POLITÉCNICA
"Ingeniamos el futuro"
CAMPUS
DE EXCELENCIA
INTERNACIONAL

Madrid Summer School on
Advanced Statistics and Data Mining

Module C9 :: Text Mining

4th July - 8th of July 2016

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

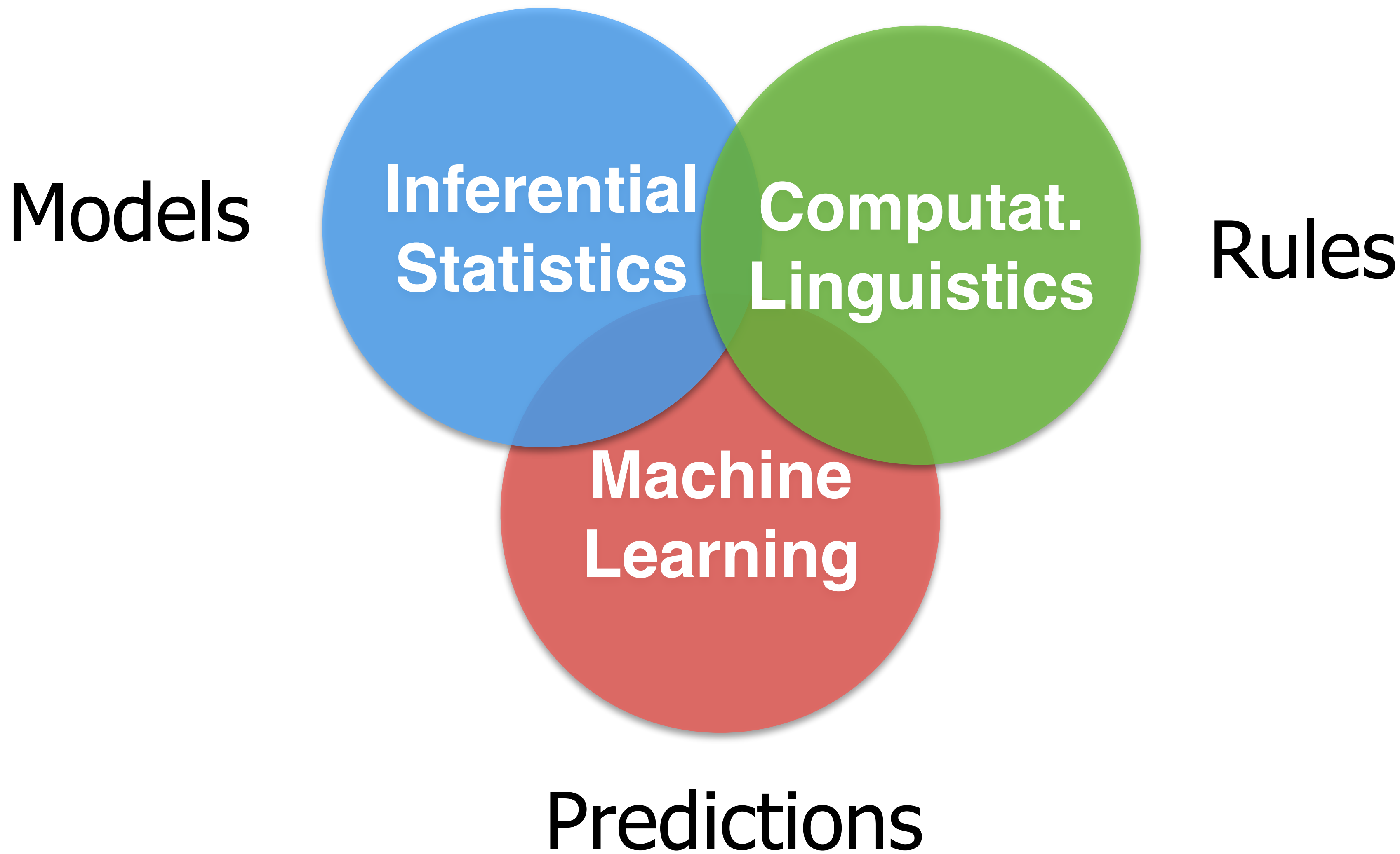
Text Mining 1
Introduction

Madrid Summer School on
Advanced Statistics and Data Mining

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

“Text mining” or “text analytics”

The discovery of new or existing facts from text by applying natural language processing (**NLP**), statistical learning techniques, or both.



Is language understanding key to artificial intelligence?

"Her" Movie, 2013

"The singularity: 2030" Ray Kurzweil
(Google's director of engineering)

"Watson" & "CRUSH" IBM

"predict crimes before they happen"

cognitive computing:
"processing information more like a human than a machine"

Applications of text mining and language processing

Text mining

(Web) Search engines

Information retrieval

Spam filtering

Document classification

Twitter brand monitoring

Opinion mining

Finding similar items (Amazon)

Content-based recommender systems

Event detection in e-mail

Information extraction

Language processing

Spelling correction

Statistical language modeling

Siri (Apple) and Google Now

Language understanding

Website translation (Google)

Machine translation

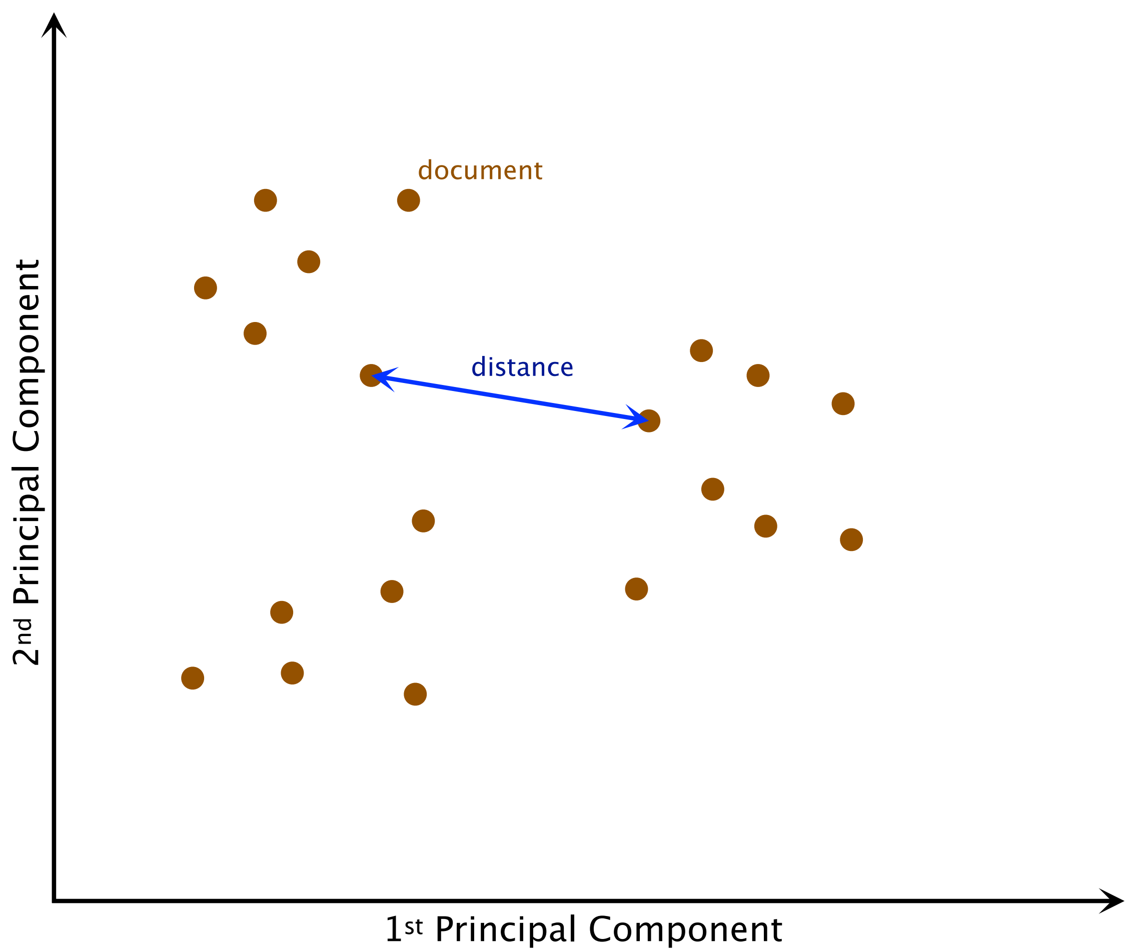
"Clippy" assistant (Microsoft)

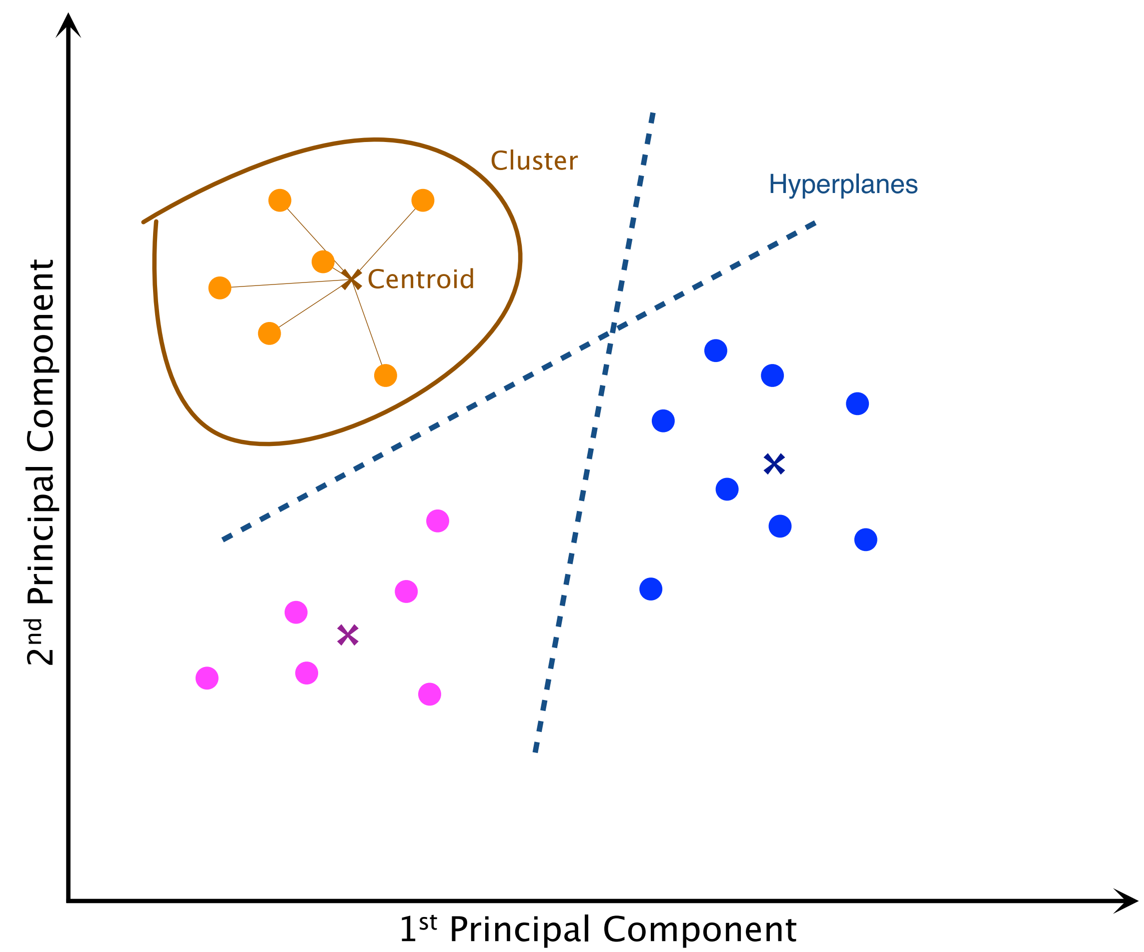
Dialog systems

Watson in Jeopardy! (IBM)

Question answering

Document or text classification and clustering





Supervised ("learning to classify *from examples*", e.g., spam filtering)

vs.

Unsupervised ("exploratory grouping", e.g., topic modeling)

Tokens and n-grams

Character-based,
Regular Expressions,
Probabilistic, …

This is a sentence.

"tokenization"

Token or Shingle

Token n-grams

unigram
shingle

This | is | a | sentence | NB: | .

bigram
2-shingle

This is | is a | a sentence | sentence .

trigram
3-shingle

This is a | is a sentence | a sentence .

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…

String matching with **regular expressions**

delimiters - like "quotes" in strings
(will be omitted for clarity)

- `/^[^@]{1,64}@\w(?:[\w.-]{0,254}\w)?\.\w{2,}$/`
  - (provided `\w` has Unicode support)
  - http://ex-parrot.com/~pdw/Mail-RFC822-Address.html

- username@server.domain.tld
- florian.leitner@upm.es
- 123$%&-@dfa-asdf.asdf-123.wow

Regular expressions quick reference 1/2

- String literals abc…
  - abc → axc **abc** xca
- Wild card . (any character)
  - a.c → **axc** **abc** xca
- Start ^ or end $ **of string**
  - ^a.c → **axc** abc xca
  - bc$ → axc abc xca

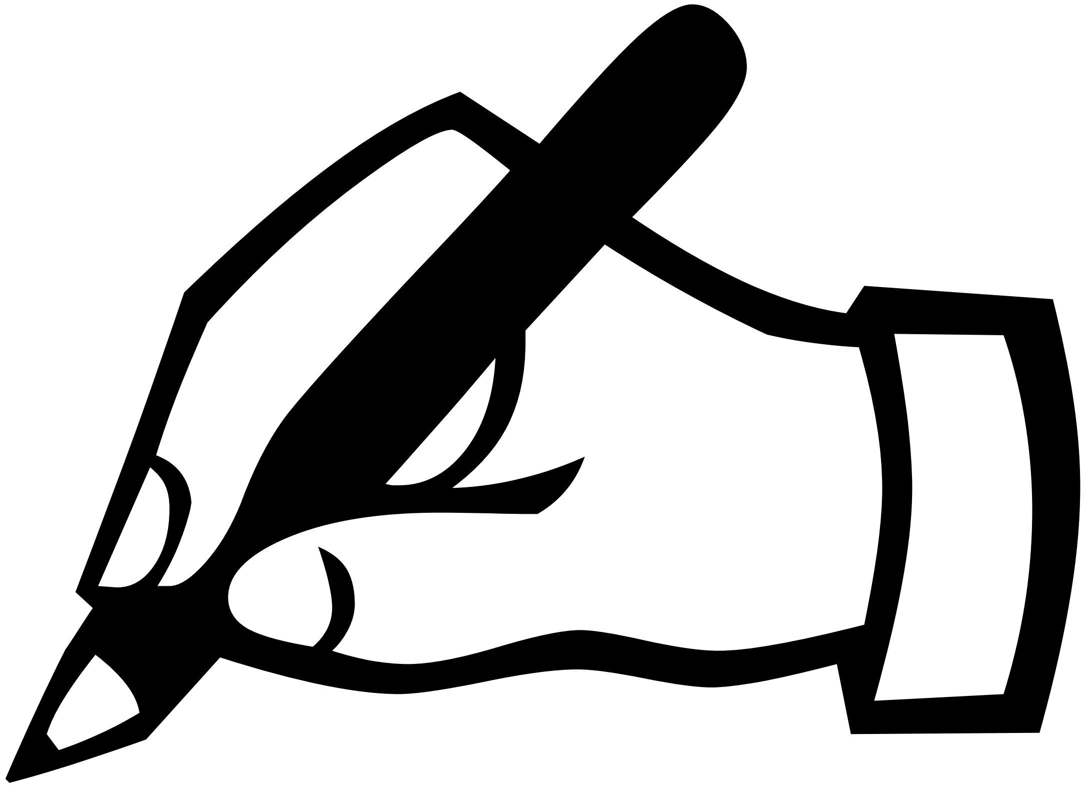
no match: …bc not at end of string
(compare with word boundaries!)

- Word boundaries \b
  - \bxc → axc abc **xc**a
  - bc\b → axc a**bc** xca
- Character choice […]
  - [ax][xbc][ac] → **axc** **abc** **xca**
- Negations [^…] ("not")
  - [ax][^b][^xb] → **axc** abc **xca**

Regular expressions quick reference 2/2

- Groups (can be **referenced** for repetitions and substitutions):
  - `(…)` and `(…)(…)\1\2`
  - `a(bc)\1` → ac abc **abcbc** **abcbc**bc
- Pattern repetitions
  - `?` "zero or one"; `*` "zero or more"; `+` "one or more":
  - `a(bc)?d` → **ad** abc **abcd** abcbcd abcbcbcd
    `a(bc)*d` → **ad** abc **abcd** **abcbcd** **abcbcbcd**
    `a(bc)+d` → ad abc **abcd** **abcbcd** **abcbcbcd**
- Counted pattern repetitions
  - `{n}` "exactly n times", `{n,}` "n or more times", `{n,m}` "n to m times"
  - `a(b){2}c` → ac abc **abbd** abbbc
    `ab{2,}c` → ac abc **abbd** **abbbc**
    `ab{0,2}c` → **ac** **abc** **abbd** abbbc

Practical: Tokenization



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**

Word vectors

Collections of vectorized texts: **Inverted index**

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

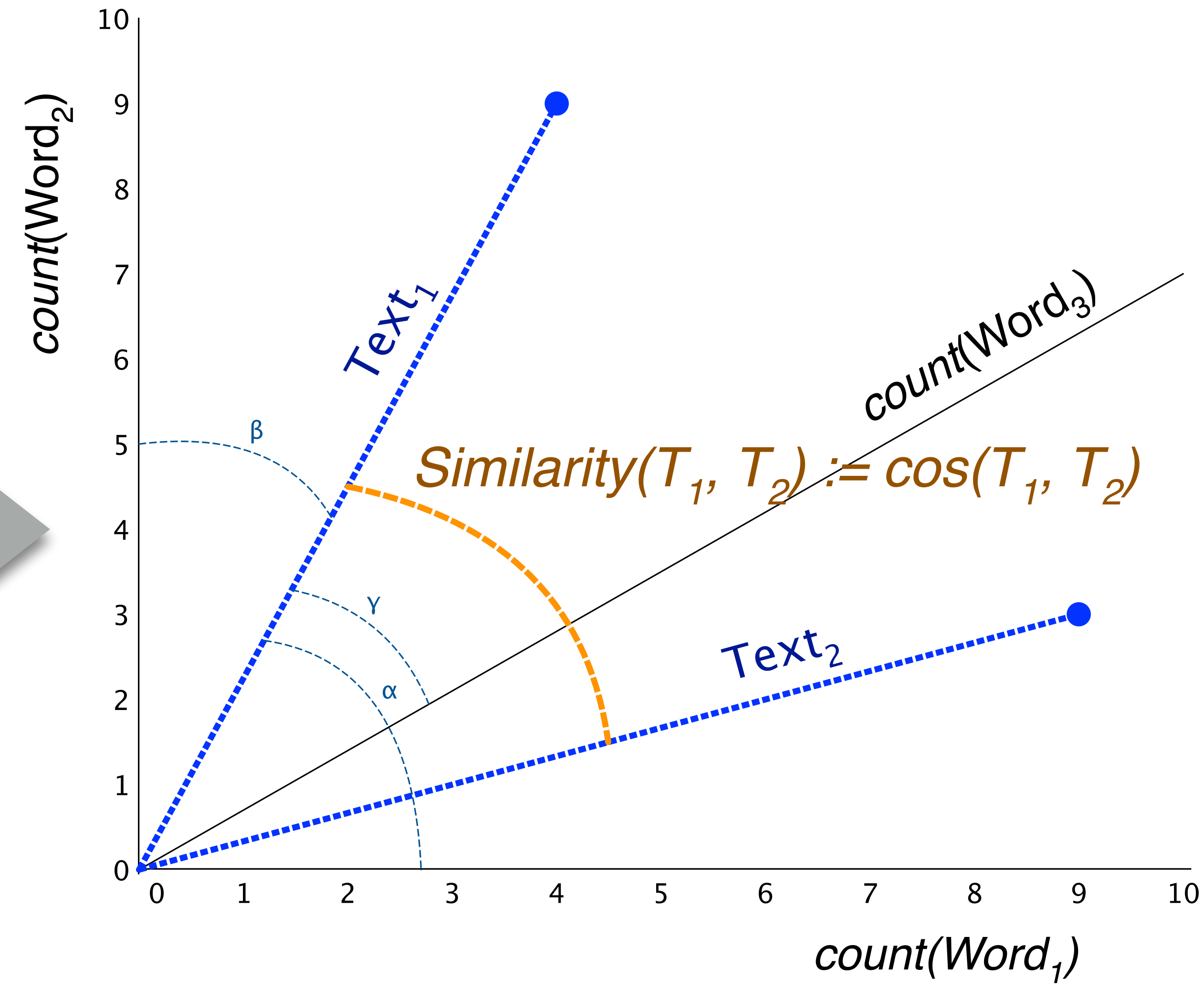
Text 2: If the mountain will not go to Moses, then Moses must go to the mountain.

each token/word is a dimension!

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 →

Comparing word vectors: **Cosine similarity**

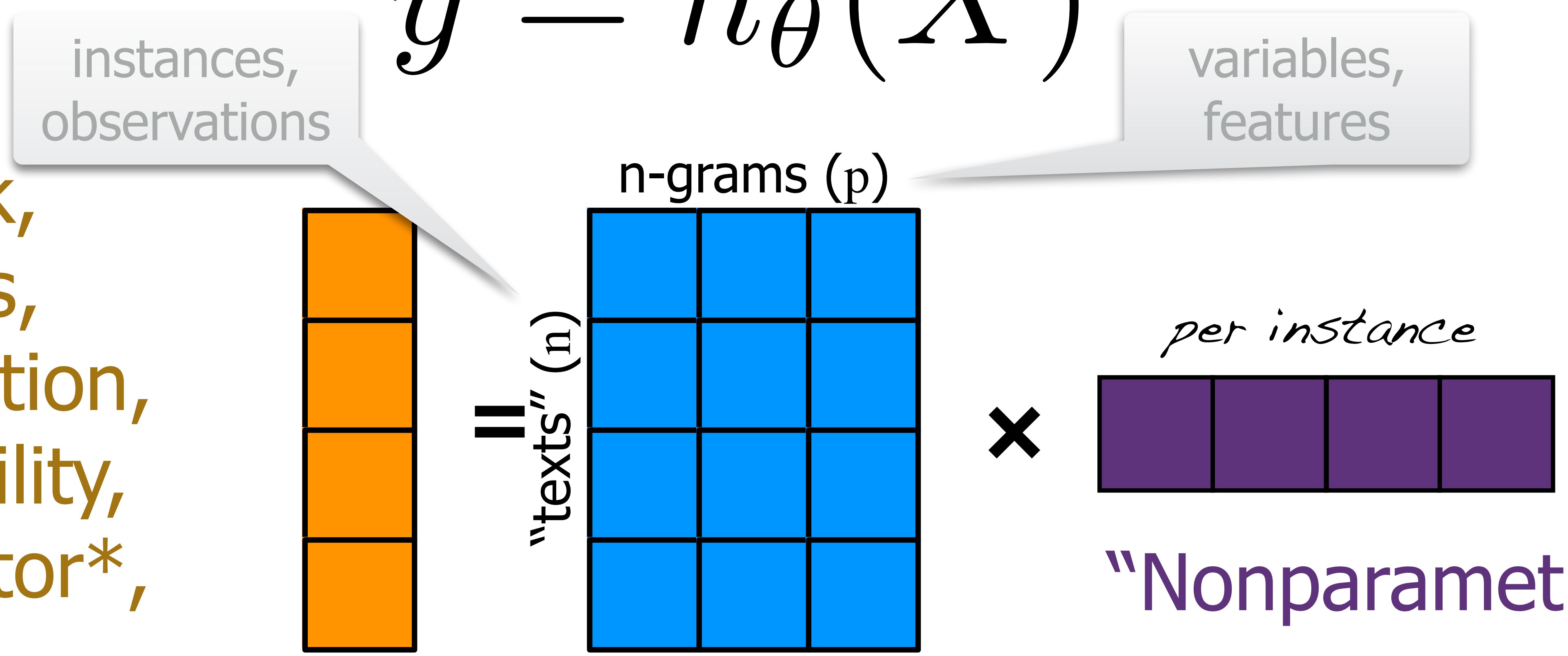


Terrier Sphinx Xapian INDRI Lucene

The essence of statistical learning

$$y = h_\theta(X)$$

Rank,
Class,
Expectation,
Probability,
Descriptor*,
…



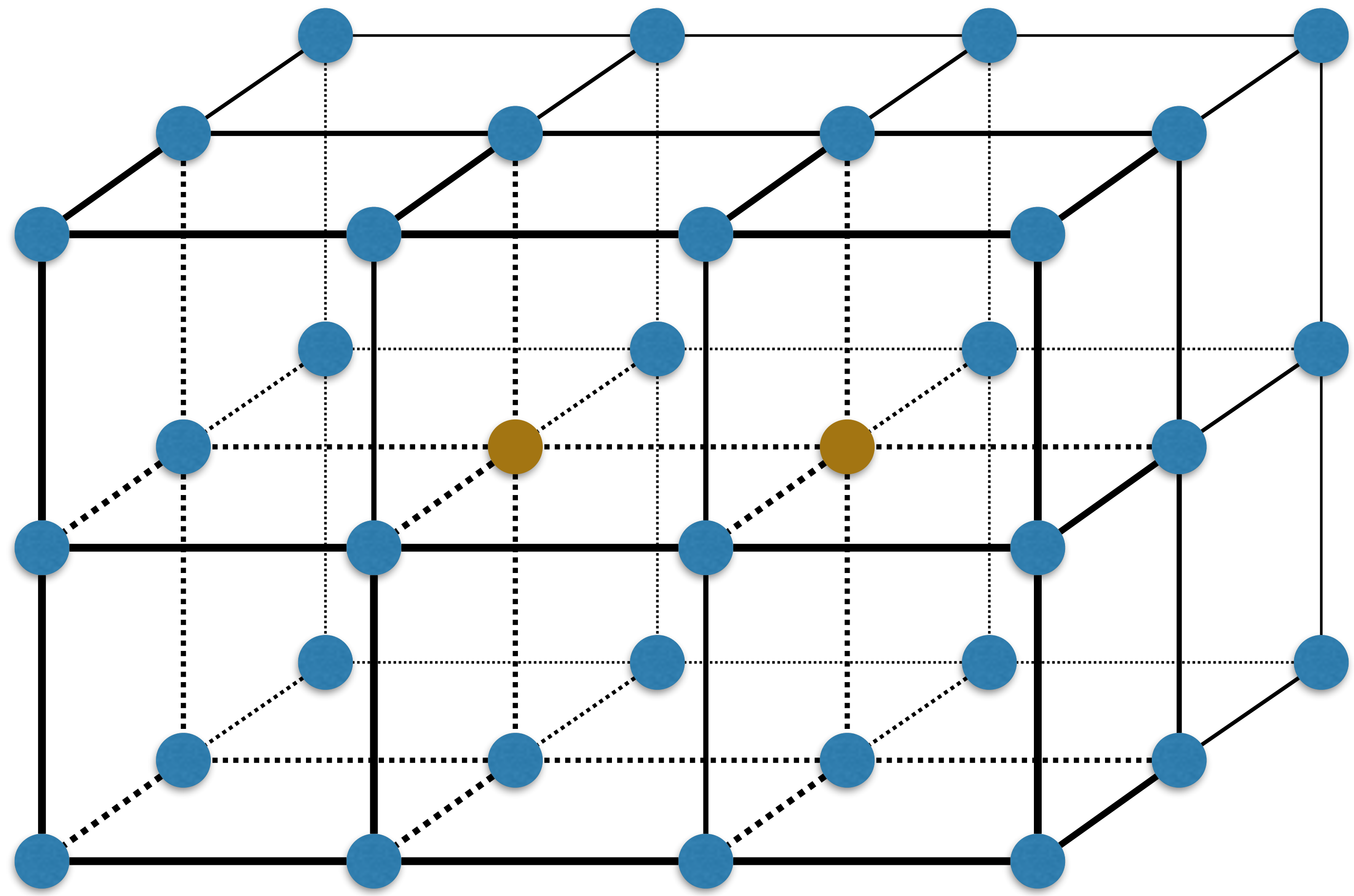
"Nonparametric"

y **X^T** **θ**

Inverted index
(transposed)

The curse of dimensionality

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



- **p ≫ 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 ("deep learning", "word embeddings"), …
- Feature **selection** (elimination): LASSO (regularization), SVM (support vectors), Bayesian nets (structure learning), locality sensitivity hashing, random projections, …

Ambiguity

Part-of-Speech tagging

The robot **wheels** out the iron.

Paraphrasing

Unemployment is on the rise.

vs

The economy is slumping.

Anaphora resolution

Carl and Bob were fighting:

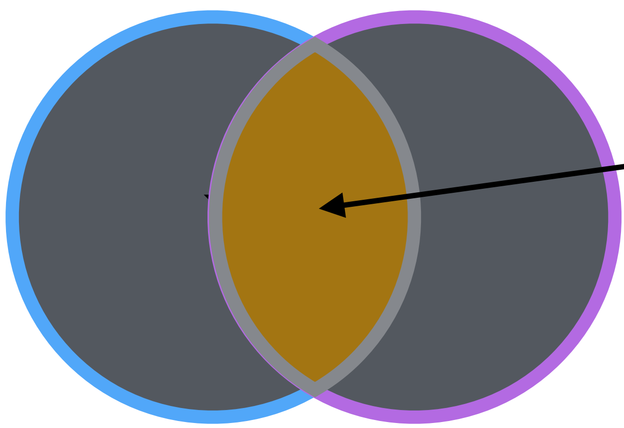
"**You** should shut up,"

Carl told **him**.

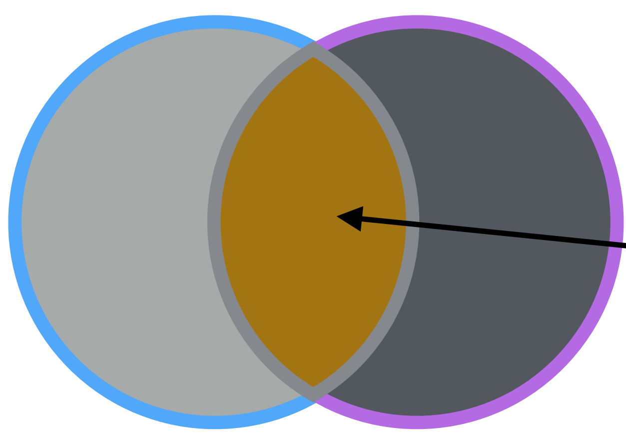
Named entity recognition

Is **Paris** really good for you?

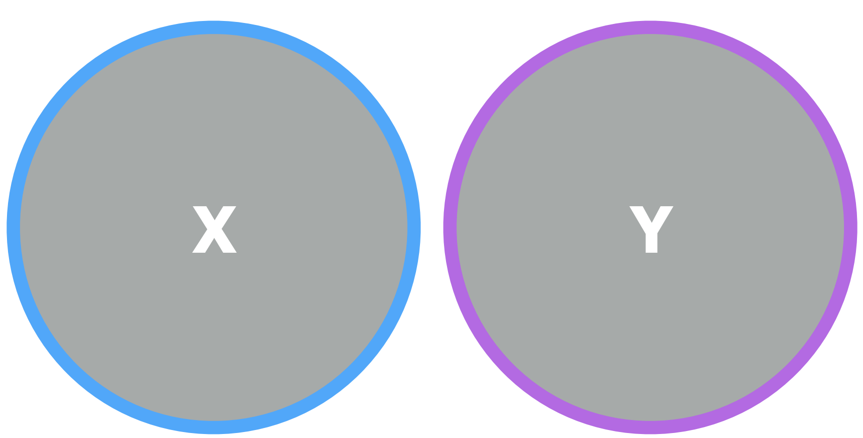
The conditional probability for dependent events



1/3



1/2



1

and

Joint Probability

P(X ∩ Y) = **P(X, Y)** = P(X × Y)

The **multiplication principle** for <u>dependent</u>* events:

P(X ∩ Y) = P(Y) × P(X | Y)

therefore, by using a little algebra:

Conditional Probability

P(X | Y) = P(X ∩ Y) ÷ P(Y)

given

*Independence

P(X ∩ Y) = P(X) × P(Y)

P(X | Y) = P(X)

P(Y | X) = P(Y)

Marginal, conditional and joint probabilities

variable/factor margin

contingency table

variable/factor

	$X=x_1$	$X=x_2$	M_Y
$Y=y_1$	$a/n = P(x_1,y_1)$	$b/n = P(x_2,y_1)$	$(a+b)/n = P(y_1)$
$Y=y_2$	$c/n = P(x_1,y_2)$	$d/n = P(x_2,y_2)$	$(c+d)/n = P(y_2)$
M_X	$(a+c)/n = P(x_1)$	$(b+d)/n = P(x_2)$	$\sum / n = 1 = P(X) = P(Y)$

Joint Probability*

$$P(x_i, y_j) = P(x_i) \times P(y_j)$$

Marginal Probability

$$P(y_i)$$

Conditional Probability

$$P(x_i \mid y_j) = P(x_i, y_j) \div P(y_j)$$

*for **independent** events

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., neighboring 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

Bayes' rule: Diachronic interpretation

prior/belief   likelihood

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

"normalizing constant"
(law of total probability)

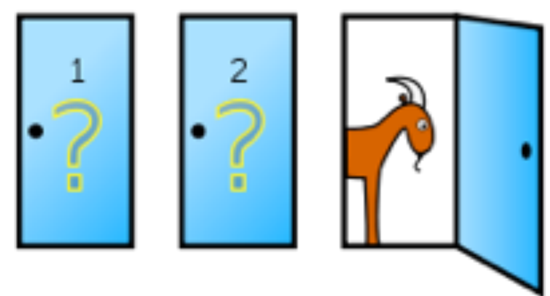
H - Hypothesis x

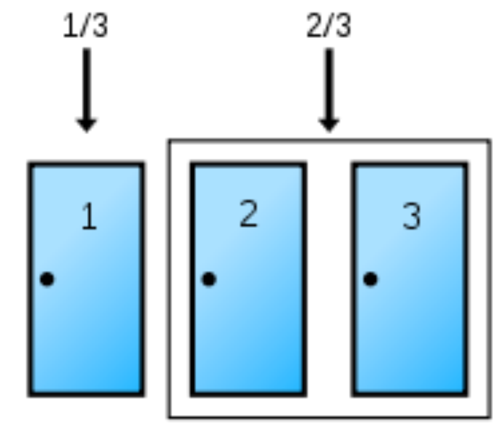
D - Data y

Bayes' rule: The Monty Hall problem

Images Source: Wikipedia, Monty Hall Problem, Cepheus

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



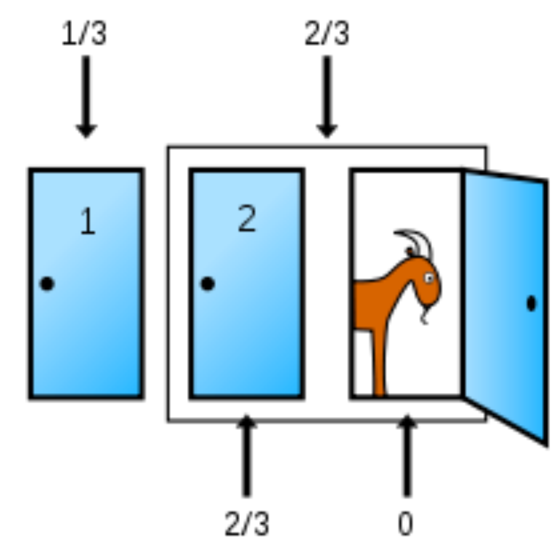


your pick

Monty Hall

	1	2	3	
Prior p(H)	1/3	1/3	1/3	
Likelihood p(D|H)	1/2	1	0	**p(D) =Σ**
p(H)* p(D|H)	1/3×1/2 =1/6	1/3×1 =1/3	1/3×0 =0	1/6+1/3 =1/2
Posterior p(H|D)	1/6÷1/2 =1/3	1/3÷1/2 =2/3	0÷1/2 =0	

H=2 D=3
given the car is behind H=1, Monty Hall opens D=(2 or 3)
H=3 D=3



similar case: a trickster hiding a stone in one of three cups...

The chain rule of conditional probability

P(A,B,C,D) = P(A) × P(B|A) × P(C|A,B) × P(D|A,B,C)

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

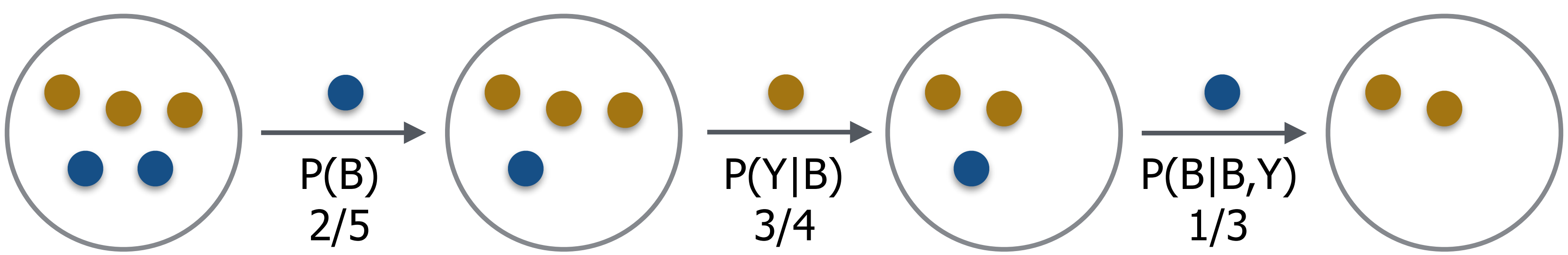
joint

conditional

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

...to w_{i-1}

from w_1...



NB: without replacement!

P(B,Y,B) = P(B) × P(Y|B) × P(B|B,Y)

1/10 = 2/5 × 3/4 × 1/3

NB: the ∑ of all possible trigram combinations will be 1!

Zipf's law: Pareto distributions

E = Expected Value → the "true" mean → dim. reduction

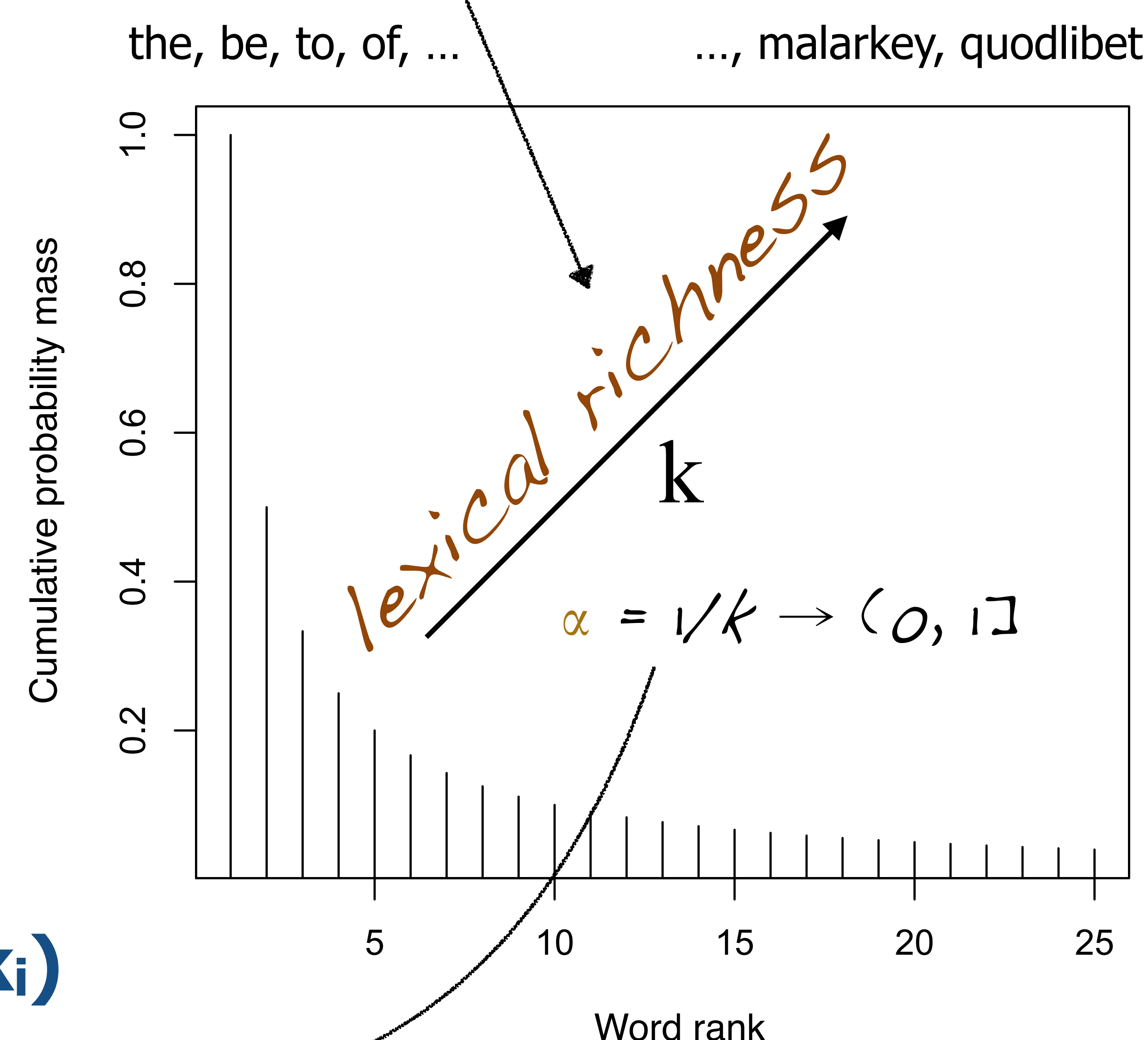
Word frequency is inversely proportional to its rank (ordered by counts)

$$f \propto 1 \div r$$
$$k = E[f \times r]$$

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"

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

The Parts of Speech (PoS)

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

" PoS tags "

- 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

The Parts of Speech (2/2)

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.)

Linguistic morphology

➡ token normalization

- [Verbal] **Inflections**
  - conjugation (Indo-European languages)
  - tense ("availability" and use varies across languages)
  - modality (subjunctive, conditional, imperative)
  - voice (active/passive)
  - …

- **Contractions**

  not a contraction: possessive s

  - don't say you're in a man**'s** world…

- **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 and PoS
  - **LemmaGen**; **morpha**; **BioLemmatizer**; **geniatagger**

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 (relevant) token

B-I-O

chunk encoding

common alternatives:

I-O

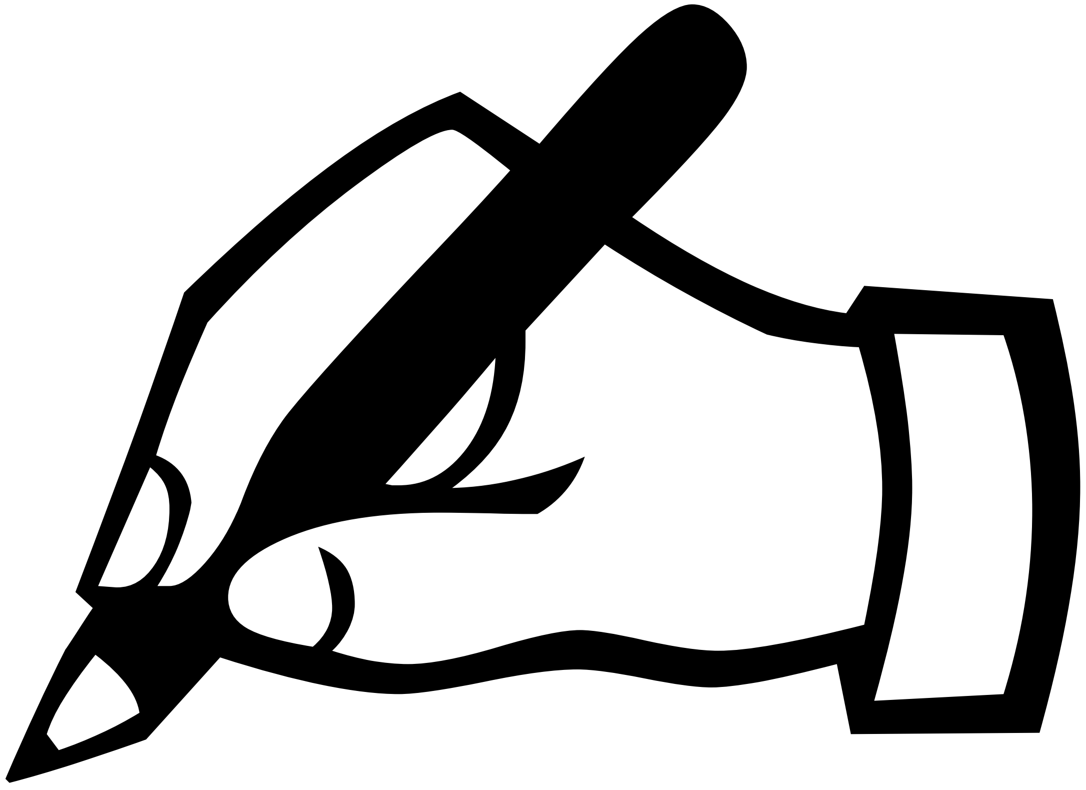
I-E-O

B-I-E-W-O

End token

(unigram) Word

Practical: Snowball stemmer



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 [PP: "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) 1/2

- Unsupervised sentence boundary detection
- $P(\bullet|\mathbf{w}_{-1}) > \mathbf{c}_{cpc}$

Dr.

  - Determines if a marker • is used as an **abbreviation** marker by comparing the **conditional probability** that the word $\mathbf{w}_{-1}$ before • is followed by the marker against some (high) cutoff probability.
    - $P(\bullet|\mathbf{w}_{-1}) = P(\mathbf{w}_{-1}, \bullet) \div P(\mathbf{w}_{-1})$
    - K&S set $\mathbf{c} = 0.99$
- $P(\mathbf{w}_{+1}|\mathbf{w}_{-1}) > P(\mathbf{w}_{+1})$

Mrs. Watson

  - Evaluates the likelihood that $\mathbf{w}_{-1}$ and $\mathbf{w}_{+1}$ 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.
- $F_{length}(\mathbf{w}) \times F_{markers}(\mathbf{w}) \times F_{penalty}(\mathbf{w}) \geq \mathbf{c}_{abbr}$

U.S.A.

  - Evaluates if any of $\mathbf{w}$'s morphology (length of $\mathbf{w}$ w/o marker characters, number of periods inside $\mathbf{w}$ (e.g., ["U.S.A"]), penalized when $\mathbf{w}$ is not followed by a •) makes it more likely that $\mathbf{w}$ is an abbreviation against some (low) cutoff.
- $F_{ortho}(\mathbf{w})$; $P_{sstarter}(\mathbf{w}_{+1}|\bullet)$; …

. Therefore

  - Orthography: lower-, upper-case or capitalized word after a probable • or not
  - Sentence Starter: Probability that $\mathbf{w}$ is found after a •

Punkt Sentence Tokenizer (PST) 2/2

- 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

An overview of free open source NLP frameworks

- **Stanford NLP Framework**
  - CoreNLP - Java
- Apache OpenNLP Framework
  - OpenNLP (& OpenNER) - Java
- General Architecture for Text Engineering
  - GATE - Java
- Unstructured Information Management Architecture
  - UIMA - Java
- **Natural Language ToolKit**
  - NLTK - Python
- FreeLing NLP Suite
  - FreeLing - C++
- The Lemur Toolkit
  - Lemur - C++
- Factorie Toolkit
  - Factorie - Scala
- The Bow Toolkit
  - Bow - C (language modeling)

Practical: Sentence segmentation