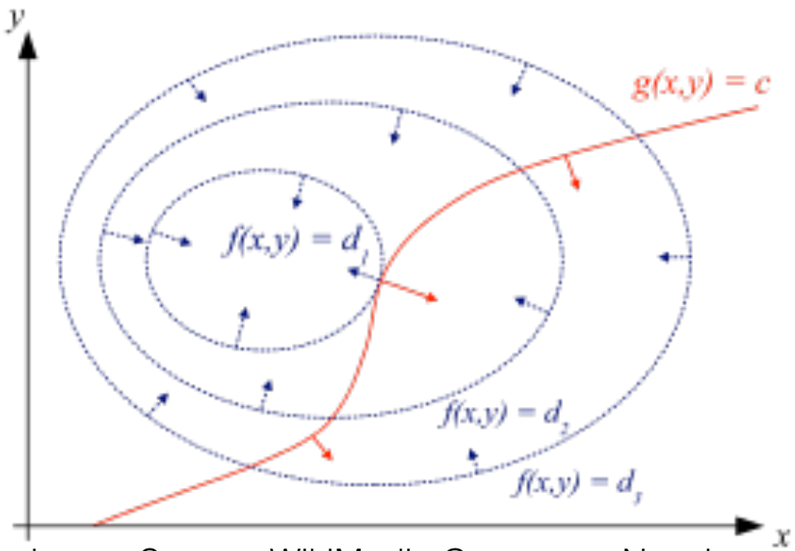


Image Source: WikiMedia Commons, Nexcis

MaxEnt vs. naïve Bayes

(All features) MaxEnt model (as observed)

Lights Working

Lights Broken

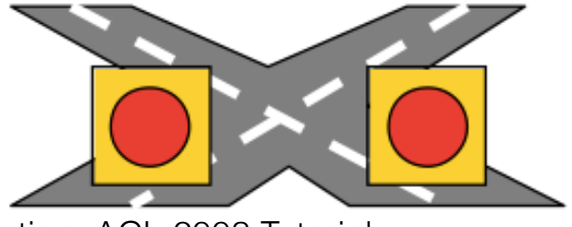


Image Source: Klein & Manning. Maxent Models, Conditional Estimation, and Optimization. ACL 2003 Tutorial

P(g,r,w) = 3/7 P(r,g,w) = 3/7 P(r,r,b) = 1/7

MaxEnt adjusts the Langrange multipliers (weights) to **model** the observed **joint probabilities**.

Note that the example has dependent features: the two stoplights!

Naïve Bayes model

- P(**w**) = 6/7
- P(r|**w**) = 1/2
- P(g|**w**) = 1/2
- P(**b**) = 1/7
- P(r|**b**) = 1
- P(g|**b**) = 0

✓ P(r,r,**b**) = (1/7)(1)(1) = 2/14

✓ P(r,g,**b**) = P(g,r,**b**) = 0

✓ P(g,g,**b**) = 0

- P(*,*,**w**) = (6/7)(1/2)(1/2) = 3/14

P(g,g,w) = 3/14

P(r,r,w) = 3/14 > P(r,r,b) = 0

But even MaxEnt cannot detect feature interaction

NB: feature dependence != feature interaction

Empirical (joint) observations

A, B	a_r	a_g
b_r	1	3
b_g	3	0

↓

Correct (target) distribution

A, B	a_r	a_g
b_r	1/7	3/7
b_g	3/7	0

2 feature model: Only A or B considered

A, B	a_r	a_g
b_r		
b_b		

only A=r

A, B	a_r	a_g
b_r	4/14	3/14
b_g	4/14	3/14

only B=r

A, B	a_r	a_g
b_r	4/14	4/14
b_g	3/14	3/14

A, B	a_r	a_g
b_r	16/49	12/49
b_g	12/49	9/49

4 feature model: All A,B combos

A, B	a_r	a_g
b_r		
b_g		

With a huge V, that creates a combinatorial explosion….

… and data sparsity issues

A, B	a_r	a_g
b_r	1/7	3/7
b_g	3/7	0

Klein & Manning. MaxEnt Models, Conditional Estimation and Optimization. ACL 2003

Evaluation metrics for classification tasks

Evaluations should answer questions like:

How to measure a change to an approach?

Did adding a feature improve or decrease performance?

Is the approach good at locating the relevant pieces or good at excluding the irrelevant bits?

How do two or more different methods compare?

Essential evaluation metrics: Accuracy, f-measure, MCC score

Patient→ Doctor↓	has cancer	is healthy
diagnose cancer	TP	FP
detects nothing	FN	TN

FP: type I...
FN: type II...
...errors

- **Precision** (P; Pos. Pred. Value)
  - correct hits [TP] ÷ all hits [TP + FP]
- **Recall** (R; **Sensitivity**, TP Rate)
  - correct hits [TP] ÷ true cases [TP + FN]
- **Specificity** (True Negative Rate)
  - correct misses [TN] ÷ negative cases [FP + TN]

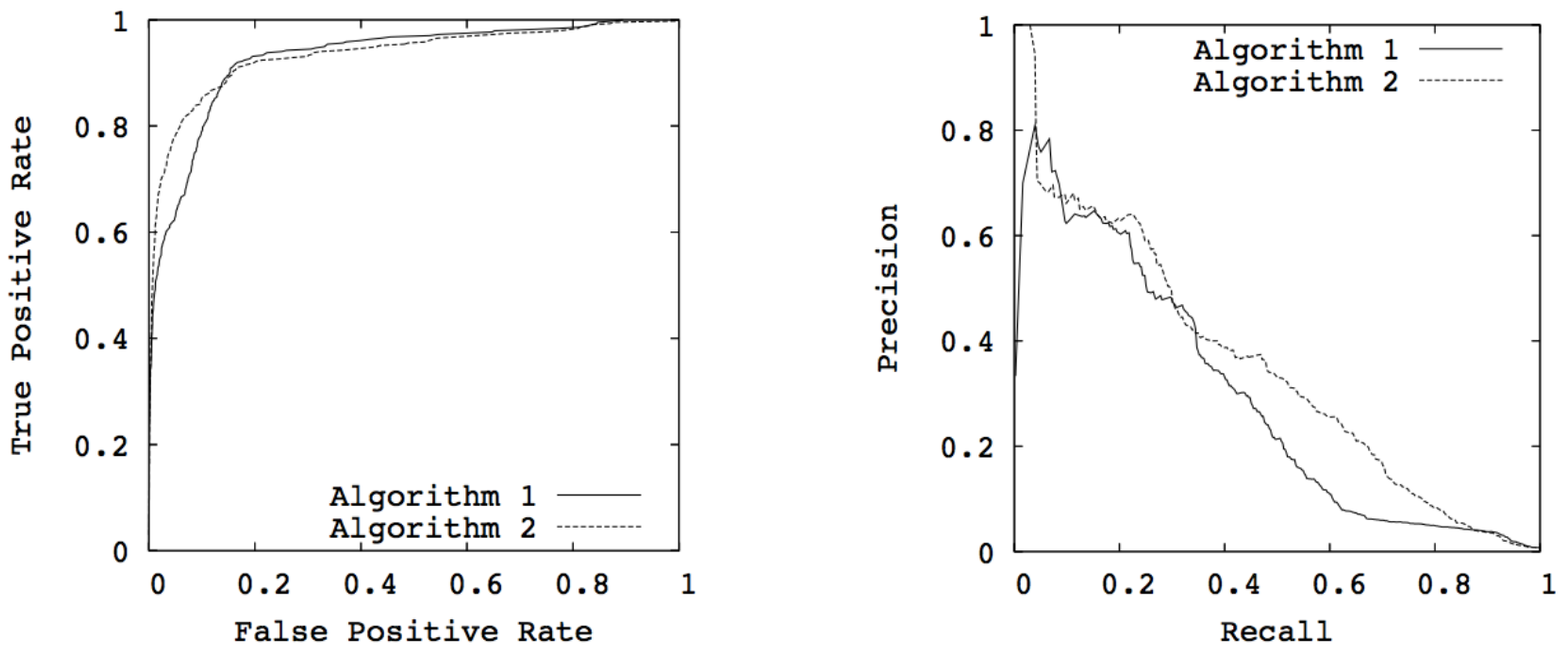
Negative Predictive Value: TN ÷ (FN + TN)
NB: errors might have differential costs!

- **Accuracy**
  - correct classifications [TP + TN] ÷ all cases [TP + TN + FN + FP])
  - highly **sensitive to** class **imbalance**
- **F-Measure** (F-Score)
  - the harmonic mean between P & R
    = 2 TP ÷ (2 TP + FP + FN)
    = (2 P R) ÷ (P + R)
  - does **not** require a **TN** count
- **MCC Score** (Mathew's Correlation Coefficient)
  - χ^2**-based**: (TP TN - FP FN) ÷ sqrt[(TP+FP)(TP+FN)(TN+FP)(TN+FN)]
  - **robust against** class **imbalance**

NB: no measure takes result order into account

Ranked evaluation results: AUC ROC and PR

Area Under the Curve
Receiver-Operator Characteristic
Precision-Recall

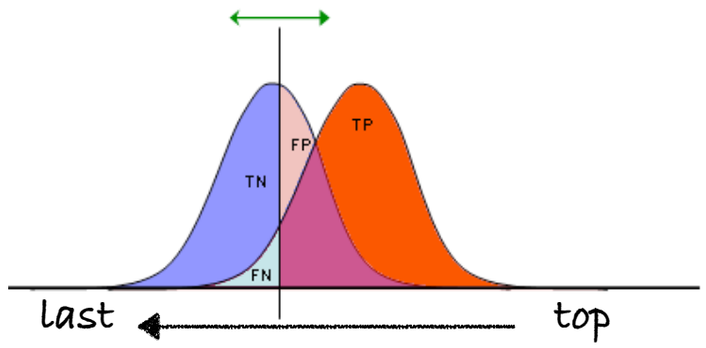


Davis & Goadrich.
ICML 2006

TPR / Recall (aka. Sensitivity)
TP ÷ (TP + FN)

FPR (not Specificity!)
FP ÷ (TN + FP)

Precision
TP ÷ (TP + FP)



TP	FP
FN	TN
1	1

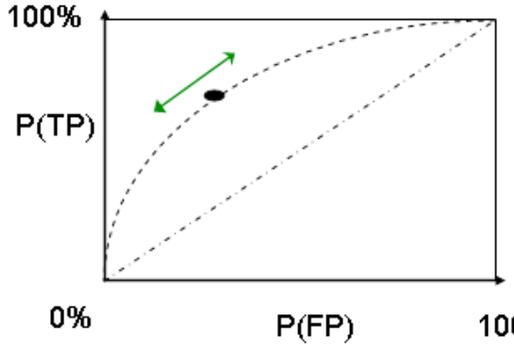


Image Source: WikiMedia Commons, kku ("kakau", eddie)

To ROC or to PR?

Curve I:
10 hits in the top 10, and 10 hits spread over the next 1500 results.

AUC ROC
0.813

Results: 20 T ≪ 1980 N

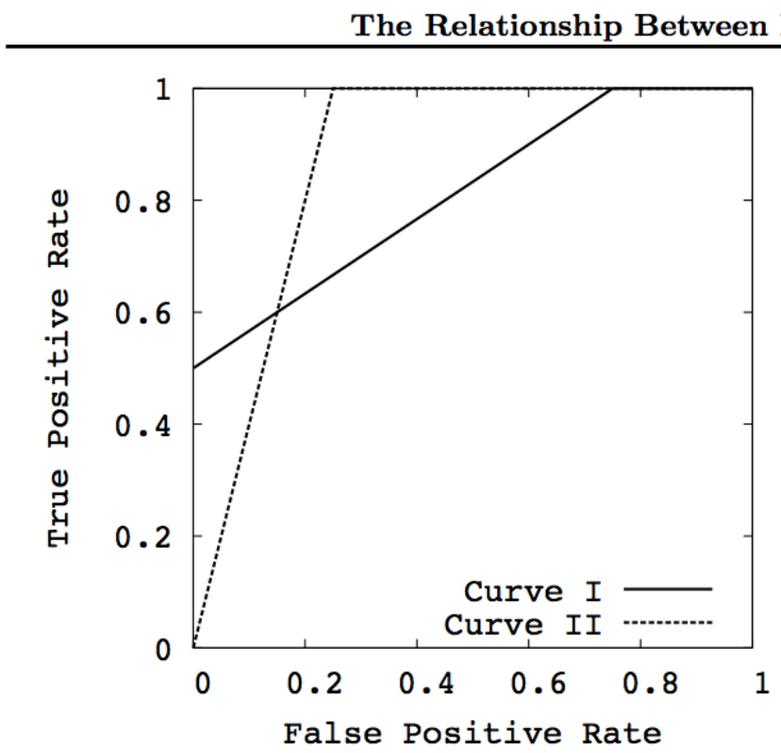


Figure 11. Comparing AUC-ROC for Two Algorithms

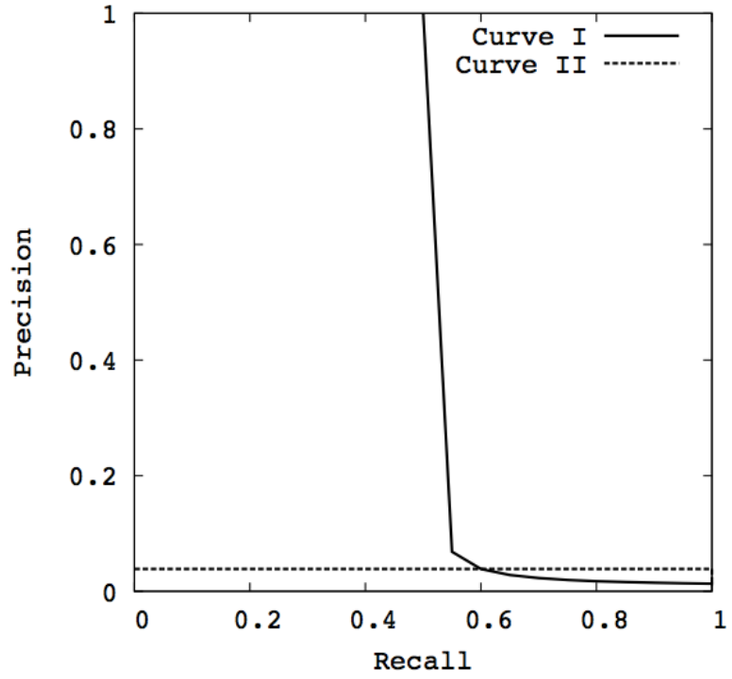


Figure 12. Comparing AUC-PR for Two Algorithms

Curve II:
Hits spread evenly over the first 500 results.

AUC ROC
0.875

"An algorithm which optimizes the area under the ROC curve is not guaranteed to optimize the area under the PR curve."

Davis & Goadrich, 2006

- Davis & Goadrich. The Relationship Between PR and ROC Curves. ICML 2006
- Landgrebe et al. Precision-recall operating characteristic (P-ROC) curves in imprecise environments. Pattern Recognition 2006
- Hanczar et al. Small-Sample Precision of ROC-Related Estimates. Bioinformatics 2010

→ Use a FPR-reweighted ROC for **imbalanced** ranking scenarios!*

Text classification approaches

efficient/fast training

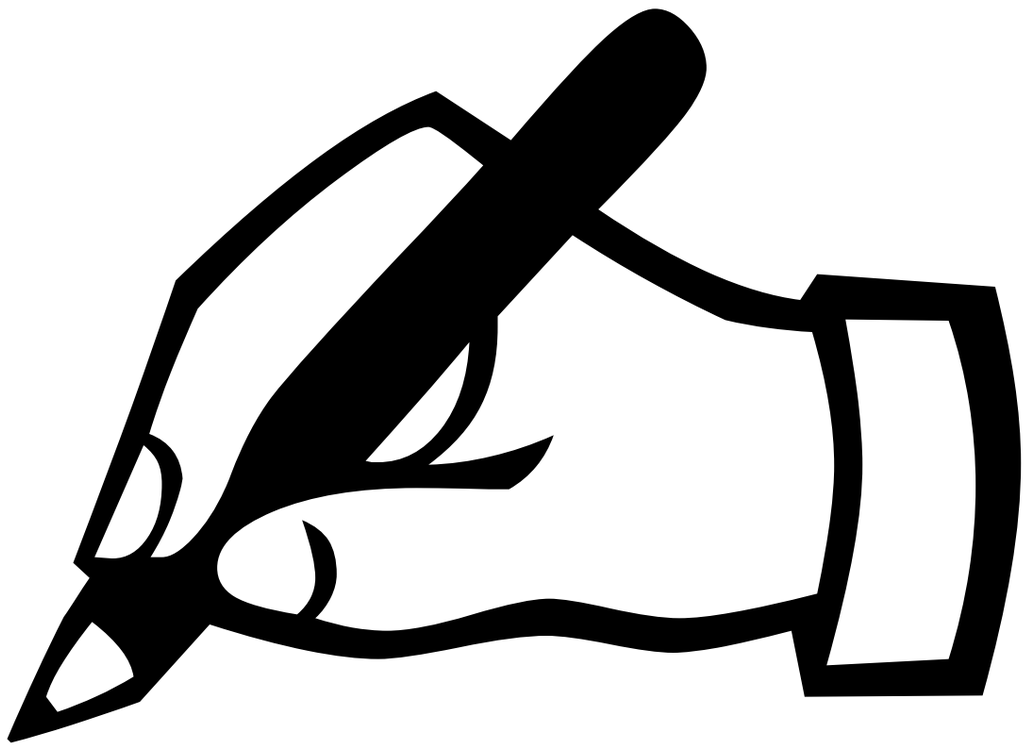
- **Multinomial naïve Bayes***
- **Nearest neighbor classification** (ASDM Course: Supervised Pattern R.)
  - incl. **locality sensitive hashing***
- **Latent semantic indexing***
- **Cluster analysis** (ASDM Course: Unsupervised Pattern Recognition)
- **Max. entropy classification** (multinomial logistic or softmax regression)*
- **Latent Dirichlet allocation***

high accuracy

- **Random forest classification** (ASDM Course: Feature subset selection)
- **Support vector machines** (ASDM Course: Support Vector Machines)
- **Gradient boosting machines** (e.g., with xgboost)
- **Deep learning** (ASDM Course: Neural Networks and Deep Learning)

* this course

Practical 7:
A TF-IDF-based text classifier



Unsupervised Methods

Madrid Summer School on
Advanced Statistics and Data Mining

Florian Leitner
Selerity, Inc.; Data Catalytics, SL; IE University
leitner@datacatytics.com

String similarity

- Finding similarly spelled or misspelled words
- Finding "similar" texts/documents
  - N.B.: no semantics (yet…)
- Detecting entries in a **gazetteer**
  - a list of domain-specific words
- Detecting **entities** in a **dictionary**
  - a list of words, each mapped to some **URI**
  - N.B.: "entities", not "concepts" (no semantics…)

String similarity measures

- Edit Distance Measures
  - Hamming Distance [1950]
  - Levenshtein-Damerau Distance [1964/5]
  - Needelman-Wunsch [1970] and Smith-Waterman [1981] Distance
    - also **align** the strings ("sequence alignment")
    - use **dynamic programming**
    - Modern approaches: BLAST [1990], BWT [1994]
- Other Similarity Metrics
  - Jaro-Winkler Similarity [1989/90]
    - coefficient of matching characters within a dynamic window minus transpositions
  - Soundex (→ homophones; spelling!)
  - …

- Basic Operations: "indels"
  - Insertions
    - ac → abc
  - Deletions
    - abc → ac
- "Advanced" Operations
  - Require two "indels"
  - Substitutions
    - abc → aBc
  - Transpositions
    - ab → ba

typos!

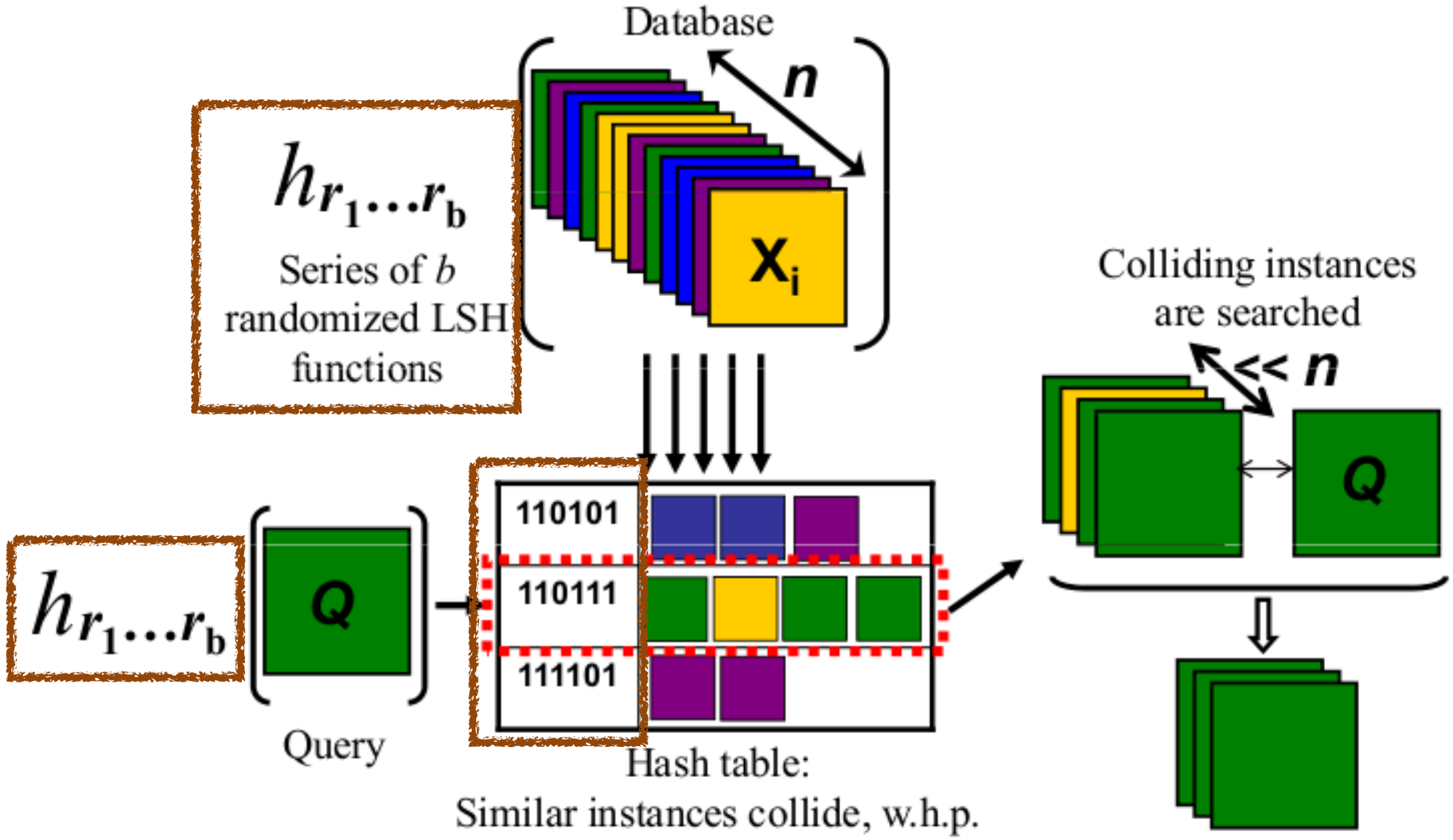
String distance measures

- Hamming Distance
  - only counts **substitutions**
  - requires **equal** string **lengths**
  - karolin ↔ kathrin = 3
  - karlo ↔ carol = 3
  - karlos ↔ carol = undef
- Levenshtein Distance
  - counts **all but transpositions**
  - karlo ↔ carol = 3
  - karlos ↔ carol = 4
- **Damerau**-Levenshtein D.
  - also allows **transpositions**
  - has **quadratic complexity**
  - karlo ↔ carol = 2
  - karlos ↔ carol = 3
- Jaro Similarity (a, b)
  - calculates $(m/|a| + m/|b| + (m-t)/m) / 3$
  - m → the # of matching chars in...
  - t → the # of transpositions in...
  - ...**window**: $\lfloor \max(|a|, |b|) / 2 \rfloor - 1$ chars
  - karlos ↔ carol = 0.74 [0,1] range
    - (4 ÷ 6 + 4 ÷ 5 + 3 ÷ 4) ÷ 3
- Jaro-**Winkler** Similarity
  - adds a bonus for matching prefixes

Locality Sensitive Hashing: Overview

- A **hashing** approach to group **near neighbors**.
- Map similar items (e.g., documents or words) into the same [hash] buckets.
- Detection of spelling errors, plagiarism, or near-duplicates.
- LSH "**maximizes**" (instead of minimizing) hash **collisions**.
- It therefore is a **dimensionality reduction** technique.
- For indexed values (n-grams), **minhashing** can be used.
  - Approach from Rajaraman & Ullman, Mining of Massive Datasets, 2010
  - http://infolab.stanford.edu/~ullman/mmds/ch3a.pdf

Locality Sensitive Hashing: (Min-)Hash signatures



M Vogiatzis. micvog.com 2013.

Minhash signatures (1/2)

Create a binary n-gram/term × word/doc matrix (likely **very** sparse!):

{ **T**: if term/n-gram in doc/word
F: if term/n-gram not in doc/word

Shingle ID	x	T_1	T_2	T_3	T_4	h_1	h_2
	0	**T**	F	F	**T**	1	1
	1	F	F	**T**	F	2	4
	2	F	**T**	F	**T**	3	2
	3	**T**	F	**T**	**T**	4	0
	4	F	F	**T**	F	0	3

Step 1/2 - permuting the row order:

$h_1(x) = (x+1)\%n$

$h_2(x) = (3x+1)\%n$

with n=5 (rows)

hash-"permuted" row IDs

Lemma: Two docs/terms will have a coinciding "true" term/n-gram when comparing shingles with a probability equal to their Jaccard Similarity.

a family of hash functions h_i

Approach: Create ~~sufficient~~ **permutations** of the row (term/n-gram) ordering so that the Jaccard Similarity can be approximated by comparing the number of coinciding vs. differing rows.

Minhash signatures (2/2)

Shingle ID

S	T_1	T_2	T_3	T_4	h_1	h_2
0	T	F	F	T	1	1
1	F	F	T	F	2	4
2	F	T	F	T	3	2
3	T	F	T	T	4	0
4	F	F	T	F	0	3

$h_1(x) = (x+1)\%n$

$h_2(x) = (3x+1)\%n$

from step one

$n=5$

Step 2/2 - generating the hash signature:

(vertical, per-document)

minhash signatures:

M	T_1	T_2	T_3	T_4
h_1	∞	∞	∞	∞
h_2	∞	∞	∞	∞

perfectly map-reduce-able
and embarrassingly parallel!

```
init matrix M = ∞

for each shingle-id s:
 for each hash-function h:
  for each text-id t:
   if S[t,s] and M[t,h] > h(s):
    M[t,h] = h(s)
```

Minhash signatures (2/2)

Shingle ID

S	T_1	T_2	T_3	T_4
0	T	F	F	T
1	F	F	T	F
2	F	T	F	T
3	T	F	T	T
4	F	F	T	F

h_1	h_2
1	1
2	4
3	2
4	0
0	3

$h_1(x) = (x+1)\%n$

$h_2(x) = (3x+1)\%n$

from step one

$n=5$

Step 2/2 - generating the hash signature:

(vertical, per-document)

minhash signatures:

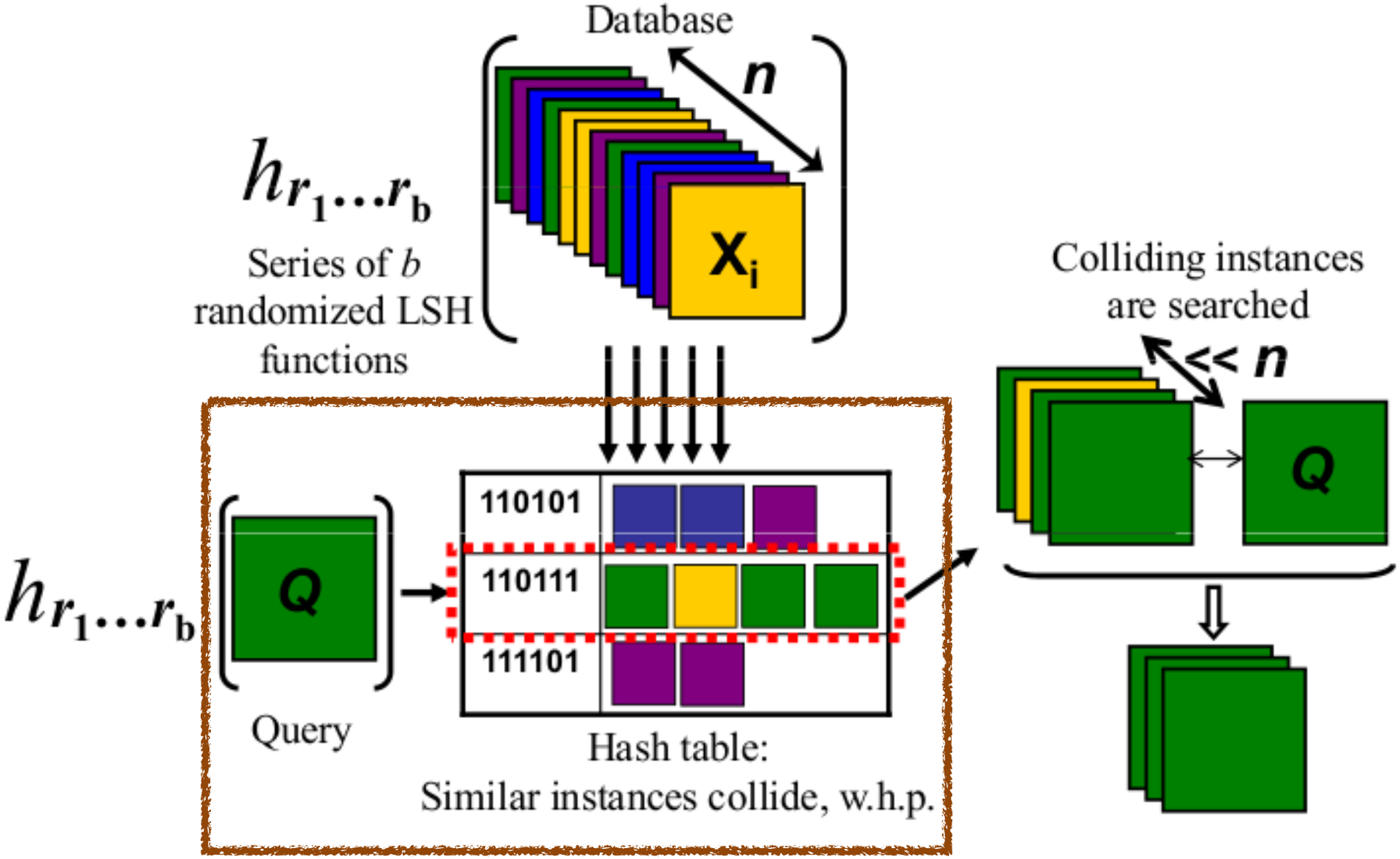
M	T_1	T_2	T_3	T_4
h_1	1	3	**0**	1
h_2	0	2	0	0

perfectly map-reduce-able and embarrassingly parallel!

```
init matrix M = ∞

for each shingle-id s:
 for each hash-function h:
  for each text-id t:
   if S[t,s] and M[t,h] > h(s):
    M[t,h] = h(s)
```

Locality Sensitive Hashing: Fine-tuning the collisions



M Vogiatzis. micvog.com 2013.

Banded locality sensitive min-hashing

Bands		T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	...
1	h_1	**1**	0	2	**1**	7	1	4	5	
	h_2	**1**	2	4	**1**	6	5	5	6	
	h_3	**0**	5	6	**0**	6	4	7	9	
2	h_5	4	0	8	8	**7**	6	5	**7**	
	h_1	7	7	0	8	**3**	8	7	**3**	
	h_4	8	9	0	7	**2**	4	8	**2**	
3	h_2	8	5	**4**	0	9	8	**4**	7	
	h_6	9	4	**3**	9	0	8	**3**	9	
	h_7	8	5	**8**	0	0	6	**8**	0	
...	...									

Bands $b \propto p_{agreement}$

Hashes/Band $r \propto 1/p_{agreement}$

(in at least one band) $p_{agreement} = \mathbf{1 - (1 - s^r)^b}$

typical: r=10 b=30

s = Jaccard(A,B)

see Notebook 06, In cell [10]

Union Find: An algorithm to group texts across bands

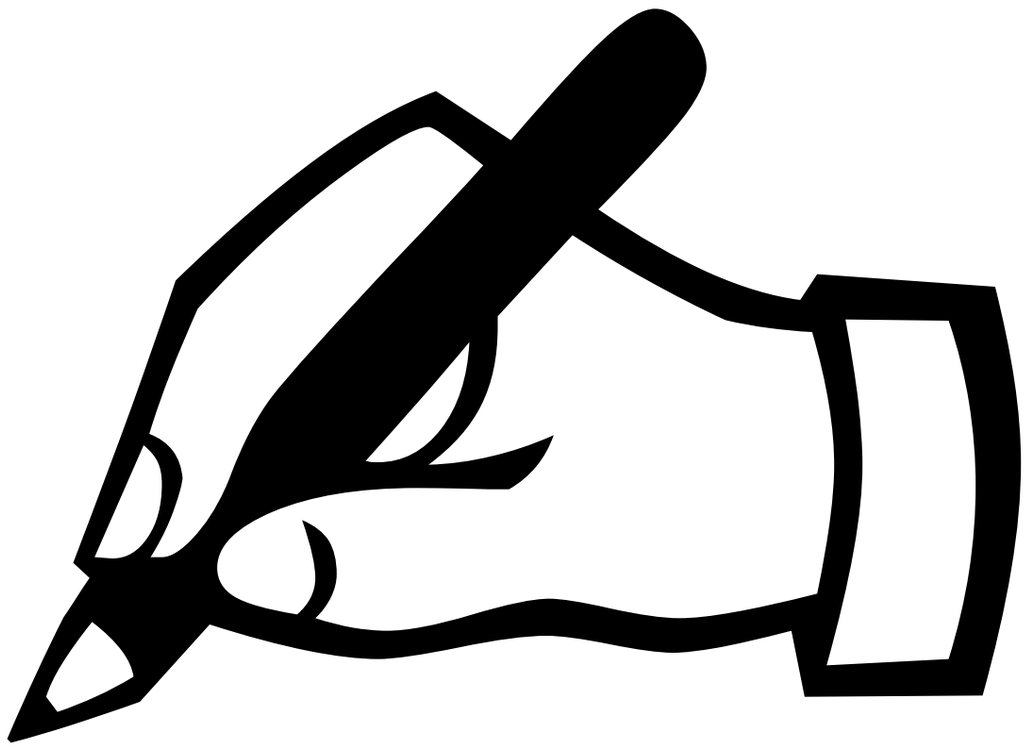
	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	...
h_1	0	**1**	2	7	**1**	**1**	4	5	
h_2	2	**5**	4	6	**5**	**5**	5	6	
h_3	5	**0**	6	6	**0**	**0**	7	9	
h_5	4	0	**2**	8	8	**2**	5	5	
h_1	7	7	**1**	8	0	**1**	7	3	
h_6	8	9	**6**	7	0	**6**	8	8	
h_2	8	5	**4**	**4**	9	8	**4**	**4**	
h_5	9	4	**3**	**3**	0	1	**3**	**3**	
h_4	8	5	**8**	**8**	0	6	**8**	**8**	
...									

new "connection":
{ 2 5 }

Document clusters

{0}, {1 4 5}, {2 3 6 7} —union (2,5)→ {0}, {1 2 3 4 5 6 7}

Practical 6: Document indexing



Sentence segmentation

- Sentences are **the** fundamental linguistic unit
  - Sentences are the boundaries or "constraints" for linguistic phenomena.
  - **Collocations** ["United Kingdom", "vice president"], idioms ["drop me a line"], phrases [e.g., the preposition phrase "of great fame"], clauses, statements, … all occur **within** a sentence.
- Rule/pattern-based segmentation
  - Segment sentences if the marker is followed by an upper-case letter
  - Works well for "clean text" (news articles, books, papers, …)
  - **Special cases**: abbreviations, digits, lower-case proper nouns (genes, "amnesty international", …), hyphens, quotation marks, …
- Supervised sentence boundary detection
  - Use some Markov model or a conditional random field to identify possible sentence segmentation tokens
  - Requires labeled examples (segmented sentences)

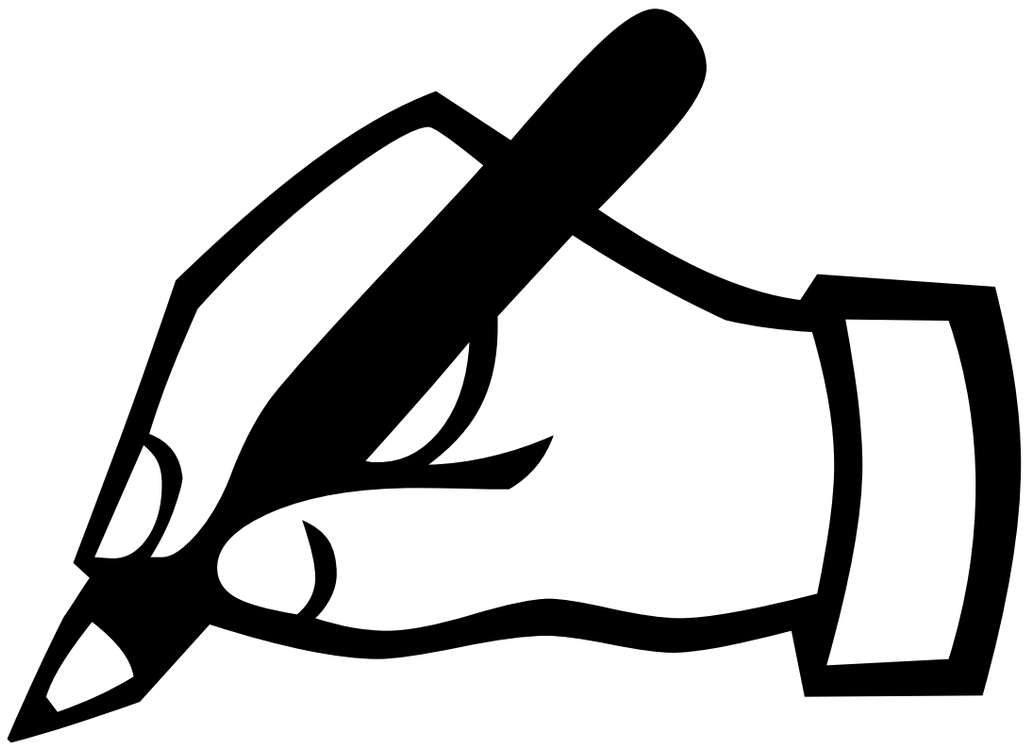
Punkt Sentence Tokenizer (PST): Rules

- Unsupervised sentence boundary detection
- $P(\bullet|w_{-1}) > c_{cpc}$ Dr.
  - Determines if a marker ● is used as an **abbreviation** marker by comparing the **conditional probability** that the word w_{-1} before ● is followed by the marker against some (high) cutoff probability c_{cpc}.
    - $P(\bullet|w_{-1}) = P(w_{-1}, \bullet) \div P(w_{-1})$
    - K&S set $c = 0.99$
- $P(w_{+1}|w_{-1}) > P(w_{+1})$ Mrs. Watson
  - Evaluates the likelihood that w_{-1} and w_{+1} surrounding the marker ● are more commonly collocated than would be expected by chance: ● is assumed an **abbreviation** marker ("not independent") if the LHS is greater than the RHS.
- $F_{length}(w) \times F_{markers}(w) \times F_{penalty}(w) \geq c_{abbr}$ U.S.A.
  - Evaluates if any of w's morphology (length of w w/o marker characters, number of periods inside w (e.g., ["U.S.A"]), penalized when w is not followed by a ●) makes it more likely that w is an abbreviation against some (low) cutoff.
- $F_{ortho}(w)$; $P_{sstarter}(w_{+1}|\bullet)$; … . Therefore
  - Orthography: lower-, upper-case or capitalized word after a probable ● or not
  - Sentence Starter: Probability that w is found after a ●

Punkt Sentence Tokenizer (PST): Summary

- Unsupervised Multilingual Sentence Boundary Detection
  - Kiss & Strunk, MIT Press 2006.
  - Available from NLTK: `nltk.tokenize.punkt` (http://www.nltk.org/api/nltk.tokenize.html)
- PST is language agnostic
  - Requires that the language uses the sentence segmentation marker as an abbreviation marker
  - Otherwise, the problem PST solves is not present
- PST factors in word length
  - Abbreviations are relatively shorter than regular words
- PST takes "internal" markers into account
  - E.g., "U.S.A"
- Main weakness: long lists of abbreviations
  - E.g., author lists in citations
  - Can be fixed with a pattern-based post-processing strategy
- NB: a marker must be present
  - E.g., chats or fora

Practical 9:
Sentence segmentation



Segmenting terms: Collocation extraction

- After splitting and tokenizing sentences, one critical segmentation step remains: Collocation extraction.
  - Working only with tokens leads to a loss of meaning (e.g., “black” and “tee”).
  - Extracting all possible n-grams produces lots of noise (e.g., “produces lots”).
- Extracting only meaningful n-grams, also known as **collocations**, would solve both issues.
- Collocations can be detected statistically in an unsupervised fashion.
  - The noise from false positive collocations is completely outweighed by the benefits of having them.

Detecting collocations: standard statistical methods

- Frequency-based methods
  - Measure the likelihood of bigram co-occurrence to detect "collocation-ness"

- **t-test** (using the **expected** probability p of a candidate bigram as the variance σ^2)
  - has problems with more frequent words and is typically only used to **rank** collocations (rather than detect them)
- **χ^2-test** (using the ratios of either word being pre-/proceeded by the other)
  - has problems when the frequency of words being compared is very small
- **PMI** (point-wise mutual information; log-ratio of the joint over the expected probability of the bigram)
  - as the other tests, has problems with data sparseness (i.e., bigrams with few examples; more robust: frequency-weighted PMI)

Collocations: Point-wise mutual information

- **Mutual information** (MI) measures the degree of dependence of two variables: how similar is $P(X,Y)$ to $P(X) \cdot P(Y)$?

"how much you can learn about X from knowing Y"

$$I(X;Y) = \sum_Y \sum_X P(x,y)\ log_2 \frac{P(x,y)}{P(x)P(y)}$$

- **Point-wise MI**: MI of two **individual** events [only]

$$PMI(w_1,w_2) = log_2 \frac{P(w_1,w_2)}{P(w_1)P(w_2)}$$

  - e.g., neighbouring words, phrases in a document, …
  - incl. a **mix of** two different co-occurring **event types** (e.g. a word and a phrase or label)
  - Can be normalized to a $[-1,1]$ range: $\frac{PMI(w_1,w_2)}{-log_2\ P(w_1,w_2)}$

- Interpretation: **-1**: the events do not occur together; **0**: the events are independent; **+1**: the events always co-occur

Collocations: The likelihood ratio test

- MLE assuming a **binomial distribution of bigrams**
- For bigram "word$_1$ word$_2$", with counts c and total words N:

$$\lambda = -2\ log \frac{B(c_{12}, c_1, p_2)B(c_2 - c_{12}, N - c_1, p_2)}{B(c_{12}, c_1, p_{2|1})B(c_2 - c_{12}, N - c_1, p_{2|\neg 1})}$$

$$B(k, n, p) = \binom{n}{k} p^k (1-p)^{n-k} \qquad p_2 = \frac{c_2}{N} \qquad p_{2|1} = \frac{c_{12}}{c_1} \qquad p_{2|\neg 1} = \frac{c_2 - c_{12}}{N - c_1}$$

- Asymptotically χ^2-distributed (explaining the "-2" multiplier)
- Can use the χ^2 statistic to find the critical values for λ

  NB: d.f. is always 1 (→bigrams!); $\lambda > 10.83$ is pretty solid

- Fairly **robust** with both frequent words and infrequent bigrams

  However, your instructor recommends PMI, as it generally works better, in terms of practical results.

  Dunning. Accurate methods for the statistics of surprise and coincidence. 1993, Comp. Ling.

Detecting collocations: phrasal filtering rules

- Select only terms that are valid noun and verb phrases
  - NB: not all collocations are noun phrases, e.g. "to make up".
  - Advantage: Can be applied after any statistical selection method.
  - Disadvantage: Requires tagging tokens with their part-of-speech.
    - i.e., this is a fully supervised approach
    - [Part-of-speech tagging is coming up later in this lecture]
  - Justeson & Katz, Comp. Ling., 1995, suggested the following rules:
    - [Adjective | Noun] - [Noun] → "strong tea", "noun phrase"
    - [Adjective] - [Adjective | Noun] - Noun → "free legal advice", "online web tutorial"
    - Noun - [Adjective | Noun] - Noun → "mean squared error", "New York City"
    - Noun - Preposition - Noun → "parts of speech", "rich and powerful"
      NB: no rules for verb phrases were suggested (far more tricky)

From syntactic to semantic similarity

Cosine Similarity, χ^2, Spearman's ρ, LSH, etc. all compare equal tokens.

But what if you are talking about "automobiles" and I am lazy, calling it a "car"?

We can solve this with Latent Semantic Indexing!

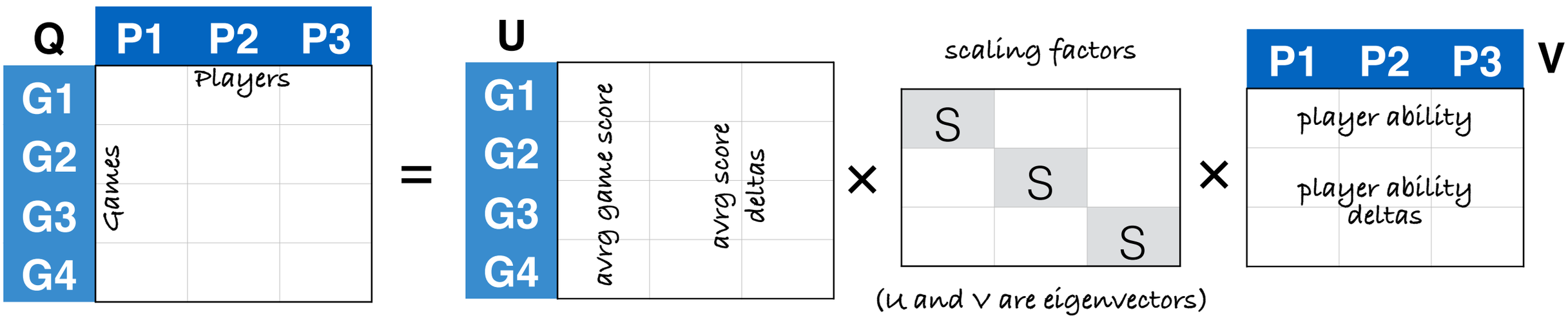
Latent Semantic Analysis (LSA 1/3)

- a.k.a. Latent Semantic **Indexing** (in Text Mining): **feature extraction** for **semantic inference**
- Linear algebra background
  - Singular value decomposition of a matrix Q: $\mathbf{Q} = \mathbf{U\Sigma V^T}$

orthonormal factors of Q (QQ^T and Q^TQ)

singular values: scaling factor

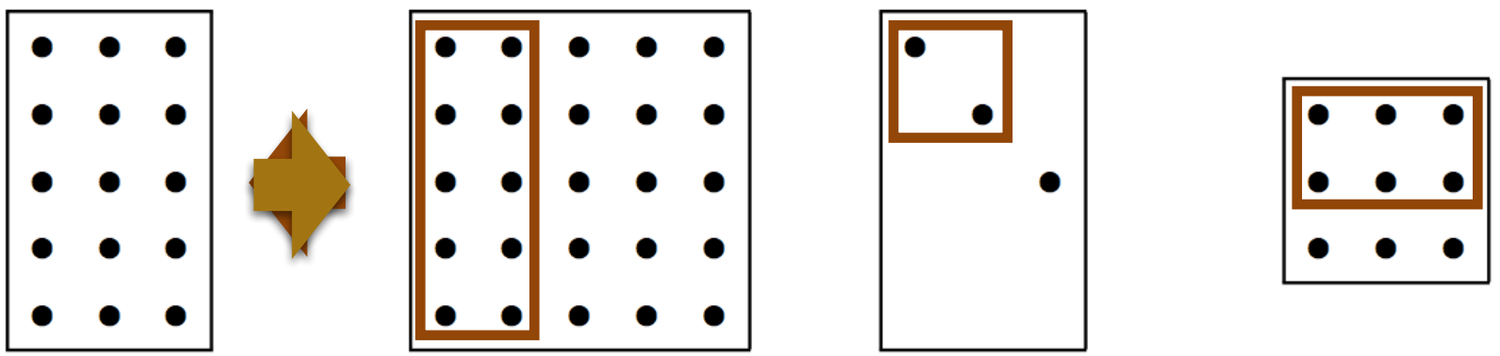
the factors "predict" Q in terms of similarity (Frobenius norm) using as many factors as the lower dimension of Q



- SVD in text mining
  - Inverted index = doc. eigenvectors × singular values × term eigenvectors

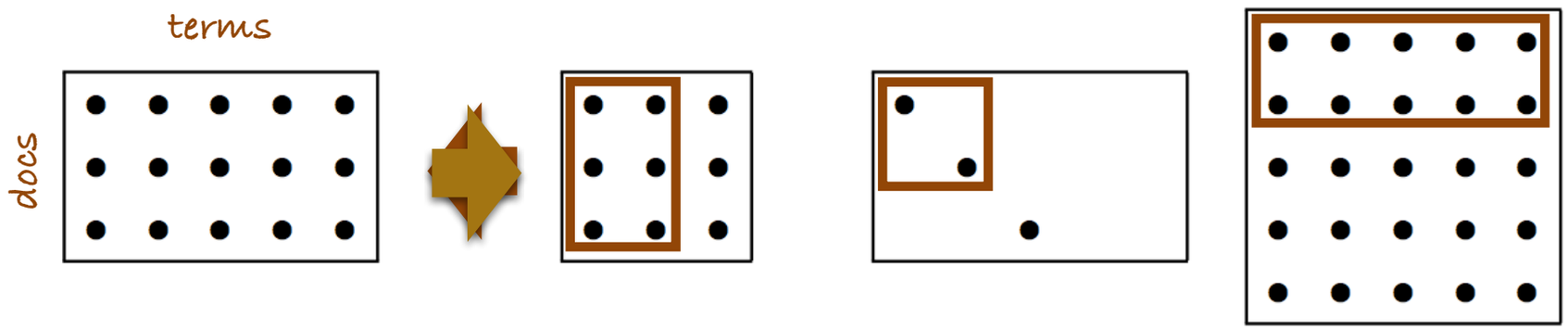
Latent Semantic Analysis (LSA 2/3)

C = $\hat{F}$eat. extraction by selecting only the largest n eigenvalues



- Inverted index = doc. eigenvectors × singular values × term eigenvectors

$$C = U \Sigma V^T$$



- Image taken from: Manning et al. An Introduction to IR. 2009

Latent Semantic Analysis (LSA 3/3)

c1: *Human* machine *interface* for ABC *computer* applications
c2: A *survey* of *user* opinion of *computer system response time*
c3: The *EPS user interface* management *system*
c4: *System* and *human system* engineering testing of *EPS*
c5: Relation of *user* perceived *response time* to error measurement

m1: The generation of random, binary, ordered *trees*
m2: The intersection *graph* of paths in *trees*
m3: *Graph minors* IV: Widths of *trees* and well-quasi-ordering
m4: *Graph minors*: A *survey*

C

[Spearman's] rho(human, user) = -0.38
rho(human, minors) = -0.29

	c1	c2	c3	c4	c5	m1	m2	m3	m4
human	1	0	0	1	0	0	0	0	0
interface	1	0	1	0	0	0	0	0	0
computer	1	1	0	0	0	0	0	0	0
user	0	1	1	0	1	0	0	0	0
system	0	1	1	2	0	0	0	0	0
response	0	1	0	0	1	0	0	0	0
time	0	1	0	0	1	0	0	0	0
EPS	0	0	1	1	0	0	0	0	0
survey	0	1	0	0	0	0	0	0	1
trees	0	0	0	0	0	1	1	1	0
graph	0	0	0	0	0	0	1	1	1
minors	0	0	0	0	0	0	0	1	1

top 2 dim

Ĉ

	c1	c2	c3	c4	c5	m1	m2	m3	m4
human	0.16	0.40	0.38	0.47	0.18	-0.05	-0.12	-0.16	-0.09
interface	0.14	0.37	0.33	0.40	0.16	-0.03	-0.07	-0.10	-0.04
computer	0.15	0.51	0.36	0.41	0.24	0.02	0.06	0.09	0.12
user	0.26	0.84	0.61	0.70	0.39	0.03	0.08	0.12	0.19
system	0.45	1.23	1.05	1.27	0.56	-0.07	-0.15	-0.21	-0.05
response	0.16	0.58	0.38	0.42	0.28	0.06	0.13	0.19	0.22
time	0.16	0.58	0.38	0.42	0.28	0.06	0.13	0.19	0.22
EPS	0.22	0.55	0.51	0.63	0.24	-0.07	-0.14	-0.20	-0.11
survey	0.10	0.53	0.23	0.21	0.27	0.14	0.31	0.44	0.42
trees	-0.06	0.23	-0.14	-0.27	0.14	0.24	0.55	0.77	0.66
graph	-0.06	0.34	-0.15	-0.30	0.20	0.31	0.69	0.98	0.85
minors	-0.04	0.25	-0.10	-0.21	0.15	0.22	0.50	0.71	0.62

test # dim to use via synonyms or missing words

From: Landauer et al. An Introduction to LSA. 1998

rho(human, user) = **0.94**
rho(human, minors) = **-0.83**

Principal Component vs. Latent Semantic Analysis

best Frobenius norm: minimizes the "std. dev." of the matrix: the "dot product (or L2 norm) of matrices";
best affine subspace: minimize the dimensions while maintaining the (parallel) direction

- **LSA** seeks for the **best linear subspace** in **Frobenius norm**, while **PCA** aims for the **best affine linear subspace**.
- **LSA can use** TF-IDF weighting as **preprocessing** step.
- **PCA requires the** (square) **covariance matrix** of the original matrix as its first step and therefore can only compute term-term or doc-doc similarities.
- **PCA matrices are more dense** (zeros occur only when true independence is detected).

A first look at probabilistic graphical models

- Latent Dirichlet Allocation: LDA (not to be confused with LDA: linear discriminant analysis)
  - Blei, Ng, and Jordan. Journal of Machine Learning Research 2003
  - For assigning "topics" to "documents" i.e., for text classification
  - An **unsupervised**, **generative** model

Latent Dirichlet Allocation (LDA 1/3)

- Intuition for LDA
  - From: Edwin Chen. Introduction to LDA. 2011
  - "Document Collection"

• I like to eat broccoli and bananas. • I ate a banana and spinach smoothie for breakfast.	➡ Topic A
• Chinchillas and kittens are cute. • My sister adopted a kitten yesterday.	➡ Topic B
• Look at this cute hamster munching on a piece of broccoli.	➡ Topic 0.6A + 0.4B

- Topic A: 30% broccoli, 15% bananas, 10% breakfast, 10% munching, …
- Topic B: 20% chinchillas, 20% kittens, 20% cute, 15% hamster, …

The Dirichlet process

A Dirichlet process is like drawing from an (infinite) "bag of dice" (with finite faces).
Alternative view: Chinese restaurant process:
New customers sit at a table with a p proportional to customers already at each table.
And, customers open a new table with a probability proportional to α. (NB: infinite tables and seats…)

- A Dirichlet is a [possibly continuos] **distribution over** [discrete/multinomial] **distributions** (probability **masses**).

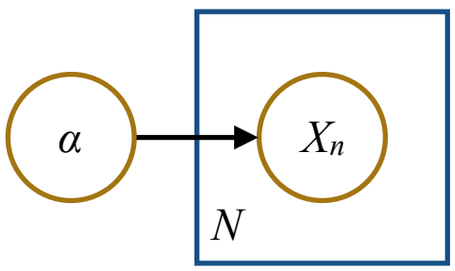
Gamma function -> a "continuous" factorial [!]

$$D(\boldsymbol{\theta}, \boldsymbol{\alpha}) = \frac{\Gamma(\sum \alpha_i)}{\prod \Gamma(\alpha_i)} \prod \theta_i^{\alpha_i - 1}$$

a Dirichlet prior: $\forall \alpha_i \in \alpha: \alpha_i > 0$

$\sum \theta_i = 1$; a Probability Mass Function

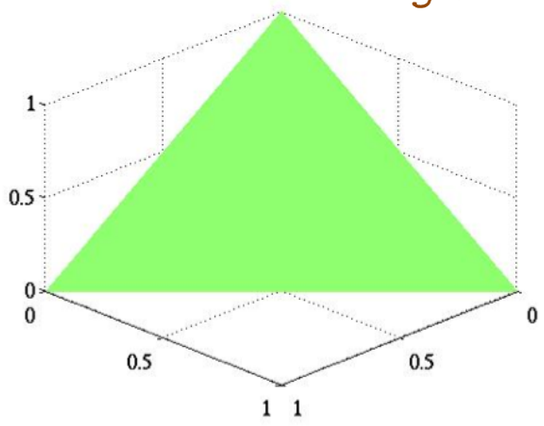
- The **Dirichlet Process samples** multiple independent, discrete **distributions** θ_i with repetition from $\boldsymbol{\theta}$ ("statistical clustering").



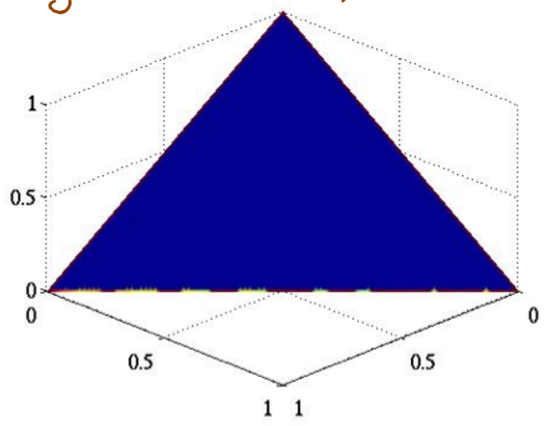
1. Draw a new distribution X from $\boldsymbol{\theta}$
2. With probability $\alpha \div (\alpha + n - 1)$ draw a new X
   With probability $n \div (\alpha + n - 1)$, (re-)sample an X_i from X

The Dirichlet prior α

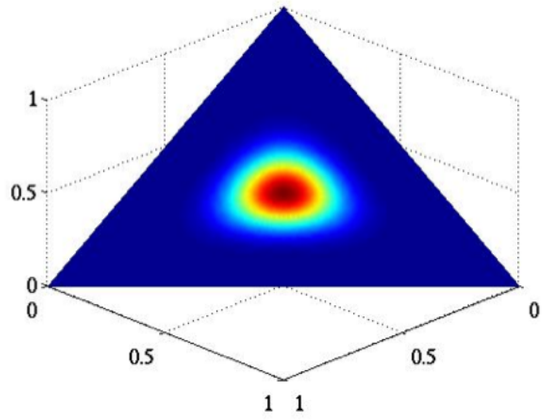
"density plots over the probability simplex in R3" ("simplex" is a fancy way for saying "triangle in arbitrary dimensions")



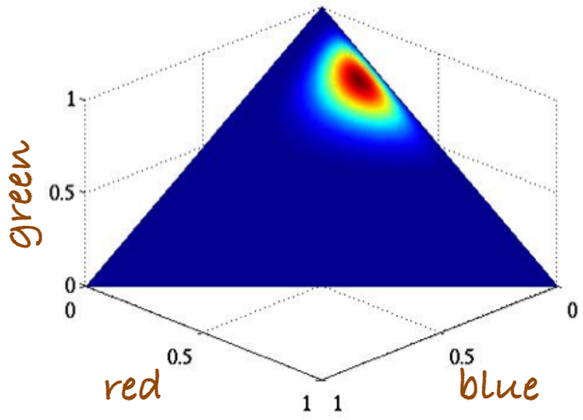
$\alpha = [1, 1, 1]$



$\alpha = [.1, .1, .1]$

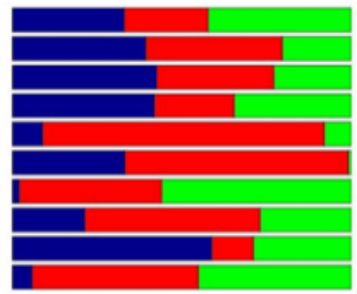


$\alpha = [10, 10, 10]$

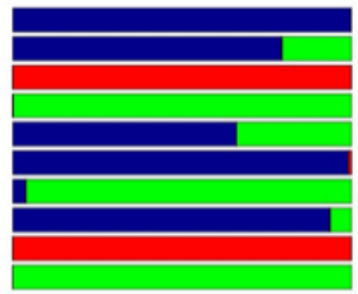


$\alpha = [2, 5, 15]$

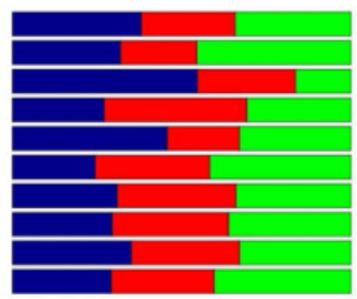
Documents and topic distributions (N=3) (aka. the "Stick breaking process")



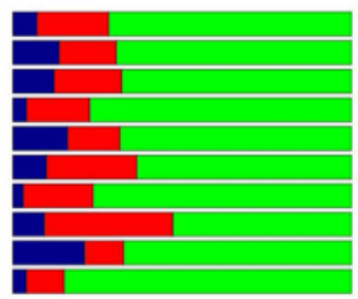
$\alpha = (1, 1, 1)$



$\alpha = (0.1, 0.1, 0.1)$



$\alpha = (10, 10, 10)$



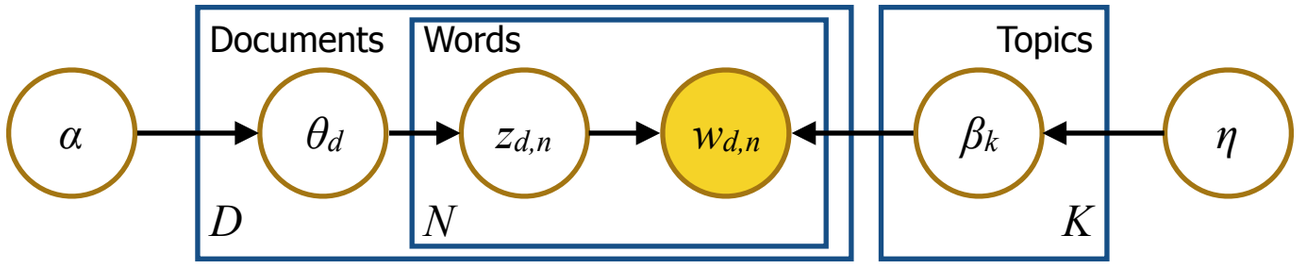
$\alpha = (2, 5, 15)$

→ all equal, =1 ➡ uniform distribution
→ all equal, <1 ➡ marginal distrib. ("choose few")
→ all equal, >1 ➡ symmetric, mono-modal distrib.
→ not equal, >1. ➡ non-symmetric distribution

Frigyik et al. Introduction to the Dirichlet Distribution and Related Processes. 2010

Latent Dirichlet Allocation (LDA 2/3)

A Document-Topic is the assignment of a Document to some Topic.
A Word-Topic is the assignment of a (non-unique!) Word (in a Document) to some Topic



Joint Probability

P(Document-T.)

P(Word | Topics, Word-T.)

$$P(B, \Theta, Z, W) = \left(\prod_k^K P(\beta_k|\eta)\right)\left(\prod_d^D P(\theta_d|\alpha)\prod_n^N P(z_{d,n}|\theta_d)P(w_{d,n}|\beta_{1:K}, z_{d,n})\right)$$

P(Topic)

P(Word-T. | Document-T.)

- α - per-document Dirichlet prior
- θ_d - topic distribution of document d
- $z_{d,n}$ - word-topic assignments
- $w_{d,n}$ - **observed** words
- β_k - word distrib. of topic k
- η - per-topic Dirichlet prior

dampens the topic-specific score of terms assigned to many topics

$$termscore_{k,n} = \hat{\beta}_{k,n}\, log \frac{\hat{\beta}_{k,n}}{\left(\prod_j^K \hat{\beta}_{j,n}\right)^{1/K}}$$

What Topics is a Word assigned to?

Latent Dirichlet Allocation (LDA 3/3)

- LDA sampling/inference in a nutshell
  - Initialization: Choose K, the number of Topics, and randomly assign one out of the K Topics to each of the N Words in each of the D Documents.
    - The **same word** can have different Topics **at different positions** in the Document.
  - Calculate the posterior probability that Word w generated Topic t: P(Topic | Word).
  - Then, for each Topic and for each Word in each Document:

1. Compute P(Word-Topic | Document-Topic): the proportion of [Words assigned to] Topic t in Document d
2. Compute P(Word | Topics, Word-Topic): the probability a Word w is assigned a Topic t (using the general distribution of Topics and the Document-specific distribution of [Word-] Topics)
   - Note that a Word can be assigned a different Topic each time it appears in a Document.
3. Given the prior probabilities of a Document's Topics and that of Topics in general, reassign P(Topic | Word) = P(Word-Topic | Document) * P(Word | Topics, Word-Topic)

  - Repeat until P(Topic | Word) stabilizes (e.g., Collapsed Gibbs sampling)
  - Better: Use collapsed **variational inference** (i.e. combining Variational Bayes) thereby making an deterministic approximation of the posterior.

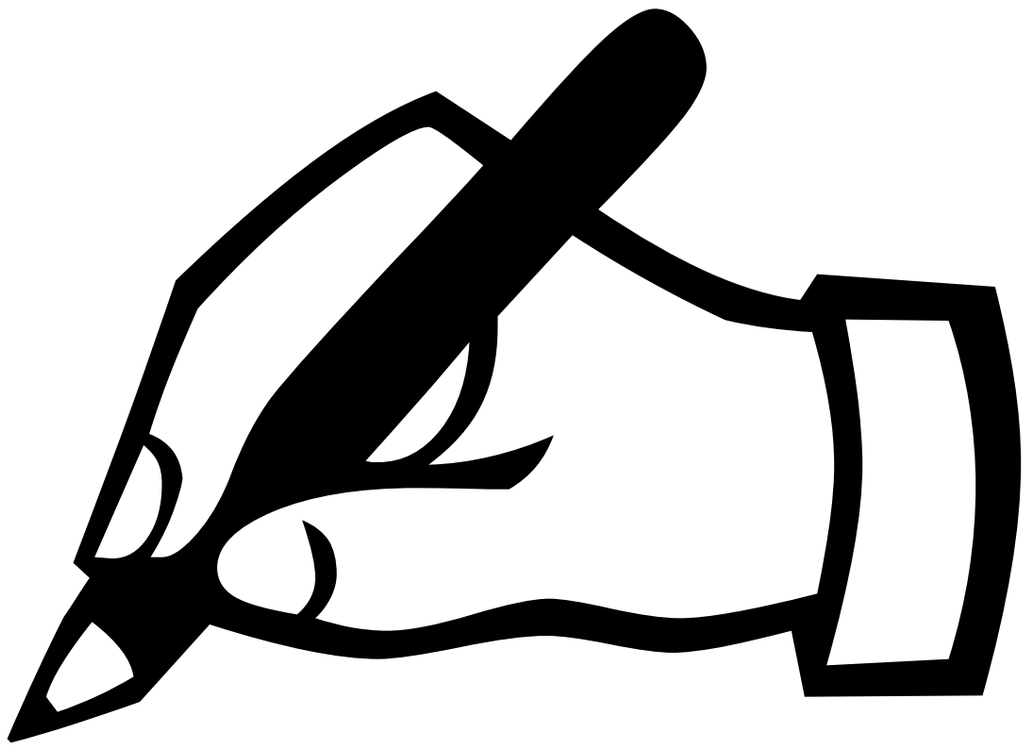
Joint Probability P(Document-T.) P(Word | Topics, Word-T.)

$$P(B,\Theta,Z,W) = \left(\prod_k^K P(\beta_k|\eta)\right)\left(\prod_d^D P(\theta_d|\alpha)\prod_n^N P(z_{d,n}|\theta_d)P(w_{d,n}|\beta_{1:K},z_{d,n})\right)$$

P(Topic) P(Word-T. | Document-T.)

Teh, Newman, Welling (2006). A Collapsed Variational Bayes Inference Algorithm for LDA

Practical 8:
Document clustering



Representation Learning

Madrid Summer School on
Advanced Statistics and Data Mining

Florian Leitner
Selerity, Inc.; Data Catalytics, SL; IE University
leitner@datacatytics.com

Representation learning

- a **transformation of raw data** to a representation that can be effectively exploited in machine learning tasks
- **obviates feature engineering** (manually developing a representation to use for the classifier)
- many feature learning techniques do not require labeled data (i.e., are fully **unsupervised**)

Distributional term representations

A trivial approach is to use a token's string itself as the representation; Numerically encode that leads us to a sparse, **one-hot** vector:

$$\text{"}tutorial\text{"} := [0\ 0\ 0\ 0\ \ldots 0\ 0\ 0\ 0\ \mathbf{1}\ 0\ 0\ 0\ 0\ \ldots 0\ 0\ 0]$$

Problem: every such vector v is orthogonal to all others, so:

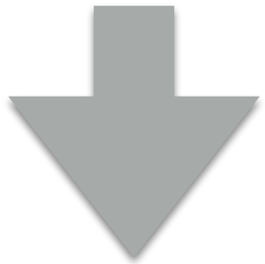
$$\mathbf{v}_1^T \cdot \mathbf{v}_2 = \mathbf{0}$$

In other words, there is no notion of similarity between those vectors.

Therefore, the goal of word representations is to [numerically] **quantify the similarity of related words**.

From one-hot encoding to word embeddings

```
fun   = [1.0, 0.0, …, 0.0, 0.0, …, 0.0]
enjoy = [0.0, 0.0, …, 1.0, 0.0, …, 0.0]
like  = [0.0, 0.0, …, 0.0, 0.0, …, 1.0]
```

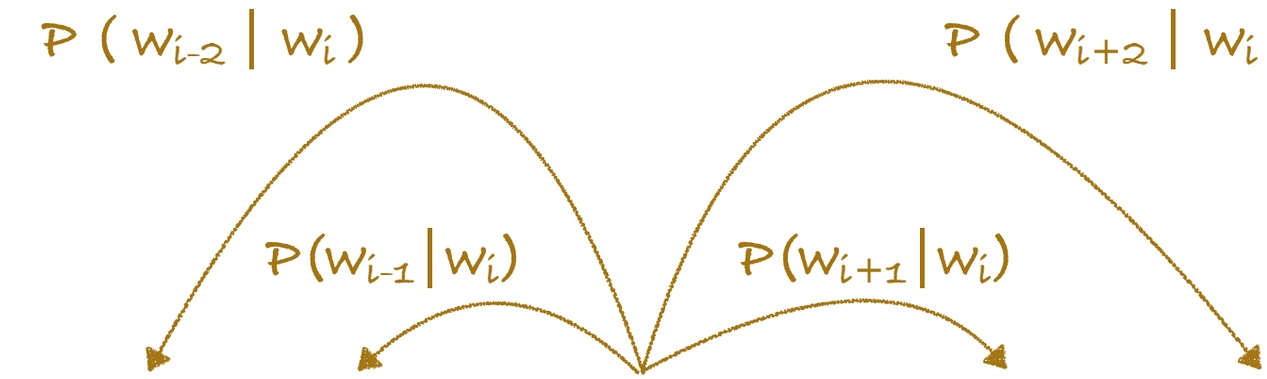


```
fun   = [0.6, 0.0, …, 0.3, 0.0, …, 0.1]
enjoy = [0.4, 0.0, …, 0.5, 0.0, …, 0.1]
like  = [0.2, 0.0, …, 0.2, 0.0, …, 0.6]
```

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

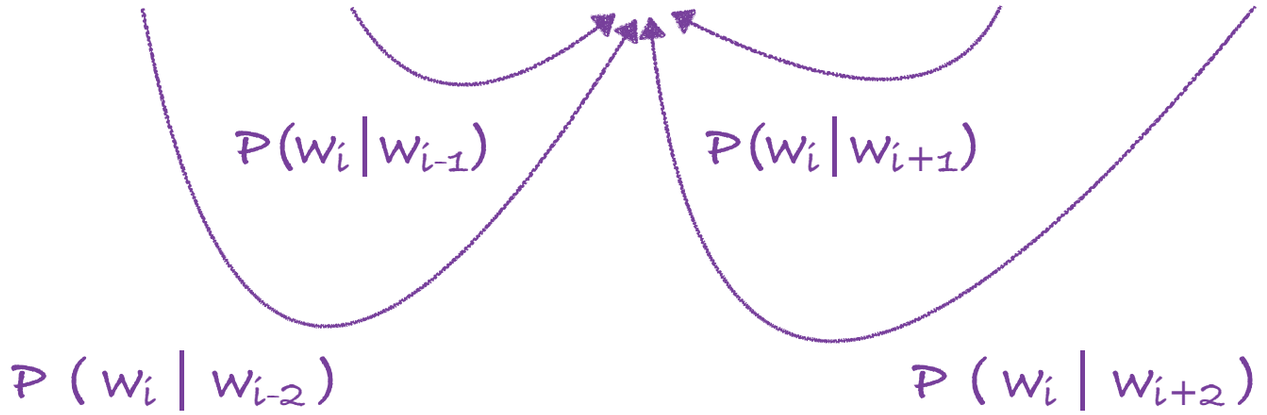
- J. R. Firth, **1957**:11
  (so the idea of word embeddings is definitely not new...)
- That is, the **context** (the surrounding words) of a word is dependent on the word itself; Put it slightly differently: a word "dictates" the possible words you can find in its surrounding.

Predicting the surrounding of words (and vice versa)



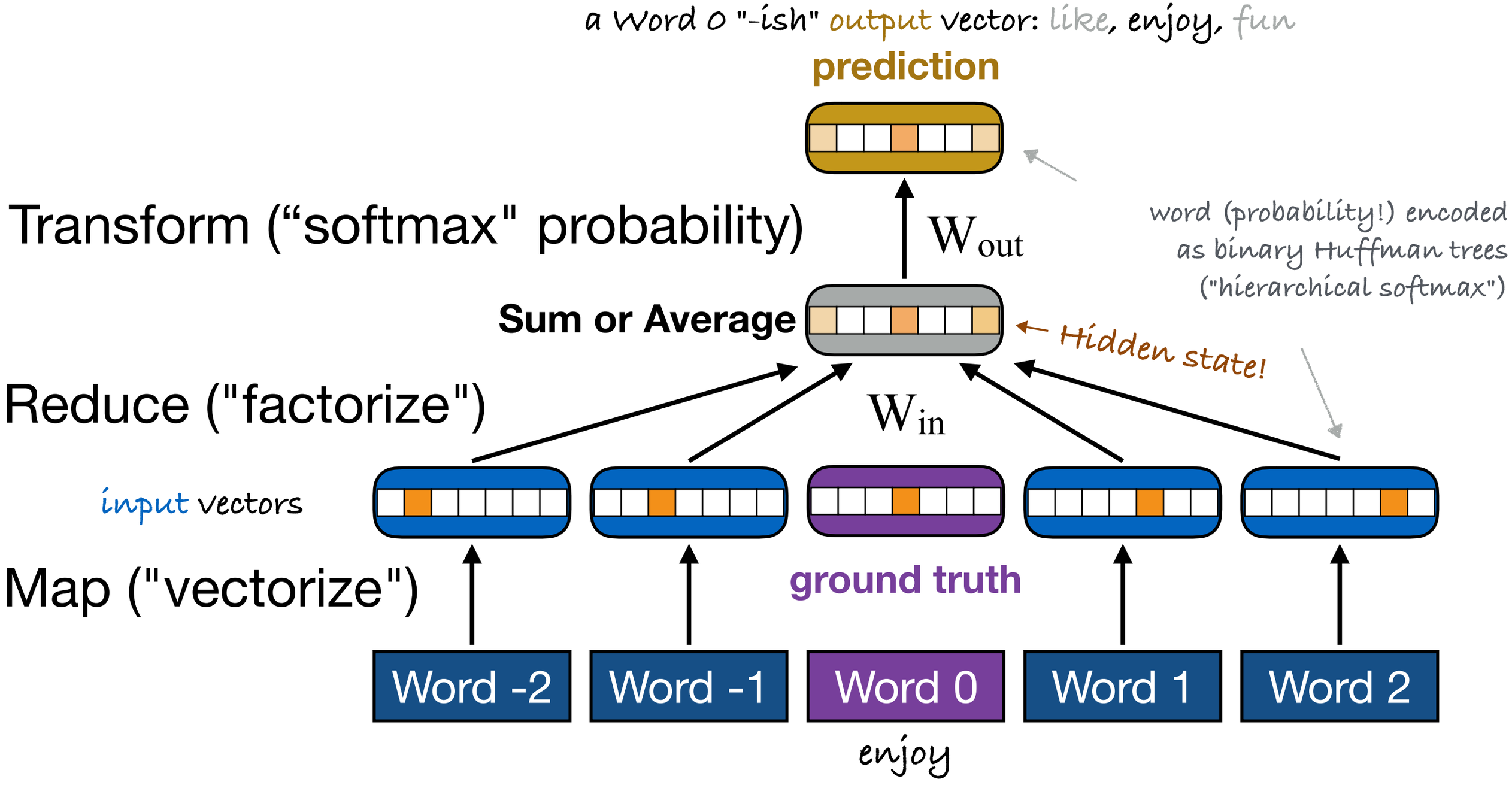
(skip-gram)

... today we learned something new ...



(CBOW)

Word embeddings with neural networks (CBOW model)



But where are the final word embeddings (vectors)?

$$\text{Embeddings} := \text{ll}\ W_{in}\ \text{ll}_2$$

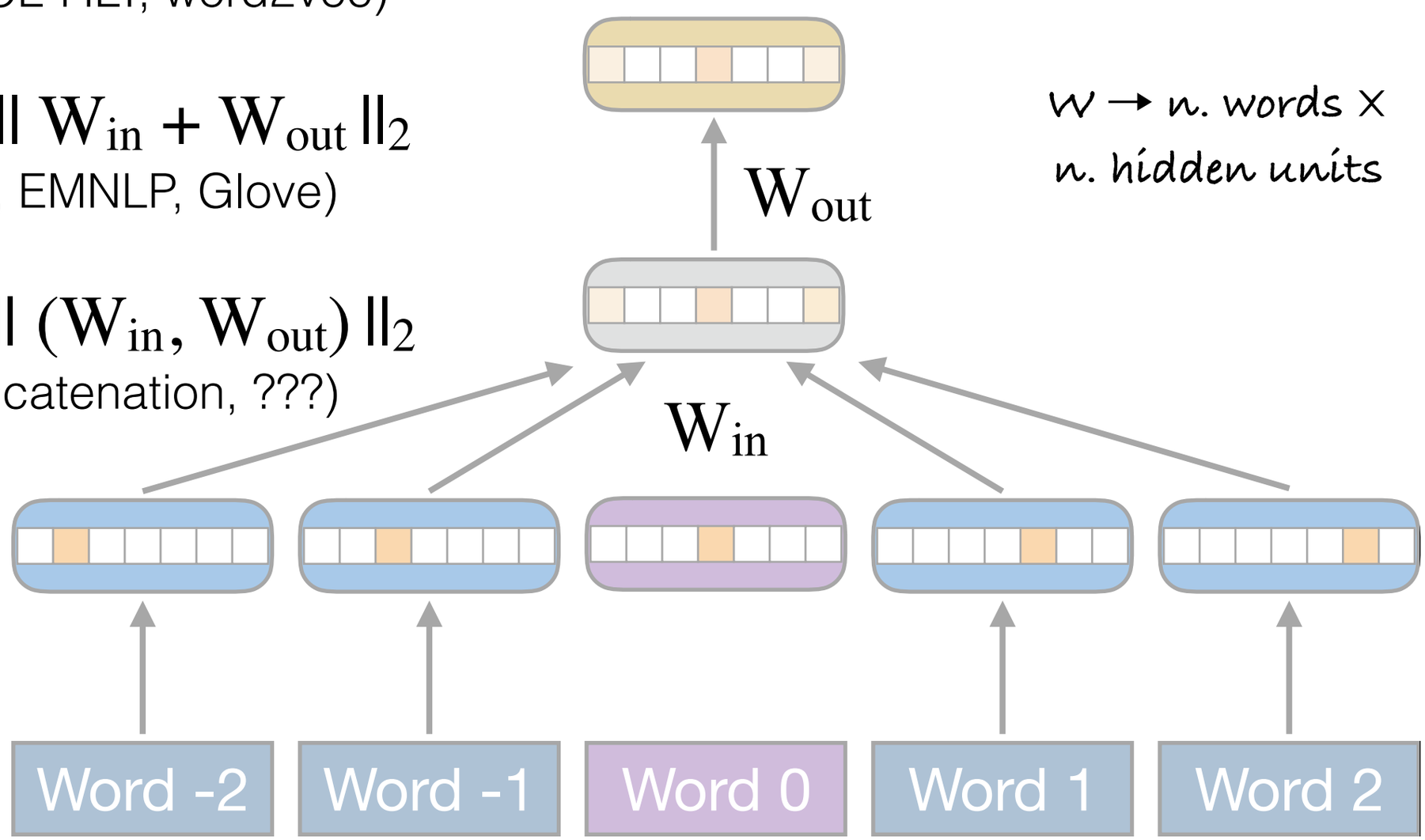
(Mikolov 2013, NAACL-HLT, word2vec)

$$\text{Embeddings} := \text{ll}\ W_{in} + W_{out}\ \text{ll}_2$$

(Pennington 2014, EMNLP, Glove)

$$\text{Embeddings} := \text{ll}\ (W_{in}, W_{out})\ \text{ll}_2$$

(in+out vector concatenation, ???)

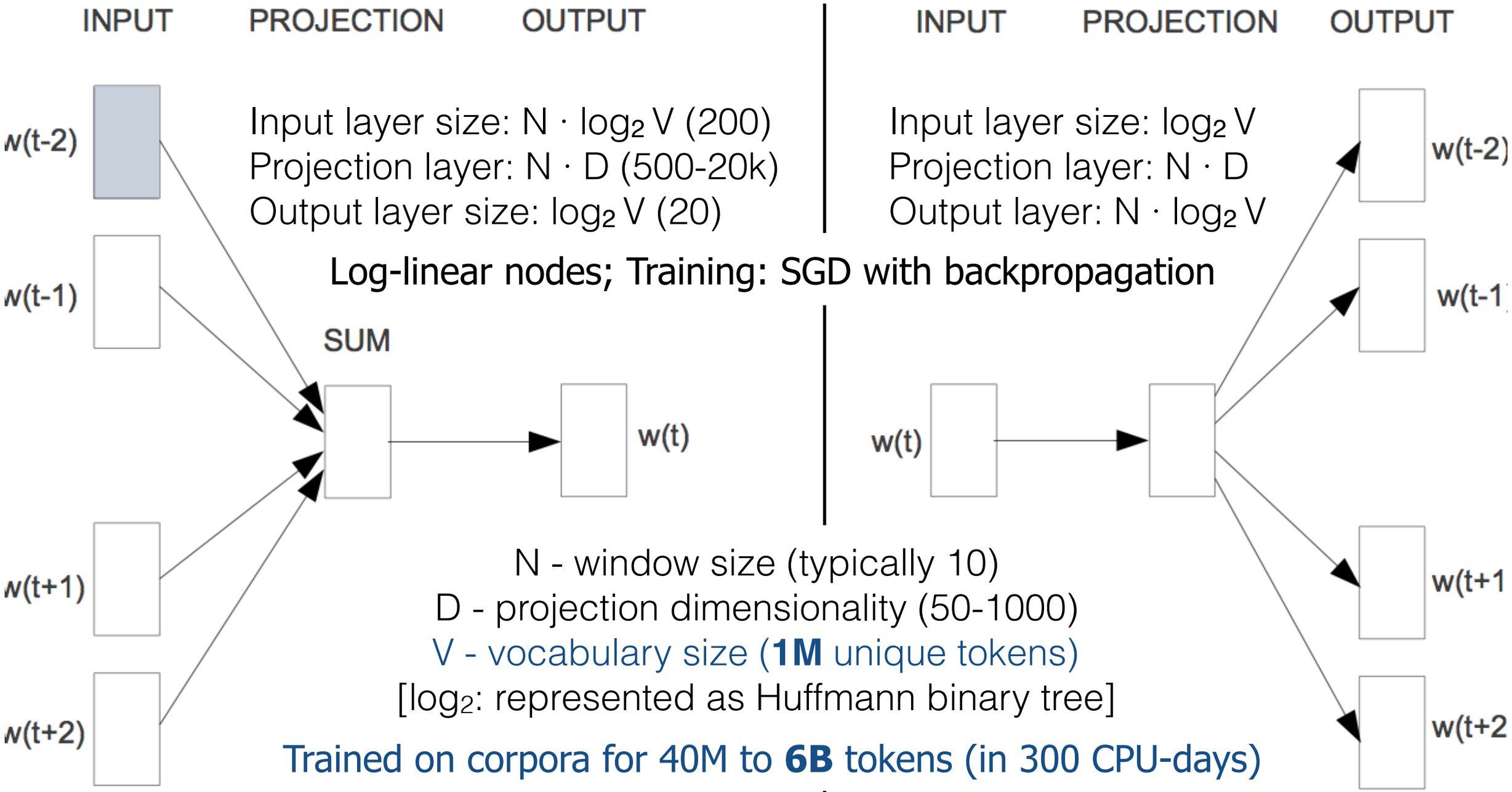


W → n. words X
n. hidden units

NB: softmax CBOW is easier to understand, but skip-gram with negative subsampling (SGNS) performs better

Neural network models of language

word2vec - Thomas Mikolov et al. - Google - 2013



Input layer size: $N \cdot \log_2 V$ (200)
Projection layer: $N \cdot D$ (500-20k)
Output layer size: $\log_2 V$ (20)

Input layer size: $\log_2 V$
Projection layer: $N \cdot D$
Output layer: $N \cdot \log_2 V$

Log-linear nodes; Training: SGD with backpropagation

N - window size (typically 10)
D - projection dimensionality (50-1000)
V - vocabulary size (**1M** unique tokens)
[$\log_2$: represented as Huffmann binary tree]

Trained on corpora for 40M to **6B** tokens (in 300 CPU-days)

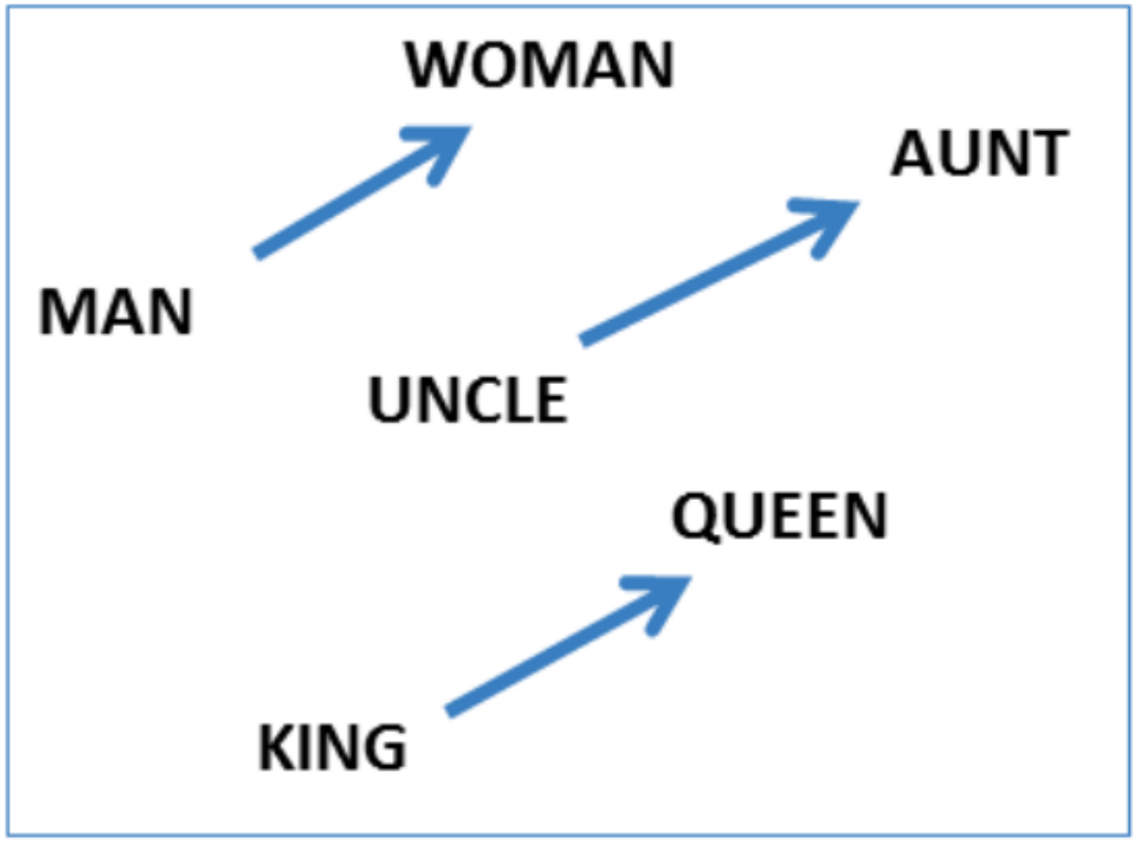
CBOW

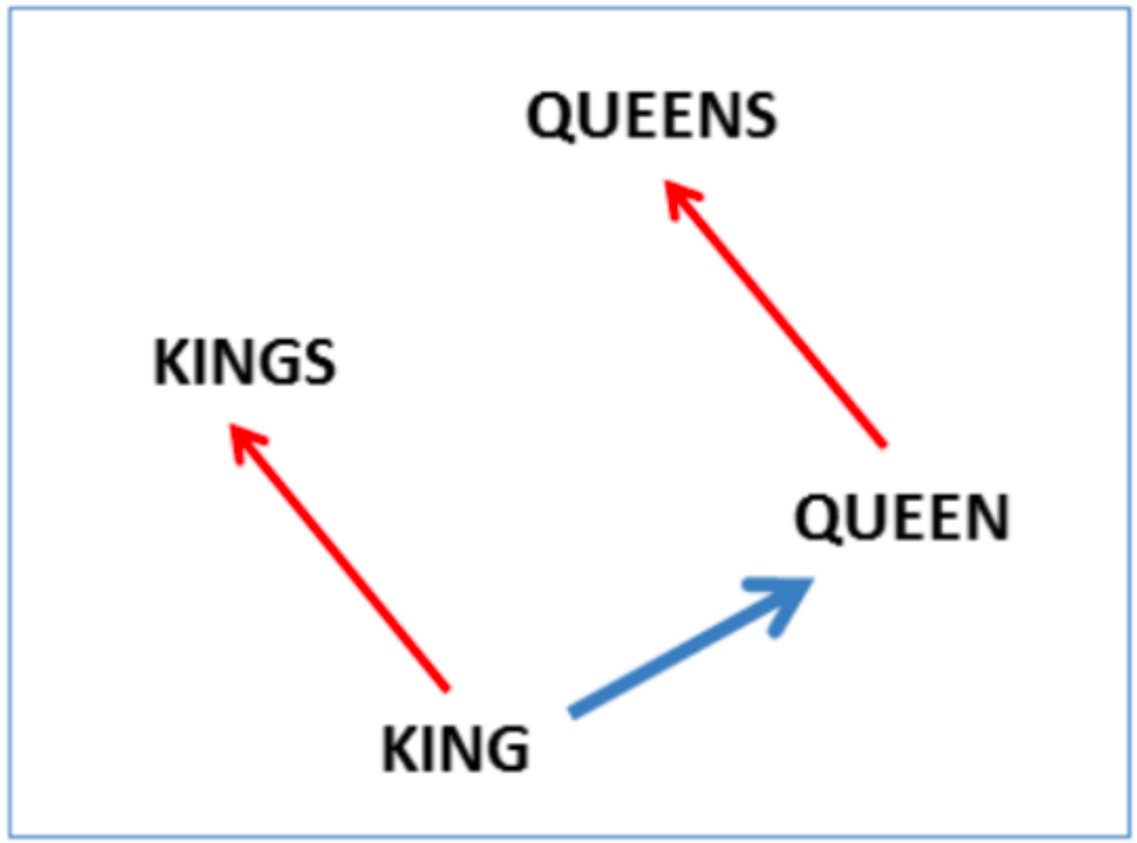
predict a word from its surrounding

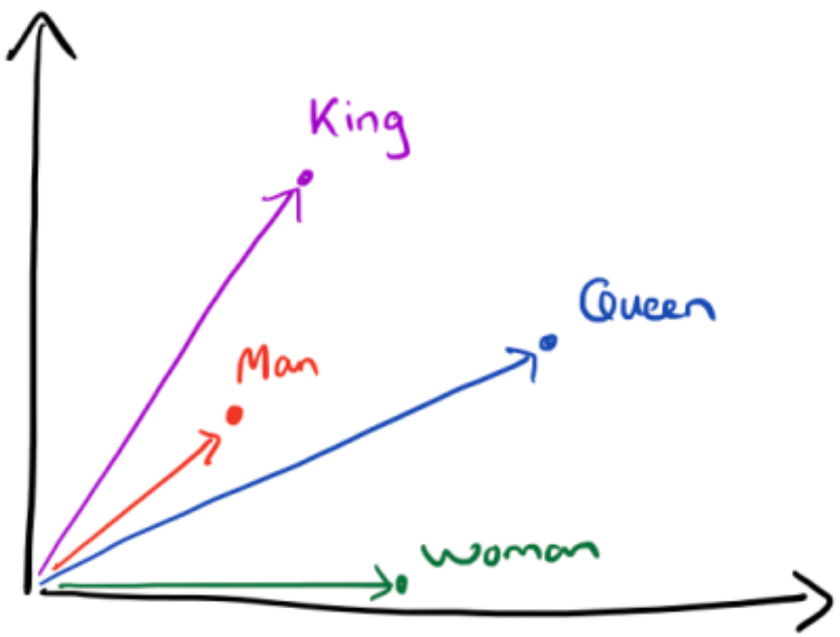
Skip-gram

predict the surrounding of a word

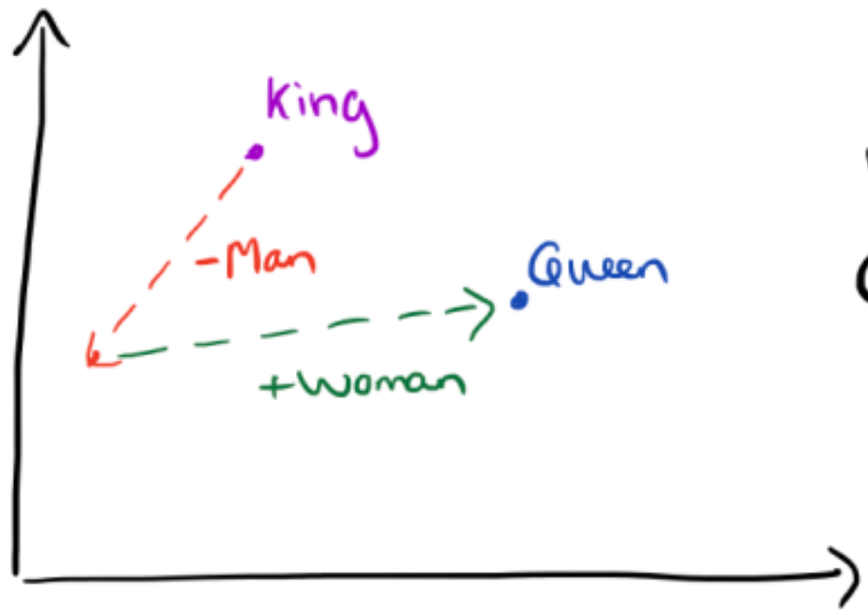
King - Man + Woman = ?







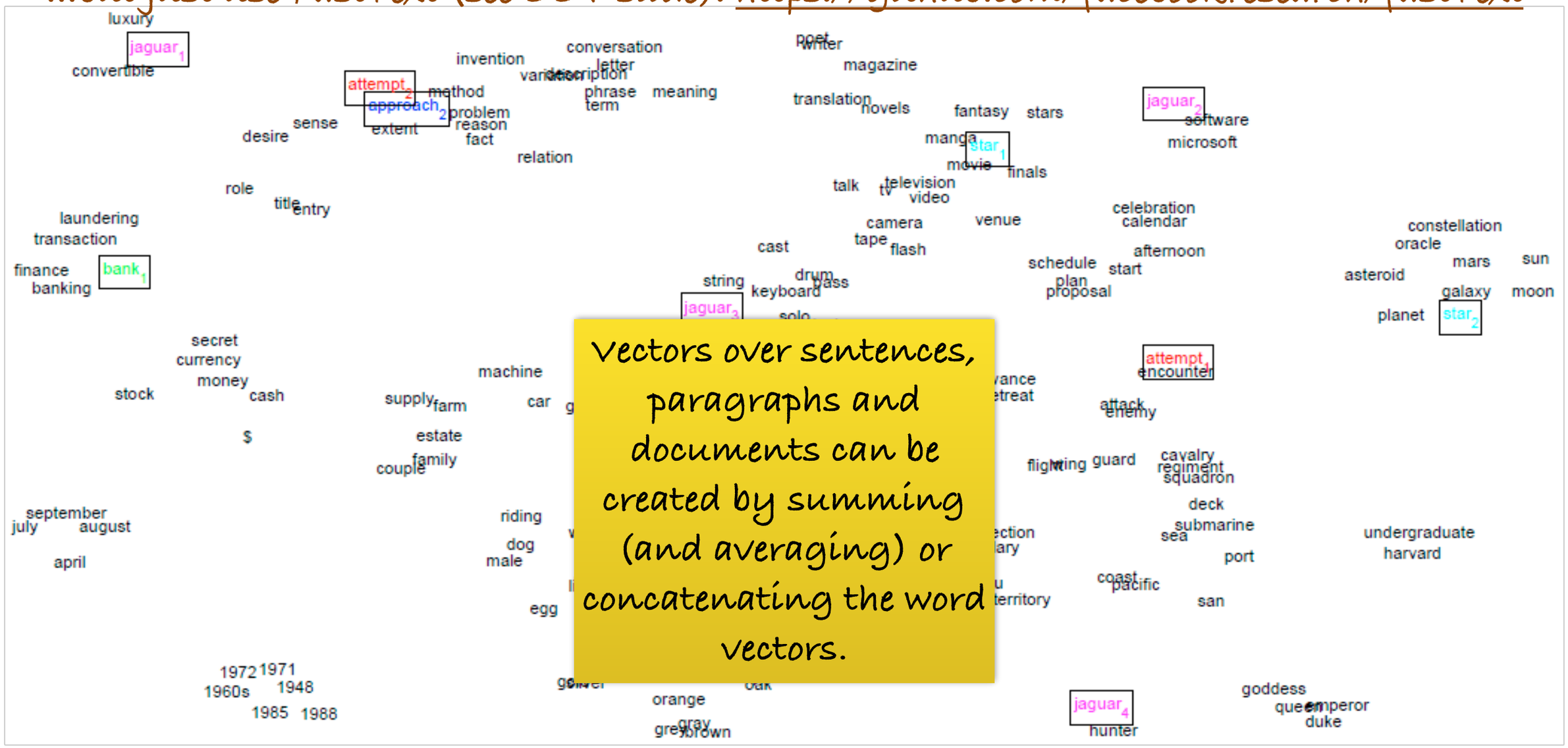
Word Vectors



Vector Composition

word2vec: co-occurrence probs.
GloVe: ratio of co-occ. probabilities

…but just use FastText (see OOV slide): https://github.com/facebookresearch/fastText



t-distributed Stochastic Neighbor Embedding: t-SNE

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. ICLR Workshop.

Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. In Proceedings of the Empiricial Methods in Natural Language Processing (pp. 1532–1543).

Out-of-vocabulary (OOV) words

Problem: no embedding for words not seen during training

Solution: instead learn the embeddings of a word's n-grams

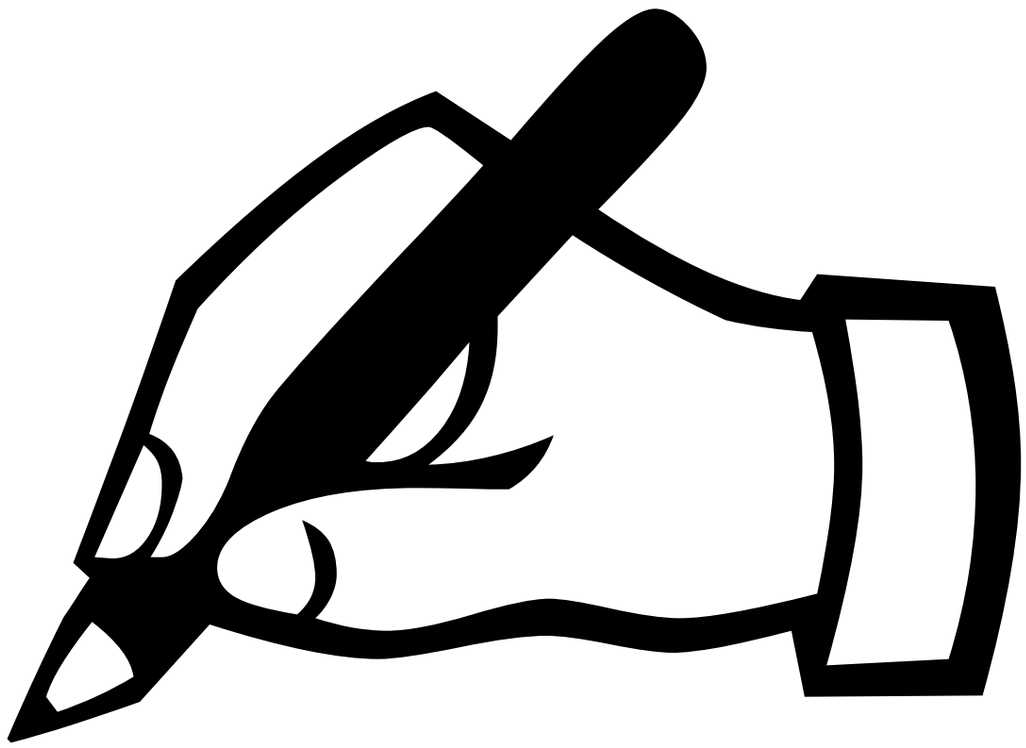
split each word into its **character** n-grams (typically, $\mathrm{n} = [3, 6]$; and just use the word "as is" for tokens with character lengths < 4)

learn to embed the n-grams, with the target embedding being the average over the predicted n-gram embeddings

fastText: Joulin et al., 2016, arXiv (Facebook)

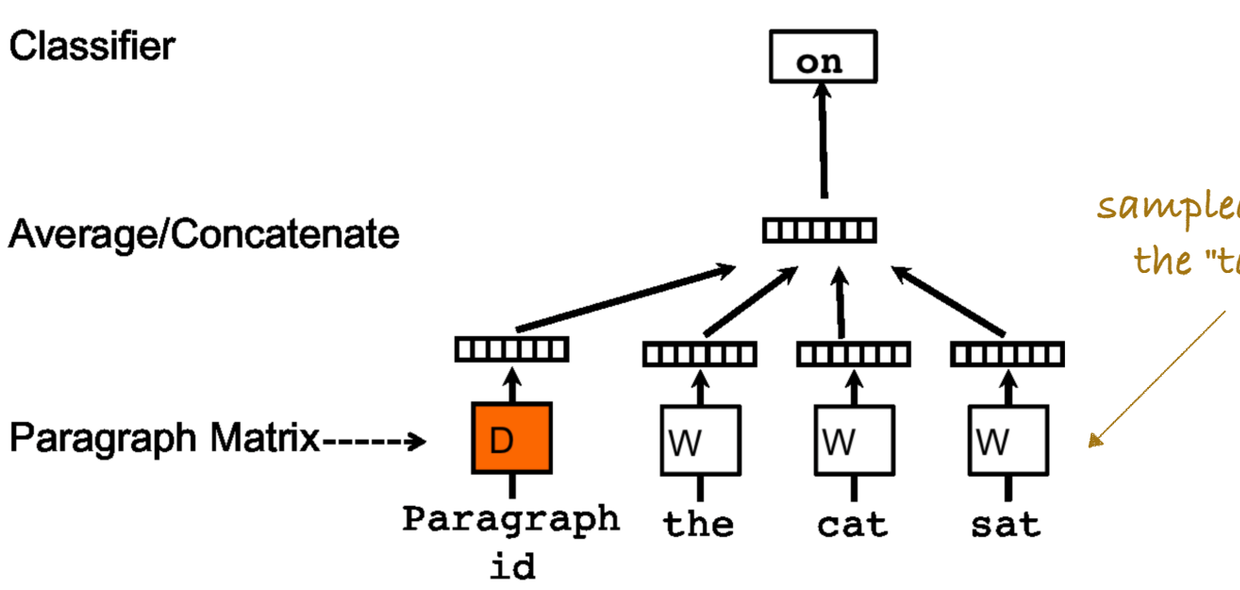
Dirty cheap hack: bucket all words into a fixed-size hash-table (smaller than the actual vocabulary) and allow for collisions (also known as the "**hashing trick**")

Practical 11:
Word embeddings

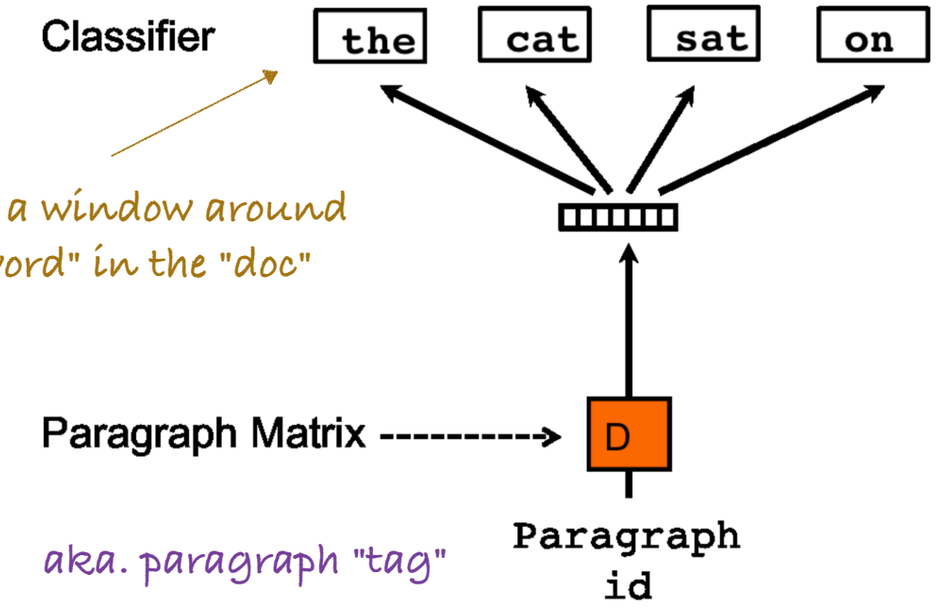


Text embeddings with paragraph vectors (doc2vec)

a "doc" here is some piece of text: a sentence, a tweet, a paragraph, or even a whole document



PV-Distributed Memory (DM)
Predict the target word from the paragraph ID vector and the words (running a window over the paragraph, centered at the target word).



Distributed BOW-PV
Predict the paragraph's words (for any window over the doc) given the paragraph ID vector.

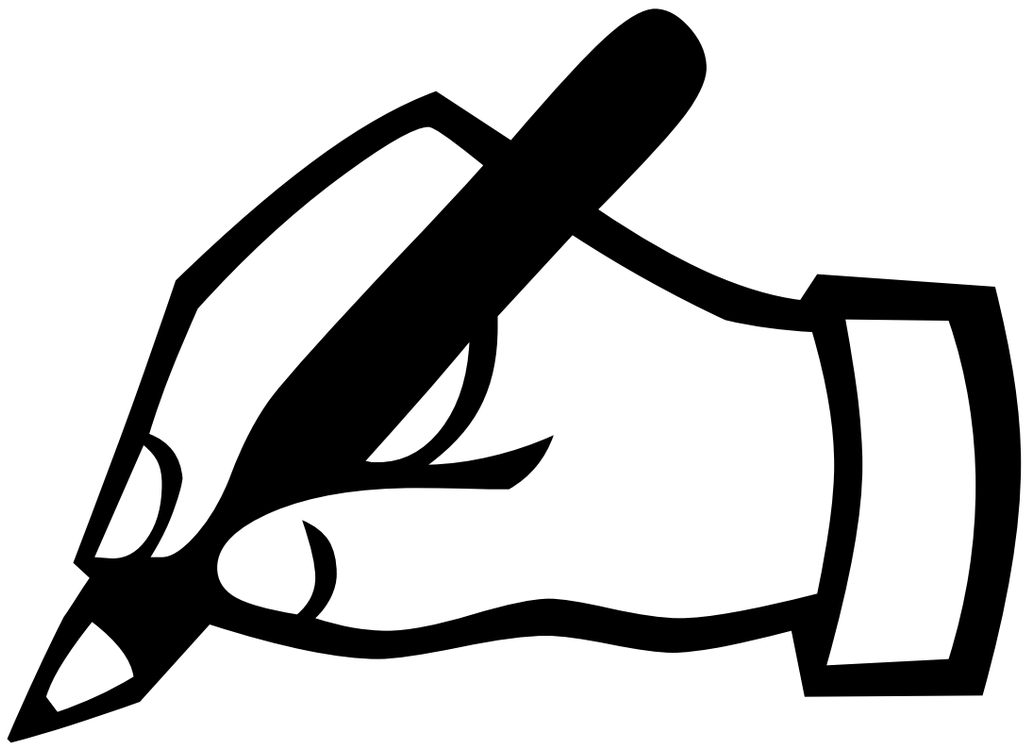
Le and Mikolov's recommendation: train both models using concatenation and combine them.

Le, Q., and Mikolov, T. (2014). Distributed Representations of Sentences and Documents. 2014

Paragraph vectors (doc2vec)

- Base idea is the same as word embeddings
  - c.f. CBOW/SGNS models
- But the **paragraph vector** D **needs to be inferred** when deploying this model ("in production")
  - i.e., you **predict** the paragraph's embedding
  - c.f. **looking up** the embedding vector for words
- D (the paragraph ID) is a **tag** for each doc
  - used as "memory" mechanism for that paragraph during training
  - typically just a unique integer per paragraph
- Scales to larger collections than LSA, but has lesser quality

Practical 12: Document embeddings



Statistical models of language and polysemy

- Polysemous words have multiple meanings (e.g., “bank”).
  - This is a real problem in scientific texts because polysemy is frequent.
- One idea: Create **context vectors** for each sense of a word (vector), by separating vectors of equal words at some cutoff similarity into two or more buckets.
  - MSSG - Neelakantan et al. - 2015
- Caveat: Performance isn’t much better than for the skip-gram model by Mikolov et al., while training is ~5x slower.
- Hacky, partial solution: use **collocations [and PoS tagging]**
  - Train your embeddings over [PoS-tagged] collocations, not just the tokens!

Word Sense Disambiguation (WSD)

Note the PoS-tag "dependency": otherwise, the two examples would have even more senses!

- Basic Example: **hard** [JJ]
  - physically hard (a hard stone)
  - difficult [task] (a hard task)
  - strong/severe (a hard wind)
  - dispassionate [personality] (a hard bargainer)
- Entity Disambig.: **bank** [NN]
  - finance ("bank account")
  - terrain ("river bank")
  - aeronautics ("went into bank")
  - grouping ("a bank of …")

- **SensEval**
  - http://www.senseval.org/
  - SensEval/SemEval Challenges
  - Provides corpora where every word is tagged with its sense
- **WordNet**
  - http://wordnet.princeton.edu/
  - A labeled graph of word senses
- **Applications**
  - Named entity recognition
  - Machine translation
  - Language understanding

Applications of term and document representations

- Classification, e.g., opinion mining (Maas et al., 2011)
- Paraphrase detection (Socher et al., 2011)
- Chunking (Turian et al., 2010; Dhillon and Ungar, 2011)
- Named entity recognition (Neelakantan and Collins, 2014; Passos et al., 2014; Turian et al., 2010)
- Dependency parsing (Bansal et al., 2014)
- … (really, they are simply everywhere)

Statistical Language Modeling

Madrid Summer School on
Advanced Statistics and Data Mining

Florian Leitner
Data Catalytics, S.L.
leitner@datacatytics.com

Motivation for statistical models of language

Two sentences:

"Colorless green ideas sleep furiously." (from Noam Chomsky's 1955 thesis)

"Furiously ideas green sleep colorless."

Which one is (grammatically) correct?

An unfinished sentence:

"Adam went jogging with his …"

What is a correct phrase to complete this sentence?

Incentive and applications

- Manual **rule-based** language processing would become **cumbersome**.
- Word frequencies (**probabilities**) are **easy to measure**, because large amounts of texts are available to us.
- Modeling language based on probabilities enables many existing **applications**:
  - Spelling correction
  - Machine translation
  - Voice recognition
  - Predictive keyboards
  - Langauge generation
  - Linguistic parsing & chunking
    - (analyses of the part-of-speech and grammatical structure of sentences; word **semantics**)

Probabilistic language modeling

- Manning & Schütze. Statistical Natural Language Processing. 1999
- A sentence W is defined as a **sequence** of words $w_1, \ldots, w_n$
- Probability of **next word** w_n in a sentence is: $P(w_n | w_1, \ldots, w_{n-1})$
  - a **conditional probability**
- The probability of the **whole sentence** is: $P(W) = P(w_1, \ldots, w_n)$
  - the **chain rule of conditional probability**
- These counts & probabilities form the **language model** [for a given document collection (=**corpus**)].
  - the model variables are **discrete** (counts)
  - only needs to deal with **probability mass** (not density)

Modeling the stochastic process of "generating words" using the chain rule

"This is a long sentence with many words…" ➡

$$P(W) = P(\text{this}) \times$$

$$P(\text{is} \mid \text{this}) \times$$

$$P(\text{a} \mid \text{this, is}) \times$$

$$P(\text{long} \mid \text{this, is, a}) \times$$

$$P(\text{sentence} \mid \text{this, is, a, long}) \times$$

$$P(\text{with} \mid \text{this, is, a, long, sentence}) \times$$

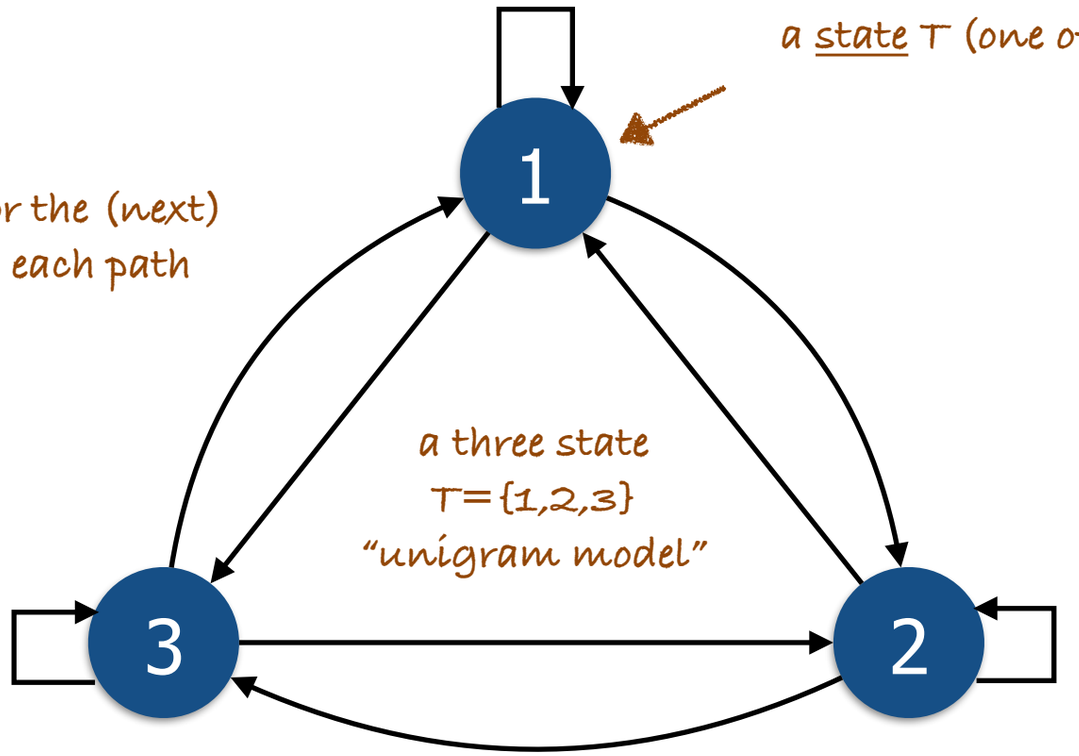
$$P(\text{many} \mid \text{this, is, a, long, sentence, with}) \times$$

….

n-grams for n > 5:
insufficient (**sparse**) data
(and expensive to calculate)

The Markov property

A Markov process is a stochastic process who's future (next) state only depends on the current state T, but not its past.



1222311223123123132…

A. A. Markov (1886).

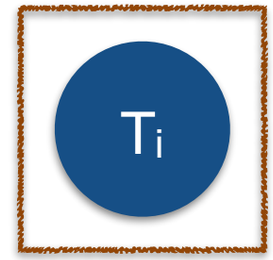
Modeling the stochastic process of "generating words" assuming it is Markovian

stochastic process: "Unigram Model"

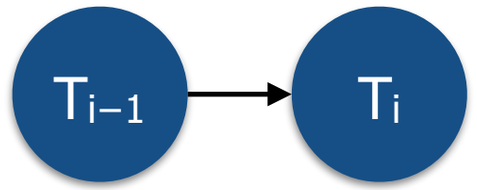
$$\prod P(w_i|w_1^{i-1}) \approx \prod P(w_i|w_{i-k}^{i-1})$$

All words before w_i

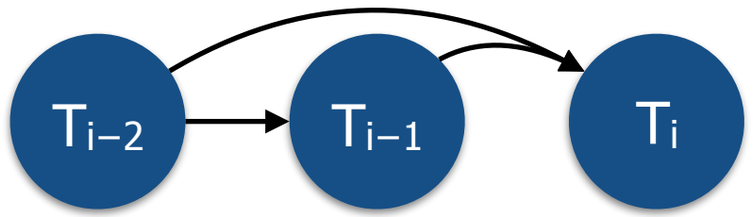
Only the k words before w_i



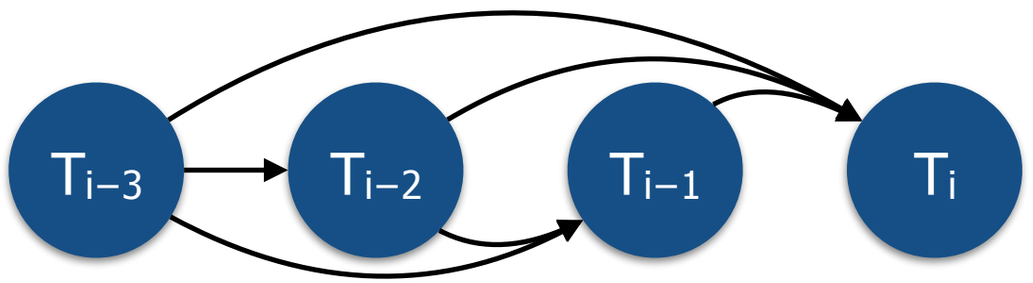
- 1st Order Markov Chains
  - Bigram Model, k=1, P(be | this)



- 2nd Order Markov Chains
  - Trigram Model, k=2, P(long | this, be)



- 3rd Order Markov Chains
  - Quadrigram Model, k=3, P(sentence | this, be, long)



- …

dependencies could span over a dozen tokens, but these sizes are generally sufficient to work by

Measuring n-gram (transition) probabilities

- Unigrams:

$$P(w_i) = \frac{count(w_i)}{N}$$

N = total word count

- Bigrams:

$$P(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

- **N-grams** (n=k+1):

$$P(w_i|w_{i-k}^{i-1}) = \frac{count(w_{i-k}^{i})}{count(w_{i-k}^{i-1})}$$

k = n-gram size - 1

▸ **Language Model:**

$$P(W) = \prod P(w_i|w_{i-k}^{i-1}) =$$

$$= \prod P(w_i|w_{i-k}, ...w_{i-1})$$

Programming tip:
transform probabilities
to logs to avoid underflows
and work with addition

Unseen n-grams and the zero probability issue

- Even for unigrams, unseen words will occur sooner or later
- The longer the n-grams, the sooner unseen cases will occur
- "Colorless green ideas sleep furiously." (Chomsky, 1955)
- As unseen tokens have **zero probability**, the probability of the whole sentence $P(\boldsymbol{W}) = 0$ would become zero, too
  - Intuition/idea: Divert a bit of the overall probability mass of each [seen] token to all possible unseen tokens
- Terminology: model **smoothing** (Chen & Goodman, 1998)
- Alternate/modern solution: **Character-based** language models (e.g., CharCNNs) and character n-gram word embeddings (e.g., FastText)

Calculating probabilities for the initial tokens

- How to calculate the first/last [few] tokens in n-gram models?
  - "This is a sentence." ➡ P(this | ???)
  - $P(w_n|w_{n-k}, \ldots, w_{n-1})$
- Fall back to **lower-order Markov models**
  - $P(w_1|w_{n-k},\ldots,w_{n-1}) = P(w_1);\ P(w_2|w_{n-k},\ldots,w_{n-1}) = P(w_2|w_1);\ \ldots$
- Fill positions prior to $n = 1$ with a **generic "start token"**.
  - left and/or right **padding**
  - conventionally, a token like "*****", or "**▪**" is used, or the pair "**<s>**" and "**</s>**"
    (but anything will do, as long as it cannot collide with a possible, real token)

NB: it is important to maintain sentence terminal tokens (., ?, !) to generate robust probability distributions; do not drop them!

Language model evaluation

- How good is the model you just trained?
- Did your update to the model do any good…?
- Is your model better than someone else's?
  - NB: You should compare on the same test set!

Extrinsic evaluation: Error rate

- Extrinsic evaluation: minimizing the error rate
  - evaluates a model's **error frequency**
  - **estimates** the model's per-**word error rate** by comparing the generated sequences to all true sequences (which cannot be established) using a **manual classification** of errors (therefore, "extrinsic")
  - time consuming, but can evaluate non-probabilistic approaches, too

a "perfect prediction" would evaluate to zero

Intrinsic evaluation: Cross-entropy

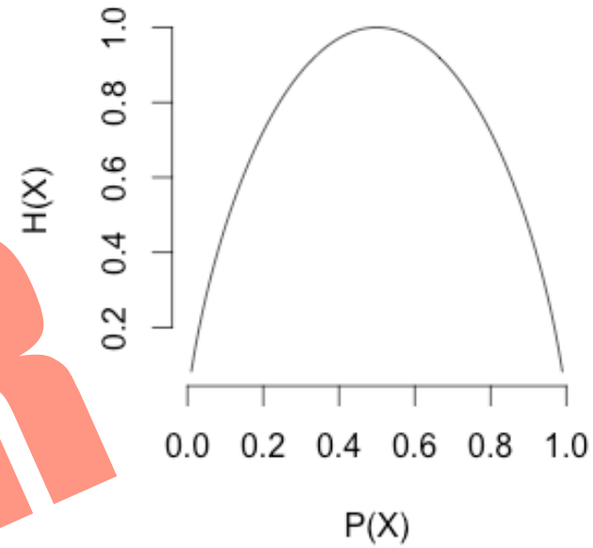
perplexity: roughly, "confusion"

- Intrinsic evaluation: minimizing **perplexity** (Rubinstein, 1997)
  - Compares a model's **probability distribution** to a "perfect" model
  - **Estimates** a distance based on **cross-entropy** between the generated word distribution and the true distribution (which cannot be observed) using the empirical distribution as a proxy
  - Efficient, but can only evaluate probabilistic language models and is only an approximation of the model quality
- **Cross-entropy**: Shannon's entropy of a given distribution plus the (inequal!) Kullback-Leibler divergence to another.

again, a "perfect prediction" would evaluate to zero

Shannon's entropy

- Answers the questions:
  - "How much information (bits) will I gain when I see w_n?"
  - "How predictable is w_n from its past?"



Bernoulli process: $H(X) = -P(X)log_2[P(X)] - (1 - P(X))log_2[1 - P(X)]$

- Each outcome provides $-log_2 P$ bits of information ("surprise").
  - Claude E. Shannon, 1948
  - The more probable an event (outcome), the lower its entropy.
  - A certain event (p=1 or 0) has zero entropy ("no surprise").

- Therefore, the entropy H in our model P for a sentence W is:
  - $H = 1/N \times -\sum P(w_n|w_1, \ldots, w_{n-1}) \log_2 P(w_n|w_1, \ldots, w_{n-1})$

From cross-entropy to model perplexity

- **Cross-entropy** then compares the model probability distribution $\boldsymbol{P}$ to the true distribution $\boldsymbol{P_T}$:

  our "surprise" for the observed sentence given the true model

  - $H(P_T, P, \boldsymbol{W}) = 1/N \times -\sum P_T(\boldsymbol{w_n}|\boldsymbol{w_{n-k}}, \ldots, \boldsymbol{w_{n-1}}) \log_2 P(\boldsymbol{w_n}|\boldsymbol{w_{n-k}}, \ldots, \boldsymbol{w_{n-1}})$

- CE can be simplified if the "true" distribution is a "stationary, ergodic process": (an "unchanging" language, i.e., a naïve assumption)

  - $H(P, \boldsymbol{W}) = 1/N \times -\sum \log_2 P(\boldsymbol{w_n}|\boldsymbol{w_{n-k}}, \ldots, \boldsymbol{w_{n-1}})$

- Then, the relationship between **perplexity** PPX and cross-entropy is defined as the following monotonic transformation:

  - $PPX(\boldsymbol{W}) = 2^{H(P, W)} = P(\boldsymbol{W})^{1/N}$

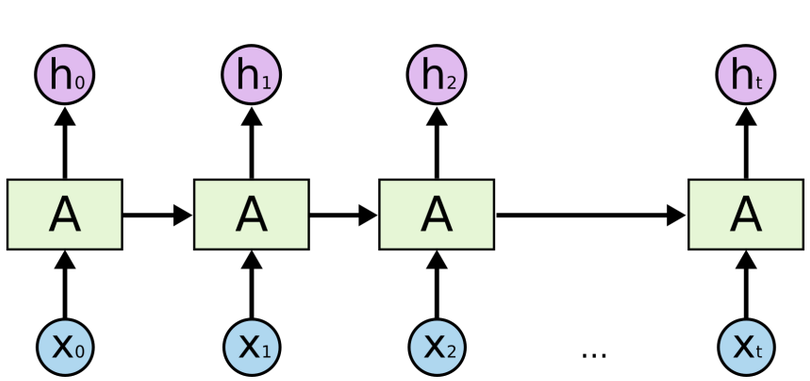
  - where $\boldsymbol{W}$ is the sentence, $P(\boldsymbol{W})$ is the Markov model, and N is the number of tokens in it

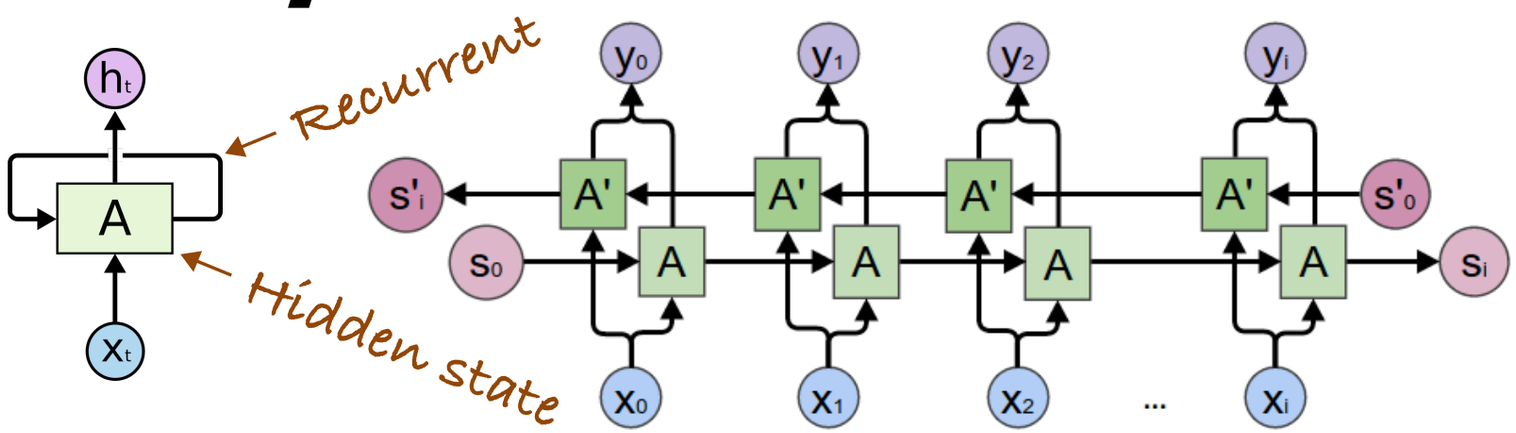
Interpreting perplexity

- The lower the perplexity, the better the model ("less surprise")
- Perplexity is an indicator of the number of equiprobable choices at each step
  - $PPX = 26$ for a model generating an infinite random sequence of Latin letters
  - An unigram Markov model having equal transition probabilities to each letter
  - Perplexity produces "big numbers" rather than cross-entropy's "small bits"
  - Typical (bigram) perplexities in **English texts** range from 50 to almost 1000 corresponding to a cross-entropy from about 5.6 to 10 bits/word.
  - Chen & Goodman, 1998
  - Manning & Schütze, 1999

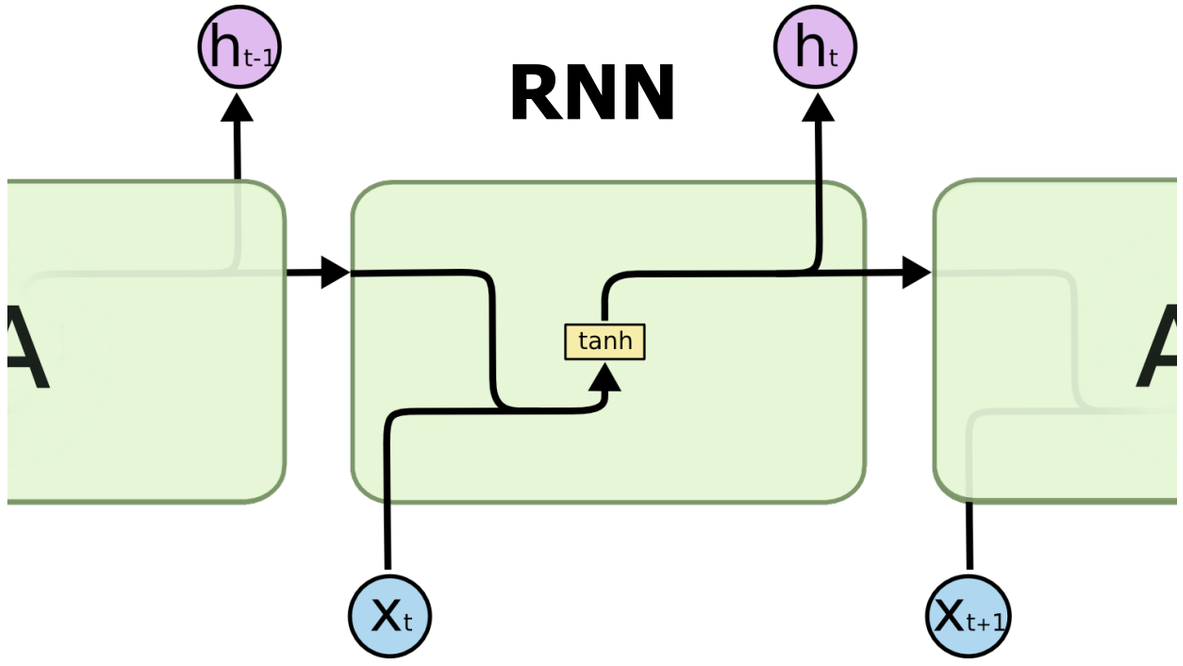
* usually, also with (Continuous Cache) Pointers: See the neural text summarization model.

NLP language models often use RNNs/LSTMs*





Use last h (or y, top right) as sentence/paragraph vector, or all as word vectors.



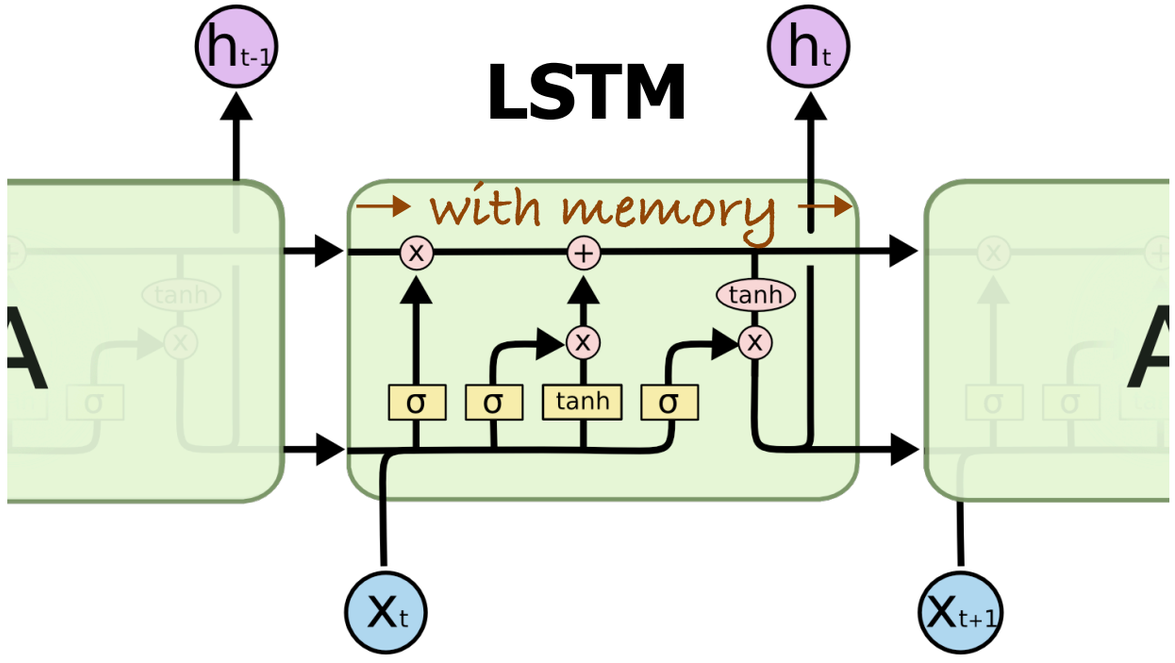


Image sources: C. Olah, http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Merty, Keskar & Socher (2017). Regularizing and Optimizing LSTM Language Models

Peters et al. (2018). Deep contextualized word representations (ELMo)

c.f.: ULMFiT: SOTA model for text classification

Attention-based Transformer models

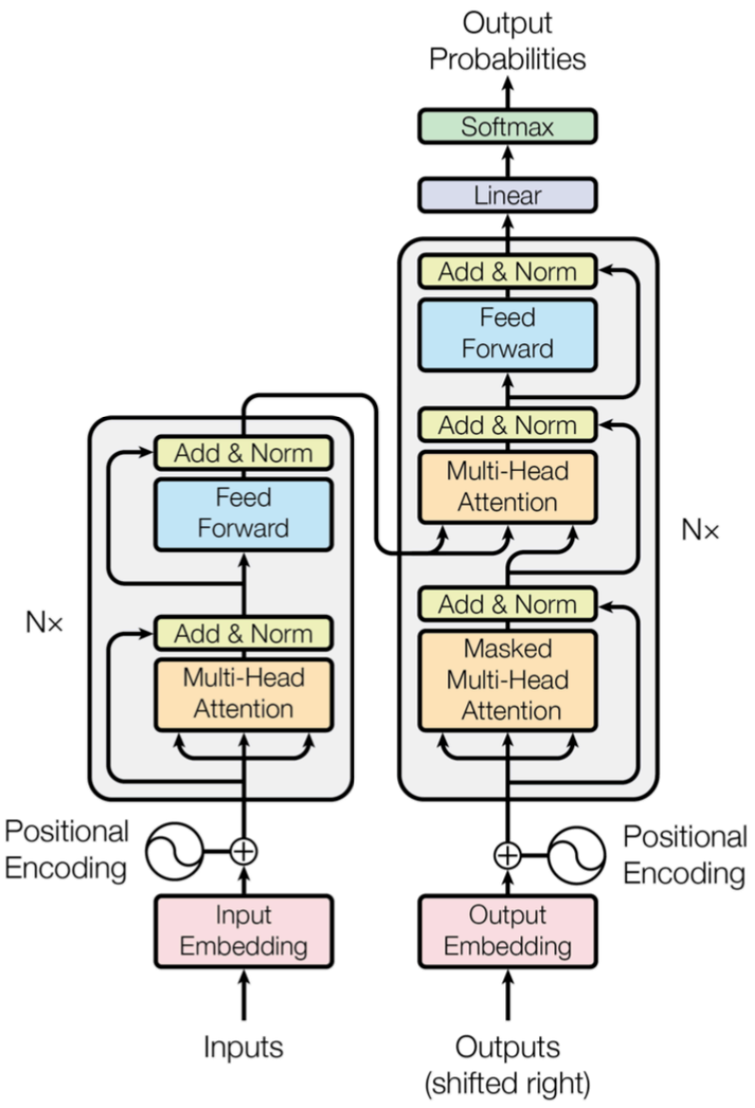
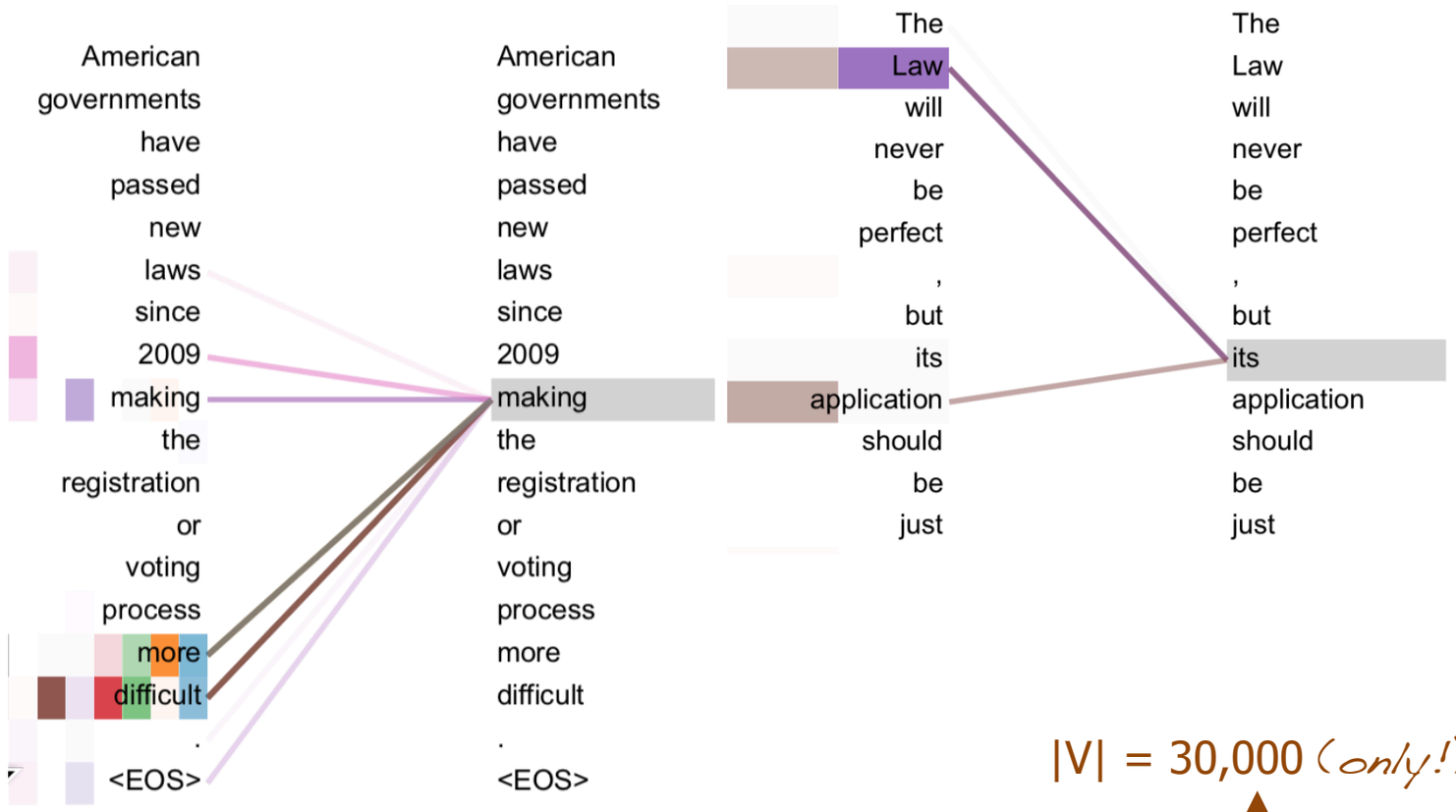
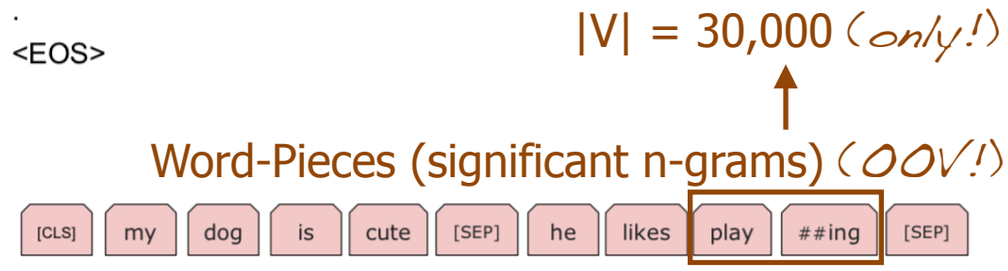


Figure 1: The Transformer - model architecture.





BERT:

Base: 120 M Parameters
N = 12 layers
Large: 340 M Parameters
N = 24 layers

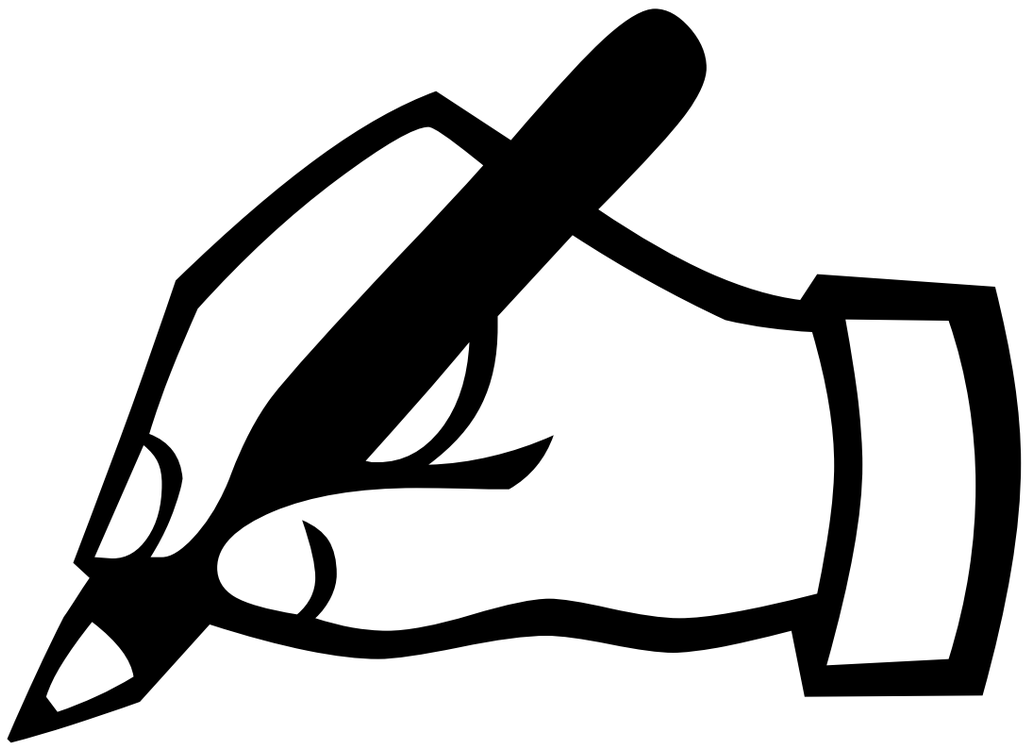
Pre-training:
(a) MASK one token in the input sequence
(b) predict if sentence B follows A

External Jupyter Notebook link:
http://nlp.seas.harvard.edu/2018/04/03/attention.html

Vaswani et al. (2017). Attention is all you need

Devlin et al. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Practical 17:
SOTA neural document classification



Information Extraction

Madrid Summer School on
Advanced Statistics and Data Mining

Florian Leitner
Selerity, Inc.; Data Catalytics, SL; IE University
leitner@datacatytics.com

Text summarization

Russian defense minister Ivanov called on Sunday for the creation of a global front for combating terrorism.

- Extractive summarization
  - Select the most informative sentences.
  - Order sentences (or leave in order).

Ivanov called for a global front combating terrorism.

- Abstractive summarization
  - Generate new text given the input document.
  - Very unique but still rather experimental.

Russia calls for a joint effort against terrorism.

Extractive summarization with TextRank

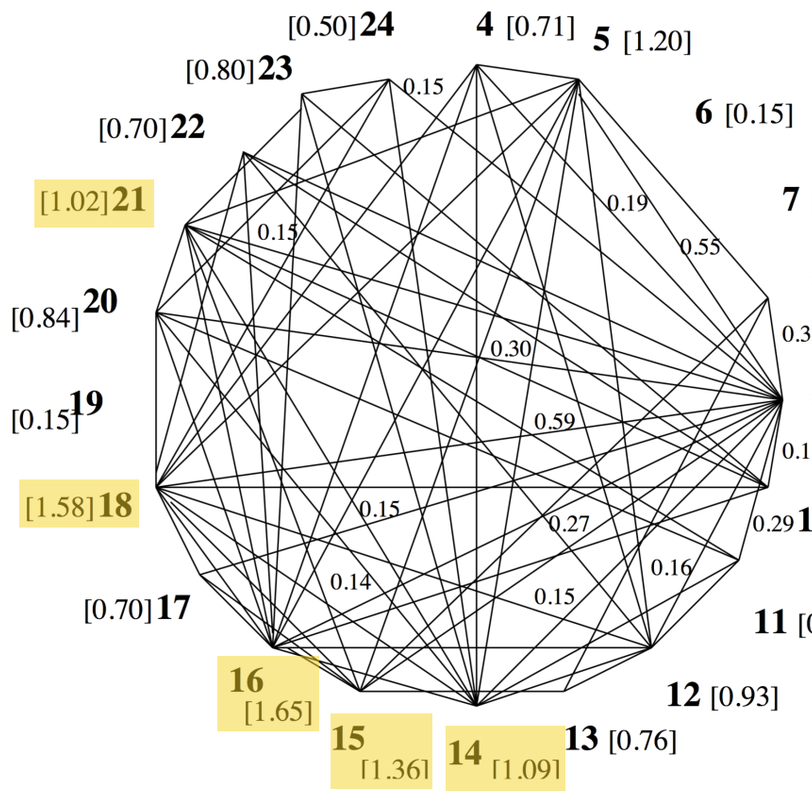
tokens, n-grams, terms, or whole sentences

1. Collect all text shingles (→ graph vertices) from the input document[s].

2. Quantify relation strength (→ edges) between those shingles from their context (**co-occurrence**) or content (**TF-IDF**).

3. Iterate a graph ranking algorithm (**PageRank**) to convergence.

4. Sort the vertices on their final score to identify the most informative shingles.



Mihalcea, R., and Tarau, P. (2004). TextRank: Bringing order into texts.

TextRank summarization with Okapi-BM25 ranking

2. Quantify relation strength (→ edges) between those shingles from their content.

"classical" TF-IDF

$$TFIDF(D_n, Q) = \sum_i^{|Q|} TF(q_i, D_n) \times IDF(q_i) \qquad TF(q_i, D_n) = log(|q_i \in D_n|)$$

Okapi BM25 "TF modification"

$$BM25(D_n, Q) = \sum_i^{|Q|} Okapi(q_i, D_n) \times IDF(q_i) \qquad Okapi(q_i, D_n) = \frac{TF(q_i, D_n)(k+1)}{TF(q_i, D_n) + k(1 - b + b\frac{|D_n|}{mean(|D|)})}$$

Main difference: the Okapi function flattens out much faster than a log-scaled Term Frequency function (alone).

https://en.wikipedia.org/wiki/Okapi_BM25

Barrios, F., López, F., Argerich, L., and Wachenchauzer, R. (2016).
Variations of the similarity function of TextRank for automated summarization.

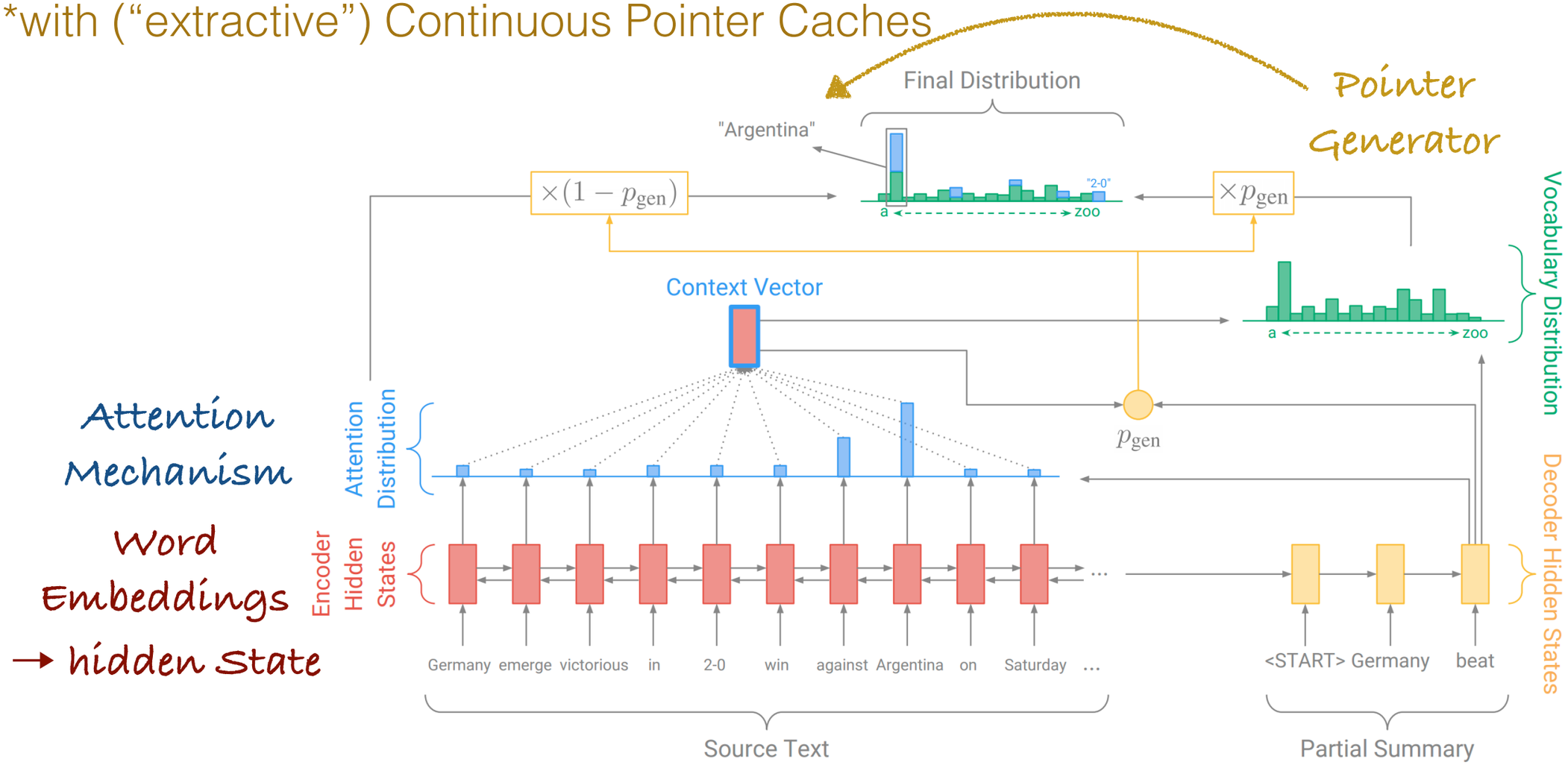
LexRank vs. TextRank

- Published simultaneously in 2004 by two independent groups
- Both are based on the same idea (graph similarity ranking)
- **LexRank** is part of a larger supervised summarization system ("MEAD") that uses features like sentence position and length.
- **LexRank** additionally covered a multi-document summarization approach (requiring post-processing; "CSIS")
- The **TextRank** authors expanded their work to keyword extraction

Erkan, G., and Radev, D.R. (2004). LexRank: Graph-based Lexical Centrality as Salience in Text Summarization.

Abstractive* summarization with Recurrent Neural Networks

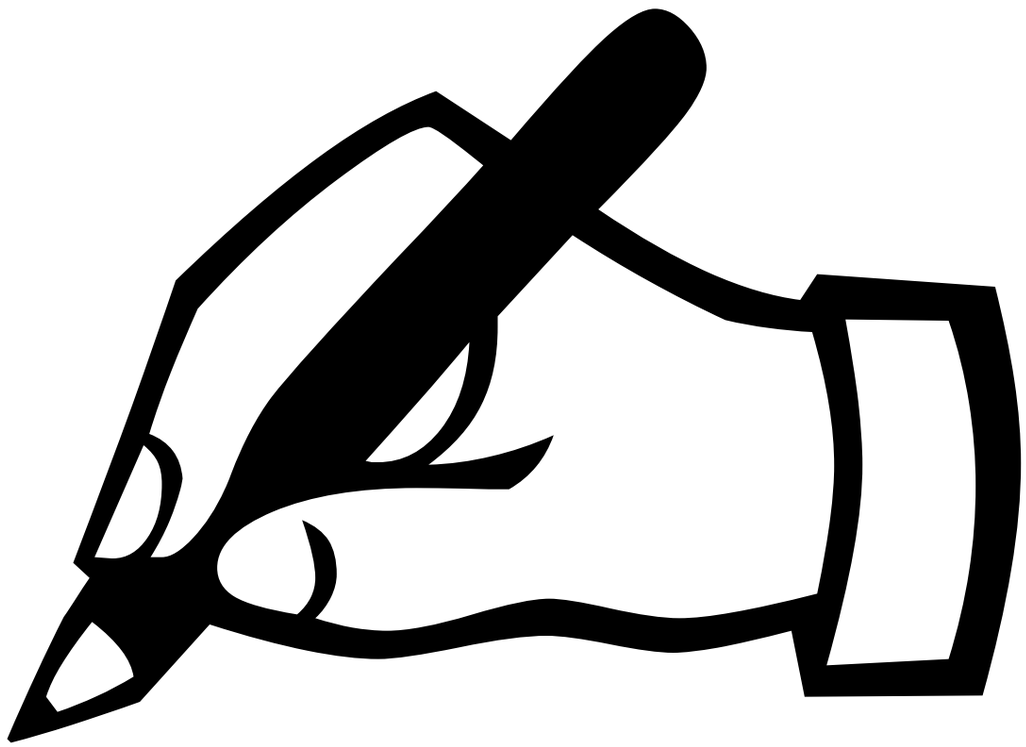
*with (“extractive”) Continuous Pointer Caches



Generates new text using the full sentence context (attention mechanism) from the current text (word embeddings), while at the same time it can copy facts/words (pointer generator) over to the new text.

See, A., Liu, P.J., and Manning, C.D. (2017). Get To The Point: Summarization with Pointer-Generator Networks.

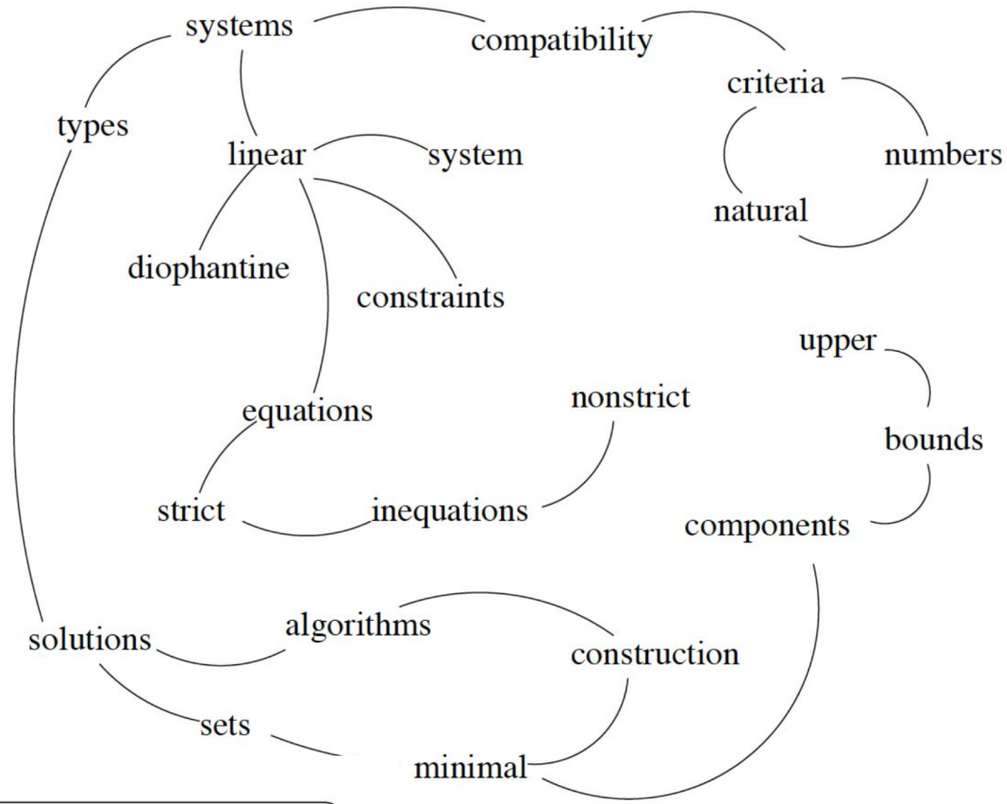
Practical 10:
Document summarization



Keyword extraction with TextRank using co-occurrence

see Extractive Summarization with TextRank for details

Compatibility of systems of linear constraints over the set of natural numbers. Criteria of compatibility of a system of linear Diophantine equations, strict inequations, and nonstrict inequations are considered. Upper bounds for components of a minimal set of solutions and algorithms of construction of minimal generating sets of solutions for all types of systems are given. These criteria and the corresponding algorithms for constructing a minimal supporting set of solutions can be used in solving all the considered types systems and systems of mixed types.



Keywords assigned by TextRank:
linear constraints; linear diophantine equations; natural numbers; nonstrict inequations; strict inequations; upper bounds

Keywords assigned by human annotators:
linear constraints; linear diophantine equations; minimal generating sets; non–strict inequations; set of natural numbers; strict inequations; upper bounds

Mihalcea, R., and Tarau, P. (2004). TextRank: Bringing order into texts.

Better keyword extraction using phrase-breaking lists: [RY]AKE

- Split text at **punctuation** and **stop-words** to get **phrases**.
- Next, proceed with the same "word graph" evaluation as TextRank
- The authors claim RAKE (viz, YAKE) performs (slightly) better than TextRank (viz, RAKE).

- Caveat emptor; RAKE only: manual stop-word tuning required
  - Performance depends on stop-word quality (wrt. both hits, and misses) [RAKE only]
  - Or requires a language where stop-words are not common inside keywords (e.g., in Spanish, "de", commonly used to chain noun phrases, might be an issue)
- YAKE does not need stop-words: Rake with a Yay!

Campos R. et al.. (2018). YAKE! Collection-independent Automatic Keyword Extractor.

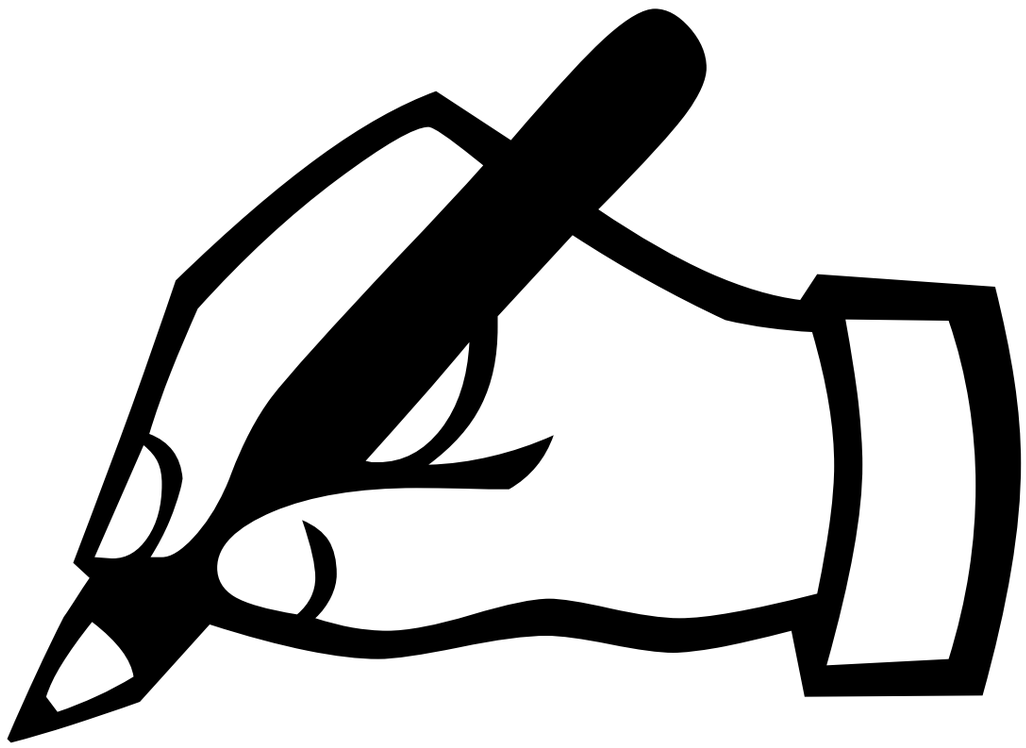
Rose, S., Engel, D., Cramer, N., and Cowley, W. (2010). Automatic keyword extraction from individual documents.

Phrasal keyword extraction alternatives

- Use the PageRank (here: "TextRank") approach as foundation

- But instead of stop-word delimiters or all n-grams, use
  - **Collocations** (already done, see unsupervised methods and PMI)
  - **Noun Phrases** (coming up with sequence modeling)

Practical 13: Keyword extraction



Sequence modeling

➡ Statistical analyses of [token] **sequences**

- Part-of-Speech (PoS) tagging & chunking
  - noun, verb, adjective, … & noun/verb/preposition/… phrases
- Named Entity Recognition (NER)
  - organizations, persons, places, genes, chemicals, …
- Information extraction
  - locations/times, phys. constants & chem. formulas, entity linking, event extraction, entity-relationship extraction, …

Part of speech corpora

I	ate	the	pizza	with	green	peppers	.
PRP	VB	DT	NN	IN	JJ	NN	.

- **Corpora** for the [supervised] **training** of PoS taggers
  - **Brown** Corpus (AE from ~1961)
  - British National Corpus: **BNC** (20th century British)
  - Wall Street Journal Corpus: **WSJ Corpus** (AE from the 80s)
  - American National Corpus: **ANC** (AE from the 90s)
  - Lancaster Corpus of Mandarin Chinese: **LCMC** (Books in Mandarin)
  - The **GENIA** corpus (Biomedical abstracts from PubMed)
  - **NEGR@** (German Newswire from the 90s)
  - Spanish and Arabian corpora should be (commercially…) available… ???

Best tip: Ask on the "corpora list" mailing list!

Methods for PoS tagging

I	ate	the	pizza	with	green	peppers	.
PRP	VB	DT	NN	IN	JJ	NN	.

- Automatic PoS tagging ➡ supervised machine learning
  - Maximum Entropy Markov models
  - **Conditional Random Fields**
  - **Convolutional Neural Networks**
  - Ensemble methods (bagging, boosting, etc.)

Probabilistic models for sequential data

- Origin: Kálmán filters (L[in.]Q[uad.]E[estim.]; Kálmán 1960)
- a.k.a. **Temporal** or **dynamic** **Bayesian networks**
  - **Static** process w/ **constant** model ➡ **temporal** process w/ **dynamic** model
  - Model structure and parameters are [still] **constant**
  - The model topology within a [constant] "**time slice**" is depicted

generative

- **Markov** Chain (MC; Markov. 1906)
- Hidden **Markov** Model (HMM; Baum et al. 1970)

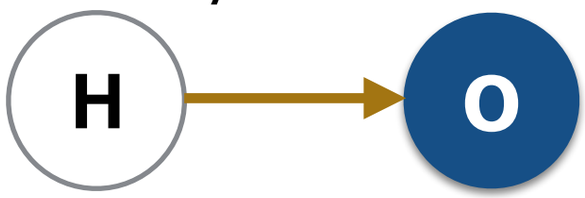
discriminative

- MaxEnt **Markov** Model (MEMM; McCallum et al. 2000)
- [**Markov**] Conditional Random Field (CRF; Lafferty et al. 2001)
  - Naming: all four models make the **Markov assumption** (see Markov property)

Probabilistic graphical models

- Graphs of hidden (blank) and **observed** (shaded) **variables** (vertices/nodes).
- The edges depict **dependencies**, and if directed, show [ideally causal] **relationships** between nodes.

directed ➡ Bayesian Network (BN)

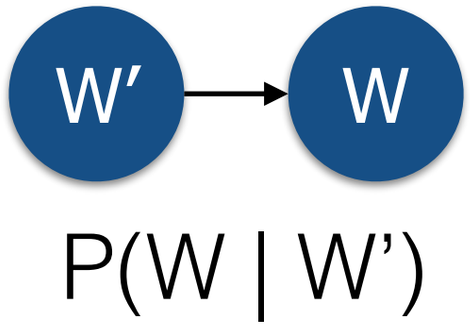


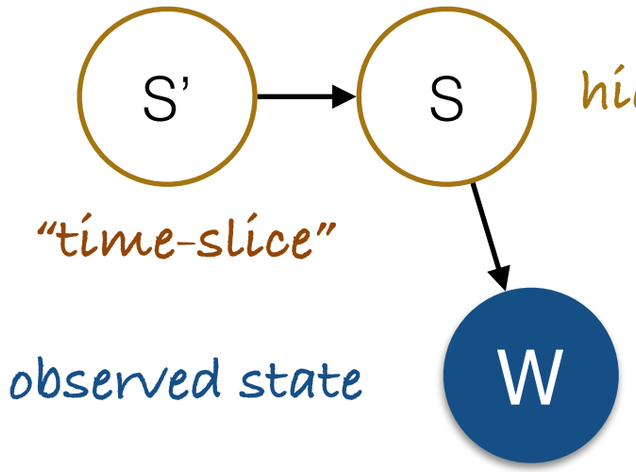
undirected ➡ Markov Random Field (MRF)

mixed ➡ Mixture Models

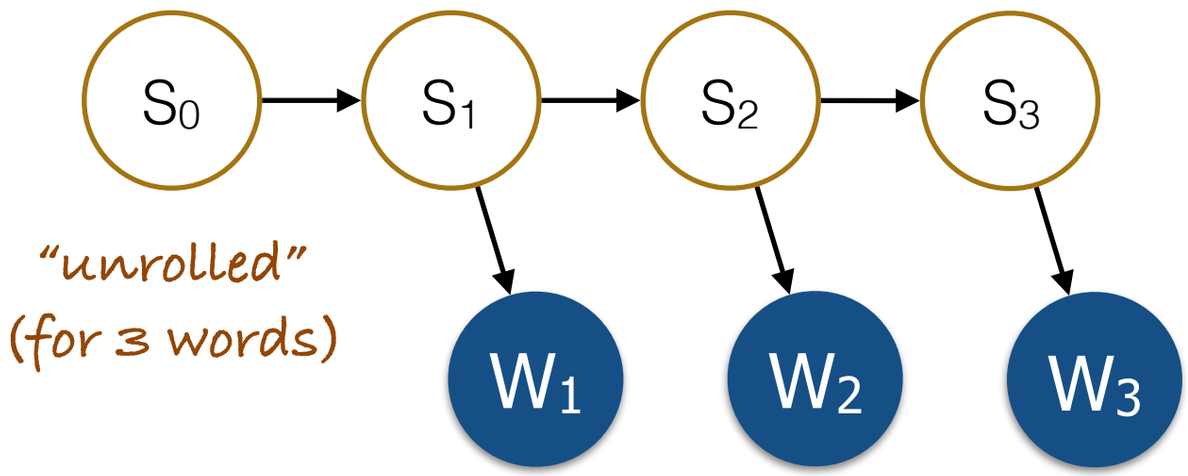
Koller & Friedman. Probabilistic Graphical Models. 2009

From a Markov chain to a Hidden Markov Model (HMM)



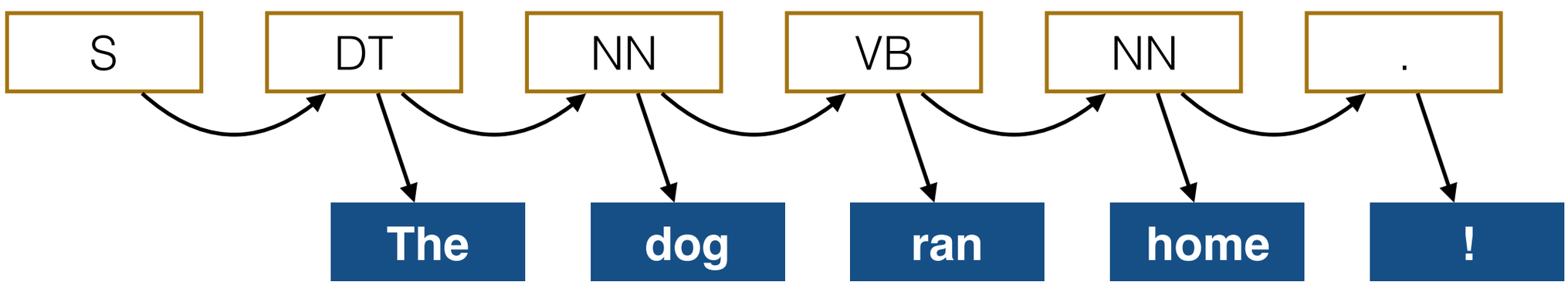


W depends on S and S in turn depends on S'



A language-based intuition for HMMs

- A Markov Chain: $P(W) = \prod P(w \mid w')$
  - assumes the observed words are in and of themselves the cause of the observed sequence.
- A HMM: $P(S, W) = \prod P(s \mid s') P(w \mid s)$
  - assumes the observed words are emitted by a hidden (not observable) sequence, for example the chain of part-of-speech-states.



NB, this is the "unrolled" model that does not depict the conditional dependencies

Three tasks solved by HMMs

Evaluation: Given a HMM, **infer** the P of the observed sequence (because a HMM is a generative model).

in Bioinformatics: Likelihood of a particular DNA element (e.g. a promoter)

Solution: **Forward Algorithm**

Decoding: Given a HMM and an observed sequence, **predict** the hidden states that lead to this observation.

in Statistical NLP: PoS annotation

Solution: **Viterbi Algorithm**

Training: Given only the graphical model and an observation sequence, **learn** the best [smoothed] parameters.

Solution: **Baum-Welch Algorithm**

all three algorithms are are implemented using dynamic programming

The two matrices of a HMM

a.k.a. "CPDs": Conditional Probability Distributions

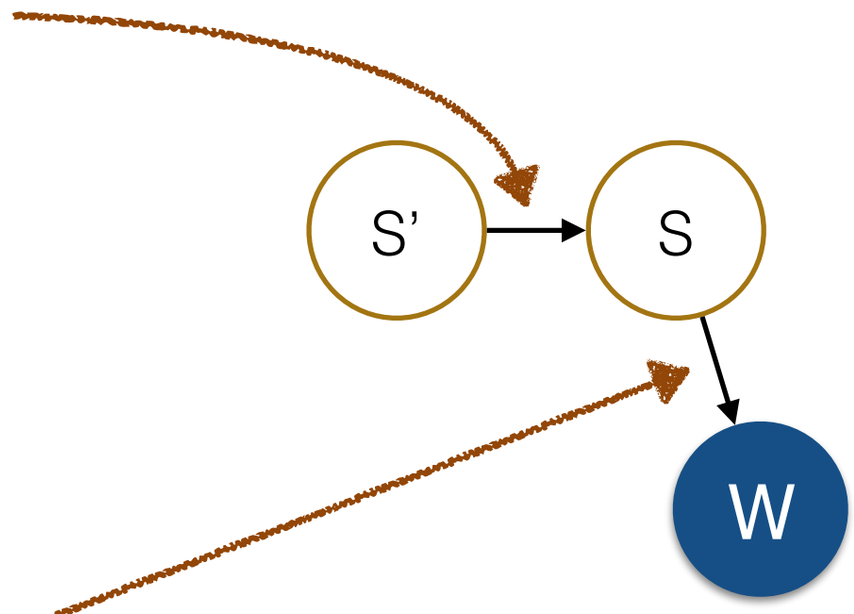
P(s	s')	DT	NN	VB	…
DT	0.03	0.7	0	
NN	0	0	0.5	
VB	0	0.5	0.2	
…				

(measured as discrete factor tables from annotated PoS corpora)

P(w | s)	$word_1$	$word_2$	$word_3$	…
DT	0.3	0	0	
NN	0.0001	0.002	0	
VB	0	0	0.001	
…				

underflow danger ➡ use "log Ps"!

Transition Matrix



Observation Matrix

very sparse (W is large)
➡ Smoothing!

Three limitations of HMMs

CRF, LSTM, Pointer caches, Attention

Markov assumption: The next state only depends on the current state.

Example issue: trigrams (long-range dependencies!)

MEMM

Output assumption: The **output** (observed value) is independent of all previous outputs (given the current state).

Example issue: word morphology (inflection, declension!)

CRF, Attention

Stationary assumption: Transition probabilities are independent of the actual time when they take place.

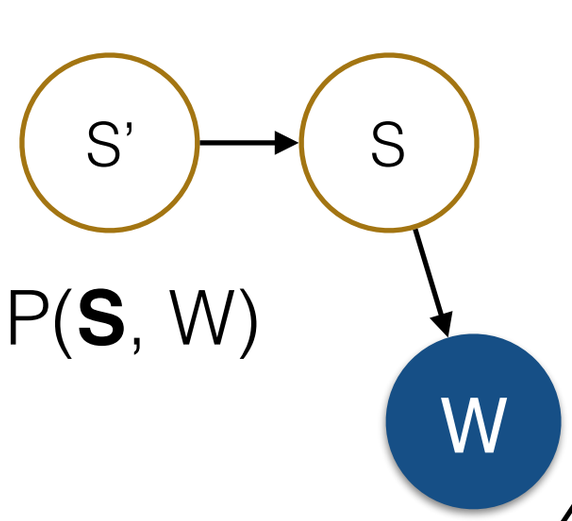
Example issue: position in sentence

(label bias problem, see next!)

From generative to discriminative sequence models

Hidden Markov Model

(first oder version)

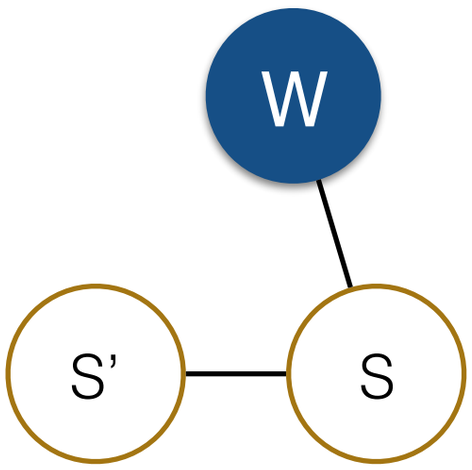


P(**S**, W)

generative model; lower bias is beneficial for small training sets

Maximum Entropy Markov Model

(first oder version)

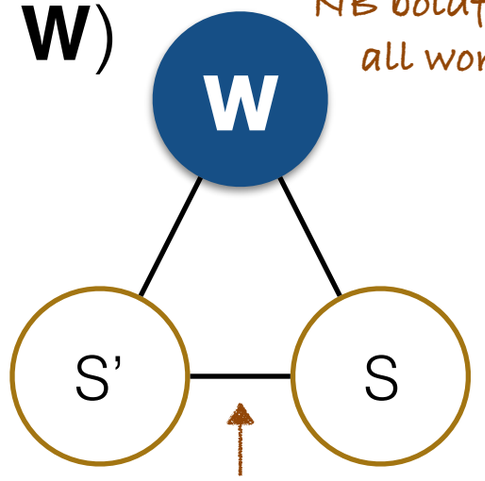


P(S | S', W)

Conditional Random Field

(linear chain version)

P(S | S', **W**)

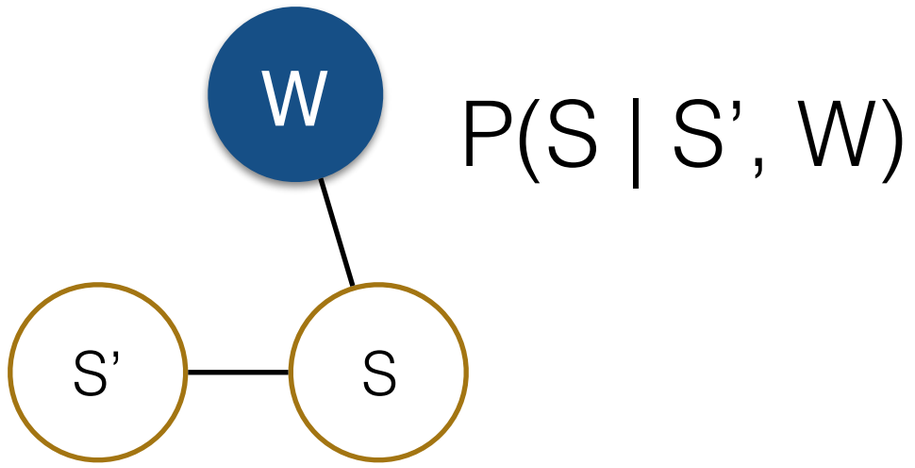


NB boldface W: all words!

this "clique" makes CRFs expensive to compute

Maximum Entropy Markov Models (MEMM)

Simplify training of $P(s \mid s', w)$
by splitting the model into $|S|$ separate
transition functions $P_{s'}(s \mid w)$ for each s'



P(S | S', W)

$$P(s \mid s', w) = P_{s'}(s \mid w)$$

Hello softmax, once again...

$$P_{s'}(s|w) = \frac{exp\left(\sum \lambda_i f_i(w, s)\right)}{\sum_{s^* \in S} exp\left(\sum \lambda_i f_i(w, s^*)\right)}$$

(The MaxEnt slide had {x, y} which here are {w, s}.)

Maximum entropy (MaxEnt 2/2) [again]

- In summary, MaxEnt is about selecting the "maximal" model p*:

$$p^* = \underset{p \in P}{argmax} \; - \sum_{x \in X} p(x) \sum_{y \in Y} p(y|x) \; log_2 \; p(y|x)$$

select some model that maximizes the conditional entropy…

- That obeys the following conditional equality constraint:

$$\sum_{x \in X} P(x) \sum_{y \in Y} P(y|x) \; f(x,y) = \sum_{x \in X, y \in Y} P(x,y) \; f(x,y)$$

…using a conditional model that matches the (observed) joint probabilities

- Next: Using, e.g., **Langrange multipliers**, one can establish the optimal λ parameters of the model that maximize the entropy of this probability:

$$p^*(y|X) = \frac{exp(\sum \lambda_i f_i(X,y))}{\sum_{y \in Y} exp(\sum \lambda_i f_i(X,y))}$$

"Exponential Model"

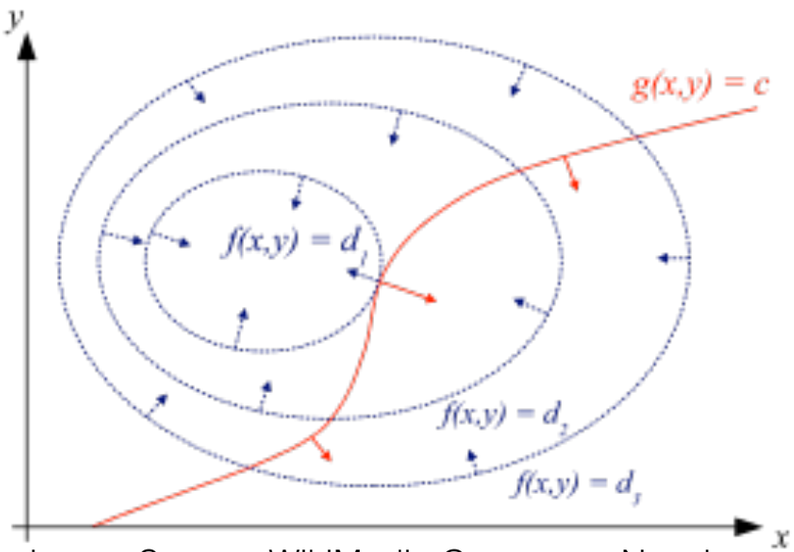
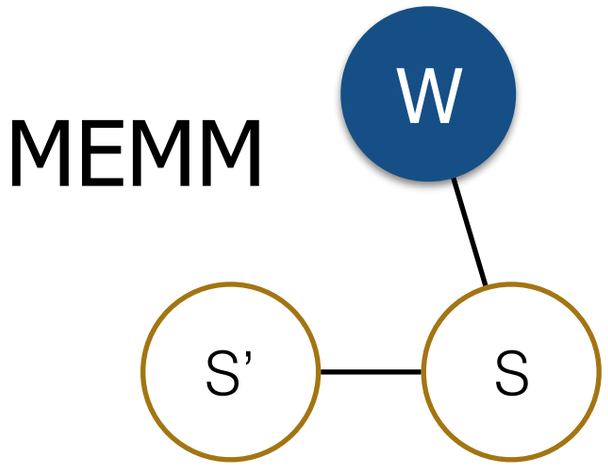


Image Source: WikiMedia Commons, Nexcis

The label bias problem of directional Markov models



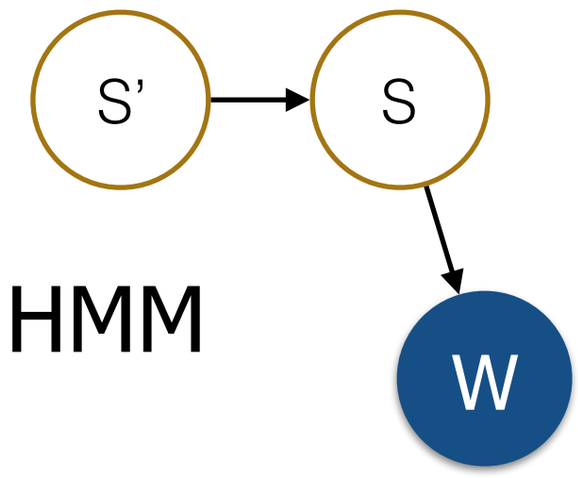
The robot wheels Fred around.

DT **NN** **VB** NN RB

The robot wheels were broken.

DT **NN** **NN** VB JJ

because of their directionality constraints, MEMMs & HMMs suffer from the label bias problem



The robot wheels are round.

DT **NN** **??** yet unseen!

Wallach. Efficient Training of CRFs. MSc 2002

The Markov random field (CRF prequel)

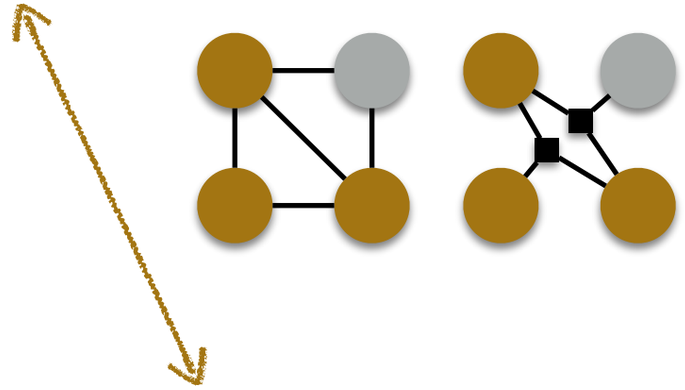
History class: Ising developed a linear field to model (binary) atomic spin states (Ising, 1924); the 2-dim. model problem then was solved by Onsager in 1944.

factor (clique potential)

$$P(X = \vec{x}) = \frac{\prod_{cl \in \vec{x}} \phi_{cl}(cl)}{\sum_{\vec{x} \in X} \prod_{cl \in \vec{x}} \phi_{cl}(cl)}$$

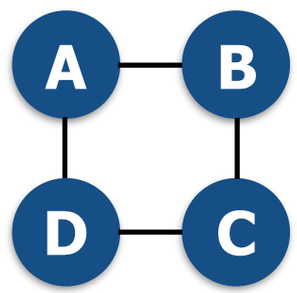
normalizing constant (partition function **Z**)

cl ... [**maximal**] **clique**; a subset of factors in the graph where every pair is connected



ϕ(A, B)			ϕ(B, C)			ϕ(C, D)			ϕ(D, A)		
a_0	b_0	30	b_0	c_0	100	c_0	d_0	1	d_0	a_0	100
a_0	b_1	5	b_1	c_0	1	c_0	d_1	100	d_1	a_0	1
a_1	b_0	1	b_0	c_1	1	c_1	d_0	100	d_0	a_1	1
a_1	b_1	10	b_1	c_1	100	c_1	d_1	1	d_1	a_1	100

factor table



factor graph

$P(a_1, b_1, c_0, d_1) =$
$10 \cdot 1 \cdot 100 \cdot 100 \div$
$7'201'840 =$
0.014

Conditional Random Field

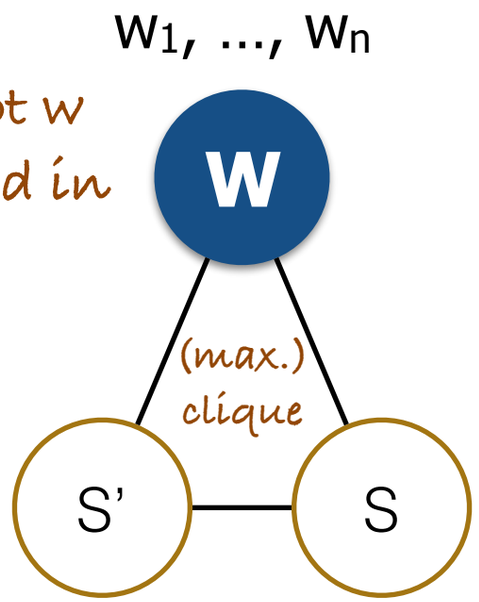
MRF:

$$P(X = \vec{x}) = \frac{\prod_{cl \in \vec{x}} \phi_{cl}(cl)}{\sum_{\vec{x} \in X} \prod_{cl \in \vec{x}} \phi_{cl}(cl)}$$

CRF:

$$P(Y = \vec{y} \,|X = \vec{x}) = \frac{\prod_{y \in \vec{y}} \phi_{cl}(y', y, \vec{x})}{\sum_{\vec{y} \in Y} \prod_{y \in \vec{y}} \phi_{cl}(y', y, \vec{x})}$$

note W (upper-case/bold), not w (lower-case): all words are used in each step!



The label bias problem is "solved" by conditioning the MRF Y-Y' on the entire observed sequence.

$$P(S|W) = \frac{exp\left(\sum \lambda_i f_i(W, s', s)\right)}{\sum_{s^* \in S} exp\left(\sum \lambda_i f_i(W, s'^*, s^*)\right)}$$

Models a per-state **exponential function** of joint probability over the **entire** observed **sequence** W.

Wallach. Conditional Random Fields: An introduction. TR 2004

Parameter estimation and L2 regularization of CRFs

(regularization reduces the effects of overfitting)

- For training $\{\mathrm{Y}^{(n)}, \mathrm{X}^{(n)}\}_{n=1}^{N}$ sequence pairs with K features
- Parameter estimation using **conditional log-likelihood**

$$\lambda = \underset{\lambda \in \Lambda}{argmax} \sum_{n}^{N} log\ P(Y^{(n)}|X^{(n)}; \lambda)$$

(c.f. Langrange multipliers for MaxEnt)

- Substitute $log\ \mathrm{P}(\mathrm{Y}^{(n)} \mid \mathrm{X}^{(n)})$ with log **exponential model**

$$\ell(\lambda) = \sum_{n=1}^{N} \sum_{y \in Y^{(n)}} \sum_{i=1}^{K} \lambda_i f_i(y', y, X^{(n)}) - \sum_{n=1}^{N} log\ Z(X^{(n)})$$

normalizing constant (partial function Z)

- Add a penalty for parameters with a to high **L2-norm**

$$\ell(\lambda) = \sum_{n=1}^{N} \sum_{y \in Y^{(n)}} \sum_{i=1}^{K} \lambda_i f_i(y', y, X^{(n)}) - \sum_{n=1}^{N} log\ Z(X^{(n)}) - \sum_{i=1}^{K} \frac{\lambda_i^2}{2\sigma^2}$$

L2: Euclidian norm ($\sum$ of squared errors)

$\frac{\|\Lambda\|_2^2}{2\sigma^2}$

free regularization parameter

Model summaries: HMM, MEMM, CRF

- A **HMM**
  - **generative** model
  - **efficient** to learn and deploy
  - trains with **little data**
  - generalizes well (**low bias**)
- A **MEMM**
  - better labeling **performance**
  - modeling of **features**
  - **label bias** problem (as HMMs)
- **CRF**
  - conditioned on **entire observation**
  - **complex features** over full input
  - training time **scales exponentially** to the number of states
  - $O(NTM^2G)$
  - N: # of sequence pairs;
    T: E[sequence length];
    M: # of (hidden) states;
    G: # of gradient computations for parameter estimation

a PoS model w/45 states and 1 M words can take a week to train...

Sutton & McCallum. An Introduction to CRFs for Relational Learning. 2006

Noun and verb phrase chunking with BIO-encoded labels

"shallow parsing"

a pangram (hint: check the letters)

The brown fox	quickly jumps	over	the lazy dog	.
DT JJ NN	RB VBZ	IN	DT JJ NN	.
B-N I-N I-N	B-V I-V	O	B-N I-N I-N	O

Performance (2nd order CRF) ~ 94%
Main problem: embedded & chained NPs (N of N and N)

Chunking is "more robust to the highly diverse corpus of text on the Web" and [exponentially] faster than [deep] parsing.

Banko et al. Open Information Extraction from the Web. IJCAI 2007 a paper with over 700 citations

Wermter et al. Recognizing noun phrases in biomedical text. SMBM 2005 error sources

PoS tagging and lemmatization for Named Entity Recognition (NER)

N.B.: This is all supervised (i.e., manually annotated corpora)!

de facto standard PoS tagset {NN, JJ, DT, VBZ, ...}
Penn Treebank

noun-phrase (chunk)

Token	PoS	Lemma	NER
Constitutive	JJ	constitutive	O
binding	NN	**binding**	O
to	TO	to	O
the	DT	the	O
peri-κ	NN	peri-kappa	B-DNA
B	NN	B	I-DNA
site	NN	site	I-DNA
is	VBZ	be	O
seen	VBN	see	O
in	IN	in	O
monocytes	NNS	monocyte	B-cell
.	.	.	O

chunk

Begin-Inside-Outside of the (relevant) token
B-I-O
chunk encoding

common alternatives:
I-O
I-E-O
B-I-E-W-O

End token

(unigram) Word

Named Entity Recognition (NER)

How much training data do I need? "corpora list"

Image Source: v@s3k [a GATE session; http://vas3k.ru/blog/354/]

The departure of Mr Hogan, who originally moved to British Midland as service director from Hertz International in 1997, surprised aviation analysts, as it was believed that he had been brought into the senior executive team of the airline, as part of the group's management succession planning.

He played a leading role in the strategic planning for the rebranding of the airline as BMI in preparation for its entry this year into the scheduled long haul market with the launch of services from Manchester to the US.

BMI has taken on the costs of entry into the North Atlantic market at an unfortunate time, as airlines in North America are facing the toughest conditions for 20 years with many carriers plunging into loss.

BMI, in which Lufthansa of Germany and SAS Scandinavian Airlines each own stakes of 20 per cent, suffered a 26 per cent fall in pre-tax profits last year from £11.1m ($15.7m) to £8.2m on a turnover that grew 16.5 per cent to £739.2m.

In the first six months this year it is understood that passenger volumes have fallen by around two per cent. The share of available seats filled, the load factor, has declined by around two percentage points, but this has been offset by a strong increase in yields, or average fare levels, by more than ten per cent.

Date
Location
Money
Organization
Percentage
Person

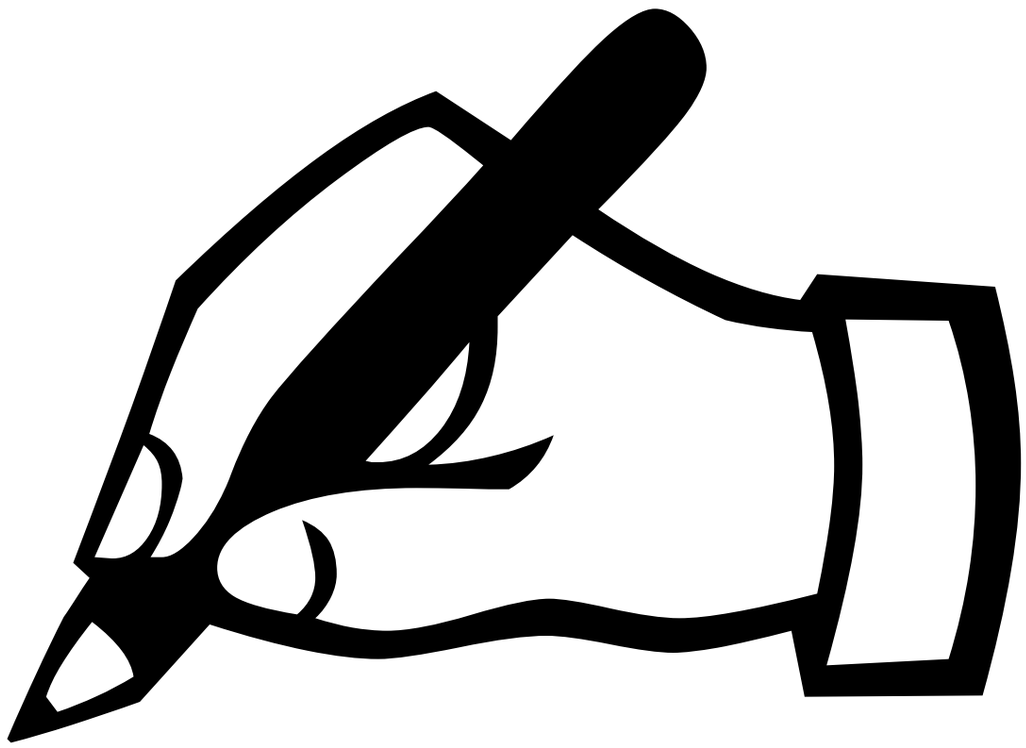
➡ Conditional Random Field

➡ Ensemble Methods; +SVM, HMM, MEMM, … ➡ `pyensemble`

NB these are corpus-based approaches (supervised)

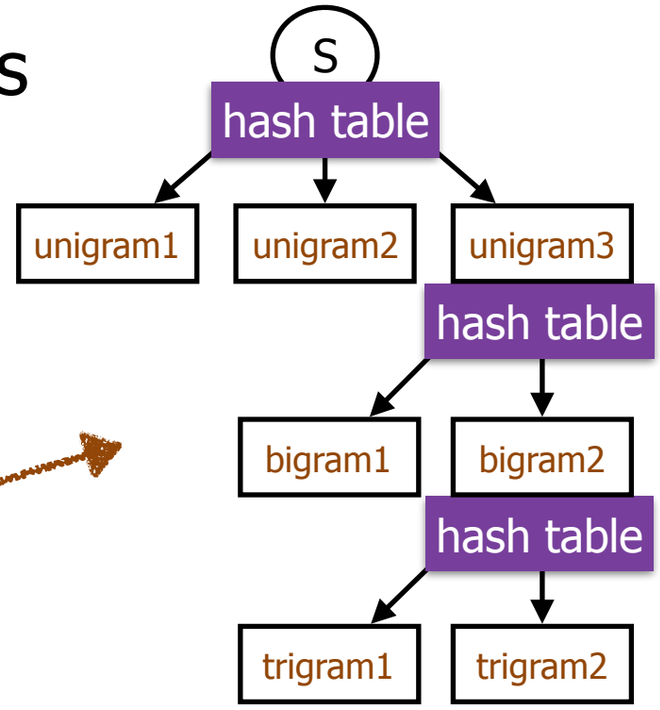
CoNLL03: http://www.cnts.ua.ac.be/conll2003/ner/

Practical 14:
Sequence tagging (PoS, NER)



Named Entity Linking: Dictionary matching

- Finding all tokens that match dictionary entries
  - hash table lookups: constant complexity - $O(1)$
- **Exact**, single token matches
  - regular hash table lookup (e.g., MURMUR3)
- Exact, multiple tokens
  - **prefix trie** of (hashed) tokens
- **Approximate**, single tokens
  - use some string metric - but do not compare all n-to-m cases...
- **Approximate**, multiple tokens
  - various "fuzzy" sequence matching algos



e.g. Jaro-Winkler similarity, Character n-gram similarity, or Locality Sensitive Hashing

Named Entity Linking: Entity disambiguation

- NER & matching is insufficient to link the exact entity:
  - Same name, different entities: Disambiguating the correct linkage.
  - Persons, locations, stock tickers, company names, genes, proteins, …
- Solution: **Homonym-specific representation learning**
  - By learning entity embeddings/context vectors

A related concern is entity relevance:
Because typically, when relevance is low, disambiguation is harder.

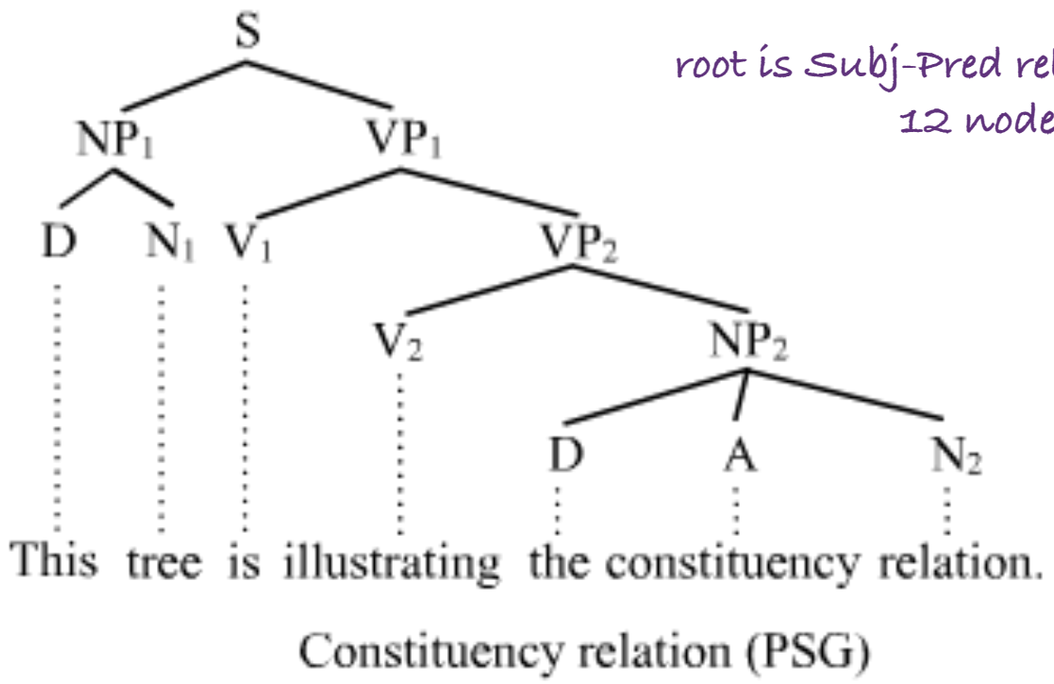
Language Processing

Madrid Summer School on
Advanced Statistics and Data Mining

Florian Leitner
Selerity, Inc.; Data Catalytics, SL; IE University
leitner@datacatytics.com

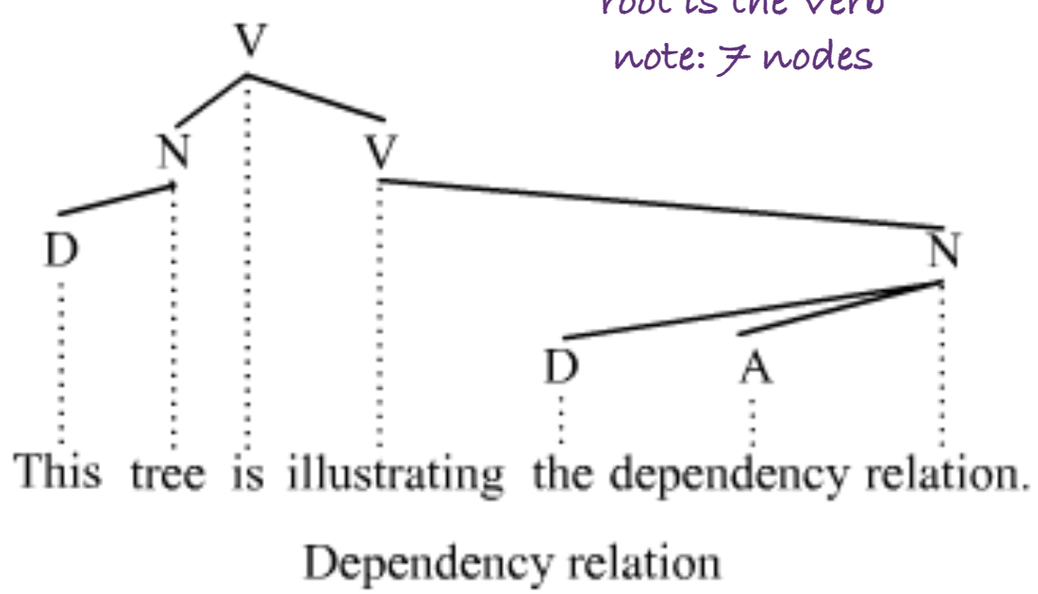
Detecting grammatical (sentence) structure

Phrase-structure (aka. **constituency**) vs. **dependency** grammars



root is Subj-Pred rel.
12 nodes

Constituency relation (PSG)



root is the Verb
note: 7 nodes

Dependency relation

https://en.wikipedia.org/wiki/Phrase_structure_grammar

P-S Grammars: **Chomsky**; Dependency Grammars: **Tesnière**

Dependency relations can be annotated with a linear-time parser.

note the one-to-many constituency vs. the one-to-one dependency relations

Tesnière's dependency relations (1959)

He ate pizza with anchovies. He ate pizza with friends.

ate(he, pizza)
~~ate(he, with anchovies)~~
has(pizza, anchovies)

Relationships

ate(he, pizza)
ate(he, with friends)
~~has(pizza, friends)~~

NB: Dependencies cannot capture **phrasal structure** (subject, object, verb phrase, etc.), and in particular, **word order**.

which can be a benefit: some languages have a free word order, e.g. Turkish or Czech
reminder: clauses and collocations are special phrasal structures

Dependency parsing 1/2

- Transition-based, arc-standard, shift-reduce, greedy parsing.
- The default approach to dependency parsing today is O(n).

  - **Transition-based**: Move from one token to the next.
  - **Arc-standard**: assign arcs when the dependent token (at the arrowhead) is fully resolved (common alternative: arc-eager → assign the arcs immediately).
  - **Shift-reduce**: A stack of words and a stream buffer: either shift next word from the buffer to the stack or reduce a word from the stack by "arcing".
  - **Greedy**: Make locally optimal transitions (assume independence of arcs).

    "don't look into your buffer"

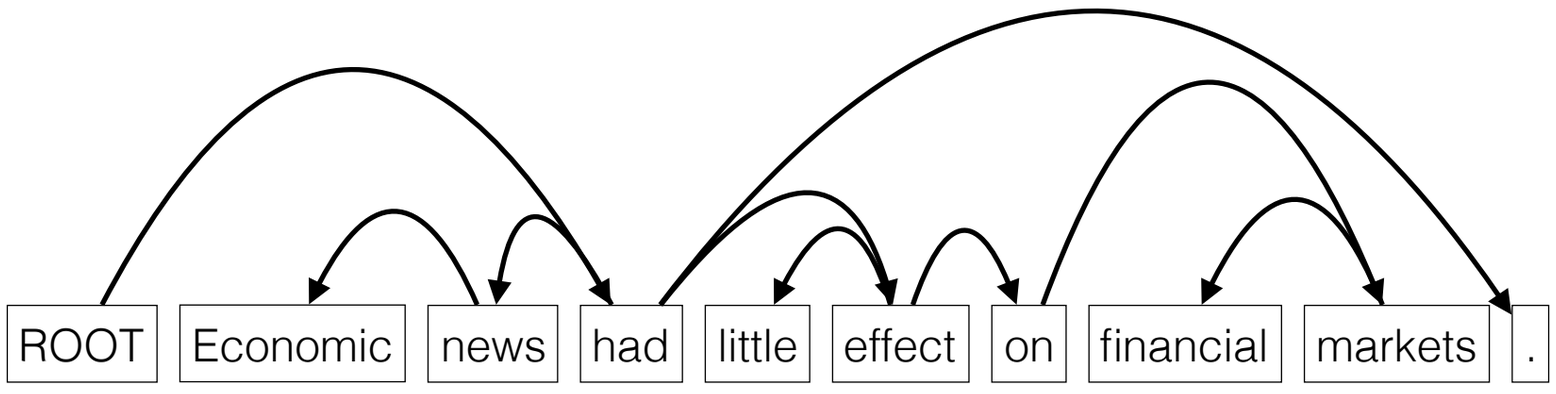
A shift-reduce parse

Buffer | Economic | news | had | little | effect | on | financial | markets | . |

Stack

. (bottom)
markets
financial
on
effect
little
had
news
Economic
ROOT (top)

Shift: Economic	Shift: financial
Shift: news	Shift: markets
Left-arc: Economic←news	Left-arc: financial←markets
Shift: had	Right-arc: on→markets
Left-arc: news←had	Right-arc: effect→on
Shift: little	Right-arc: had→effect
Shift: effect	Shift: .
Left-arc: little←effect	Right-arc: had→.
Shift: on	Right-arc: ROOT→had



Dependency Parsing. Kübler et al., 2009

A shift-reduce parse

Shift: Economic

Shift: news

Left-arc: Economic←news

Shift: had

Left-arc: news←had

Shift: little

Shift: effect

Left-arc: little←effect

Shift: on

Shift: financial

Shift: markets

Left-arc: financial←markets

Right-arc: on→markets

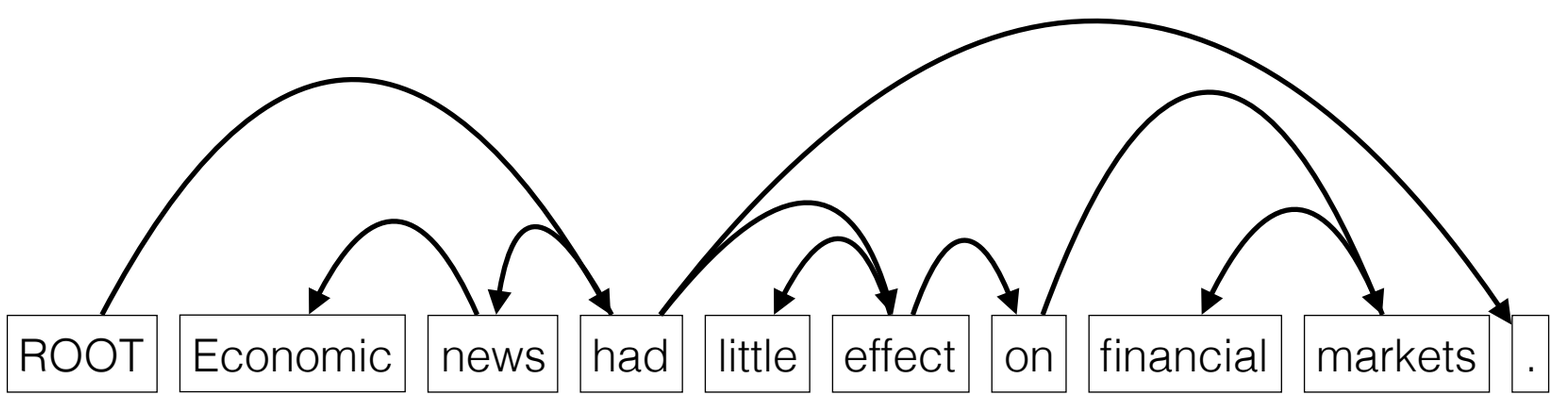
Right-arc: effect→on

Right-arc: had→effect

Shift: .

Right-arc: had→.

Right-arc: ROOT→had



Dependency Parsing. Kübler et al., 2009

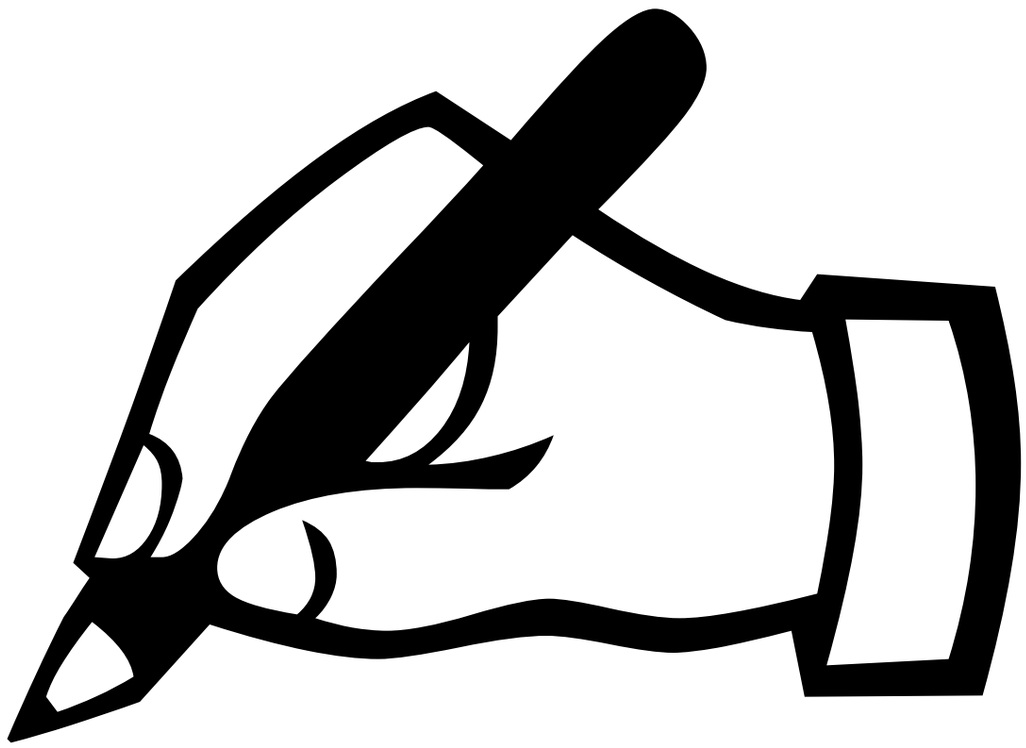
Dependency parsing 2/2

- (Arc-standard) Transitions: **shift** or **reduce** (left-arc, right-arc)
- Transitions are chosen using some classifier
  - Maximum entropy classifier, support vector machine, single-layer perceptron, perceptron with one hidden layer (→ Stanford parser, 2014 edition, SpaCy v1), more complex deep nets (→ Google's SyntaxNet, SpaCy v2)

i.e., probably outdated on latest versions…

- Main issues:
  - Few large, well annotated training corpora ("dependency **treebanks**"). Biomedical domain: GENIA; Newswire: WSJ, Prague, Penn, …
  - **Non-projective** trees (i.e., trees with arcs crossing each other; common in a number of other languages, e.g. German) with arcs that have to be drawn between nodes that are not adjacent on the stack.

Practical 16: Dependency parsing



Four approaches to relationship/event extraction

- **Co-mention window**
  - E.g.: if ORG and LOC entity within same sentence and no more than x tokens in between, treat the pair as a hit.
  - Low precision, high recall; trivial, many false positives.
- **Dependency parsing**
  - If a path covering certain nodes (e.g. prepositions like "in/IN" or predicates [~verbs]) connects two entities, extract that pair.
  - Balanced precision and recall, computationally expensive.
- **Pattern extraction** (over the seq. tags)
  - e.g.: <ORG>+ <IN> <LOC>+ (IN: preposition)
  - High precision, low recall; cumbersome, but very common.
  - Pattern **learning** can help.
- **Machine Learning**
  - Features for sentences with entities and some classifier (e.g., SVM, neural net, MaxEnt, Bayesian net, …) (token-distance, num. of tokens between the entities, tokens before/after them, etc.)
  - Highly variable milages.
    … but loads of fun in your speaker's opinion :)

Free open source text mining and NLP software

- R. Řehůřek's "RaRe's" Gensim
  - Gensim - Python [LGPL]
- M. Honnibal's spaCy
  - spaCy - Python
- Natural Language ToolKit
  - NLTK - Python
- Stanford NLP Framework
  - CoreNLP - **Java** [GPL]
- J.D. Choi's ClearNLP
  - NLP4J - **Java**
- Apache OpenNLP Framework
  - OpenNLP (& OpenNER) - **Java**
- A2I's AllenNLP
  - AllenNLP - **PyTorch** - Python/CUDA
- DMLC's Gluon NLP
  - gluon-nlp - **MXNet** - Python/CUDA
- Intel AI Lab's NLP Architect
  - NLP Architect - **TF**/**DyNet** - Python
- Chris Dyer's DyNet
  - DyNet - C++
- Facebook's fastText
  - fastText - C++/Python
- Google's BERT model
  - Versions for **TensorFlow** and **PyTorch**

State-of-the-Art NLP Models in Deep Learning

- SOTA concepts
  - Transfer learning (self-supervised)
  - Self-Attention (linguistic structures, e.g., coreferences) & contextual embeddings (polysemous words)
  - Top-K & nucleus sampling decoders (natural language generation, NLG)
- Deep Learning platforms
  - Name (recommendation)
  - TensorFlow (industry, GCP)
  - PyTorch (academia)
  - MXNet (Apache/OSS, AWS)
  - DyNet (NLP research)
- Google's **BERT** transformer
  - Impl. in TensorFlow and PyTorch
- CMU's **XLNet** (improved BERT)
  - Very new; as of June 2019 TensorFlow impl. only (PyTorch likely to come)
- AllenAI's **ELMo** embeddings
  - Impl. in PyTorch; contributed contextual embeddings; LSTM-based
- OpenAI's **gpt-2** nucleus sampling (NLG SotA)
  - Impl. in TensorFlow and PyTorch

MBA Rule #1:
Always Counter Buzz Words with Buzz Words

@TomHCAnderson
tomhcanderson.com

The End.