Generative Language Modelling

Thiemo Fetzer

University of Warwick & University of Bonn & CEPR & LSE & AEAI

April 22, 2025

Plan

Information overflow

Sparsity as feature of natural language

N-Gram Language Model

Excursion: Part of Speech Tagging

The Speed of Language: 39 Bits per Second

Italians speak fast—up to **9 syllables/sec**—while Germans average **5–6 syllables/sec**. Yet both transmit about the same **information rate**: **39 bits/sec**.

What Does That Add Up To?

- 39 bits/sec $\times$ 86,400 sec/day = 3,369,600 bits/day
- $\approx$ 421,200 bytes $\Rightarrow$ **0.40 MB/day**
- Assuming 6 bytes/word: $\approx$ 70,200 words/day
- At 500 words/page: $\approx$ **140 pages of ASCII English text**

> *"Languages differ in how they encode information, but they converge in how much they communicate per second."* — *Based on a cross-linguistic study*

`https://www.science.org/content/article/human-speech-may-have-universal-transmission-rate-39-bits-se`

Plan

Information overflow

Sparsity as feature of natural language

N-Gram Language Model

Excursion: Part of Speech Tagging

Fundamentals about text data

There are very few "fundamental law's" in computational linguistic. The exception are *Heap's Law* and *Zipf's Law*, which highlights why most text data is *sparse*.
Typically we will define a model of language that is a *stochastic* process.

- Study the single occurence of a word, not its frequency - *Bernoulli process*
- Modeling word frequencies: *Poisson* or *multionomial* distribution.

Heap's Law

Heaps' law (also called Herdan's law) is an empirical relationship which describes the number of *distinct words* in a document (or set of documents) as a function of the *document length* (so called type-token relation). It can be formulated as

$$|V| = kN^{\beta}$$

In log-log form, this power law becomes a straight line

$$\log(|V|) = k + \beta \log(N)$$

where $|V|$ is the size of the vocabulary (the number of types) and N is the number of tokens.

Illustration of Heap's Law in State of Union speeches

```
library(quanteda)
library(data.table)
data(SOTUCorpus, package = "quantedaData")
DF <- summary(SOTUCorpus)
plot(log(DF$Types), log(DF$Tokens)) + abline(lm(log(DF$Tokens) ~ log(DF$Types)))
```

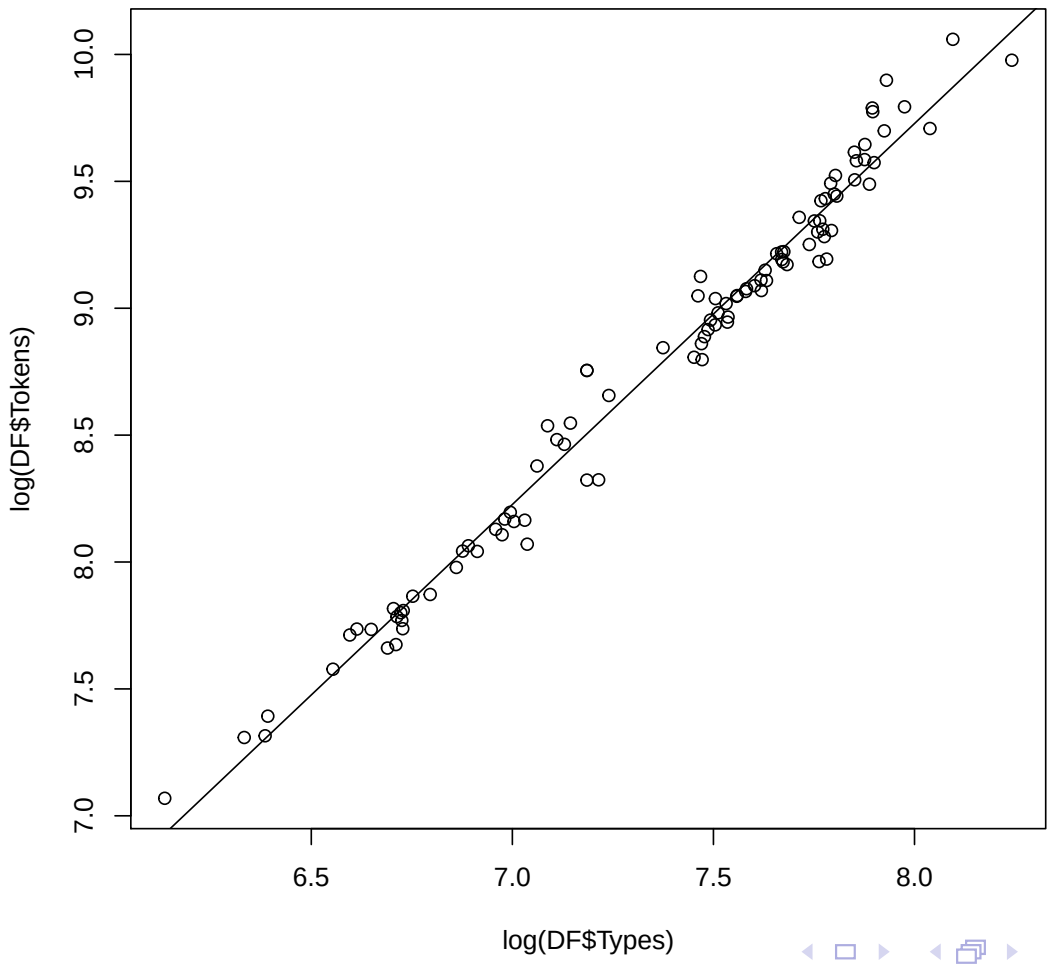


Illustration of Heap's Law in State of Union speeches

```
summary(lm(log(DF$Tokens) ~ log(DF$Types)))

##
## Call:
## lm(formula = log(DF$Tokens) ~ log(DF$Types))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.2245 -0.0664 -0.0120  0.0603  0.2751
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -2.2803     0.1569   -14.5   <2e-16 ***
## log(DF$Types)   1.5011     0.0212    70.7   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1 on 98 degrees of freedom
## Multiple R-squared:  0.981, Adjusted R-squared:  0.981
## F-statistic: 5e+03 on 1 and 98 DF,  p-value: <2e-16
```

For larger corpora, the coefficient is typically smaller. Stemming and further tokenization typically lowers the vocabulary space.

Zipf's Law

Zipf's Law is a law about the frequency distribution of words *within a document*.
Zipf's Law states that s frequency of any word is inversely proportional to its rank in the frequency table.
Formally: Word frequency

$$f = \frac{a}{r^b}$$

where r is the rank in the (empirical) word frequency distribution.
Again, logging

$$\log(f) = \log(a) - b\log(r)$$

Illustration of Zipf's Law

```
OBAMA <- subset(SOTUCorpus, filename == "su2012.txt")
TOK <- tokenize(OBAMA, removePunct = TRUE)
TOK <- data.table(token = tolower(unlist(TOK)))
TOK <- TOK[, .N, by = token][order(N, decreasing = TRUE)]
TOK[1:20]
```

```
##     token   N
##  1:   the 294
##  2:    to 230
##  3:   and 204
##  4:    of 170
##  5:     a 160
##  6:  that 144
##  7:    in 108
##  8:   our  84
##  9:    we  84
## 10:   for  63
## 11:    is  59
## 12:  will  57
## 13:  this  52
## 14:    on  51
## 15:     i  50
## 16:    it  48
## 17:  with  47
## 18:  from  47
## 19:  more  43
## 20:    as  39
```

```
TOK[, `:=`(rank, 1:nrow(TOK))]
```

Illustration of Zipf's Law

```
plot(log(TOK$N), log(TOK$rank)) + abline(lm(log(TOK$N) ~ log(TOK$rank)))

## numeric(0)
```

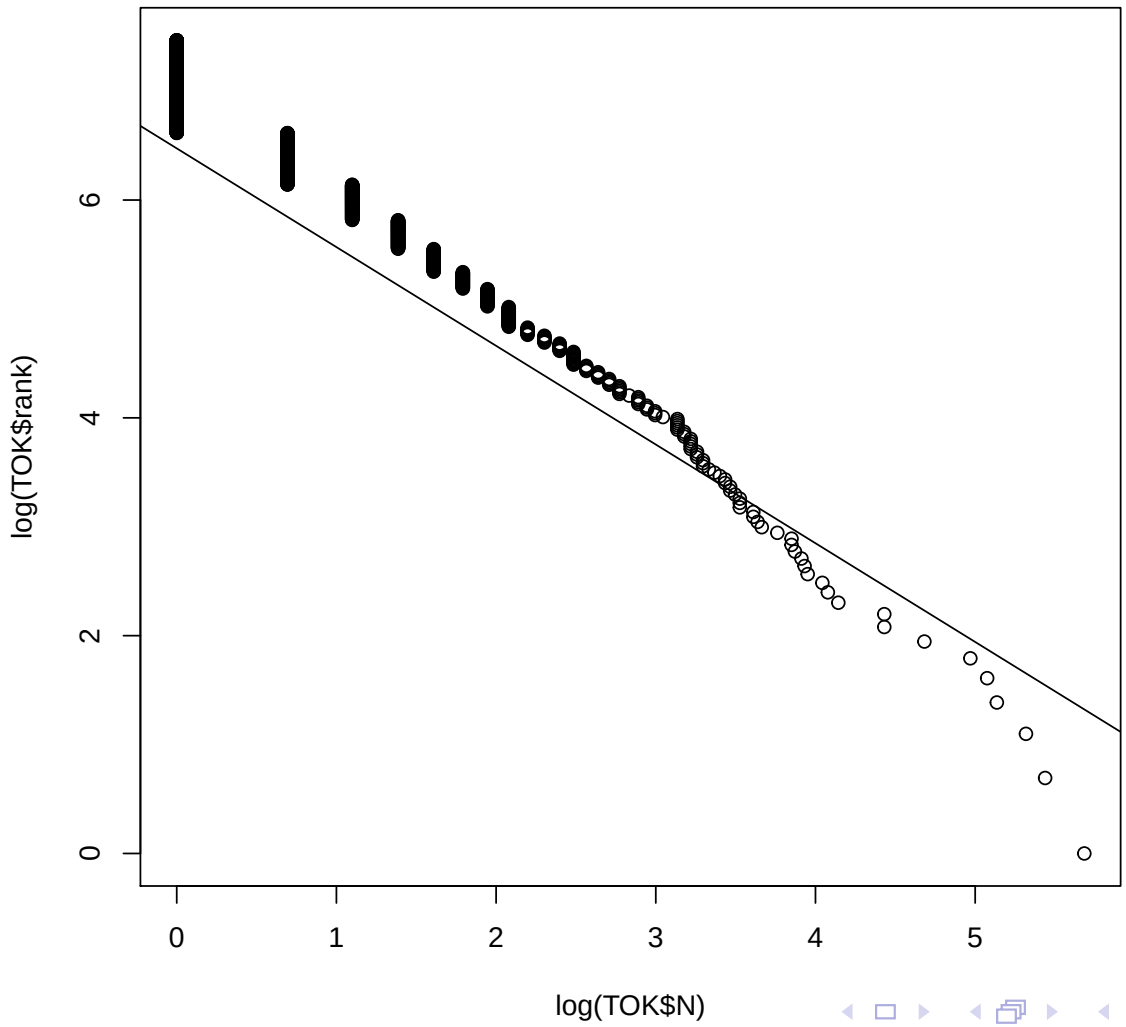


Illustration of Zipf's Law

```
summary(lm(log(TOK$N) ~ log(TOK$rank)))

##
## Call:
## lm(formula = log(TOK$N) ~ log(TOK$rank))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.7907 -0.1110  0.0411  0.1406  0.2938
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)    6.47428    0.02937     220   <2e-16 ***
## log(TOK$rank) -0.90642    0.00449    -202   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.186 on 1747 degrees of freedom
## Multiple R-squared:  0.959, Adjusted R-squared:  0.959
## F-statistic: 4.08e+04 on 1 and 1747 DF,  p-value: <2e-16
```

Implications of Heap's and Zipf's Law

- *Heap's Law* and *Zipf's Law* imply that data matrices constructed from text data is very *sparse*.
- Sparsity implies that there would be many zeroes.
- Most data processing steps for text data involve *densifying* the word frequency distribution.
- We next discuss a range of steps commonly used to densify.

Word Tokenization and Normalization

- **Tokenization** - task of segmenting running text into words.
  - Plain vanilla approaches would just `str_split(text," ")` - splitting by white spaces.
  - More sophisticated methods apply *locale* (language) specific algorithms.
- **Normalization**- task of putting words/tokens into a standardized format.
  - For example `we're` to `we are`.
  - Casefolding of tokens (lower-case or upper case)

Plan

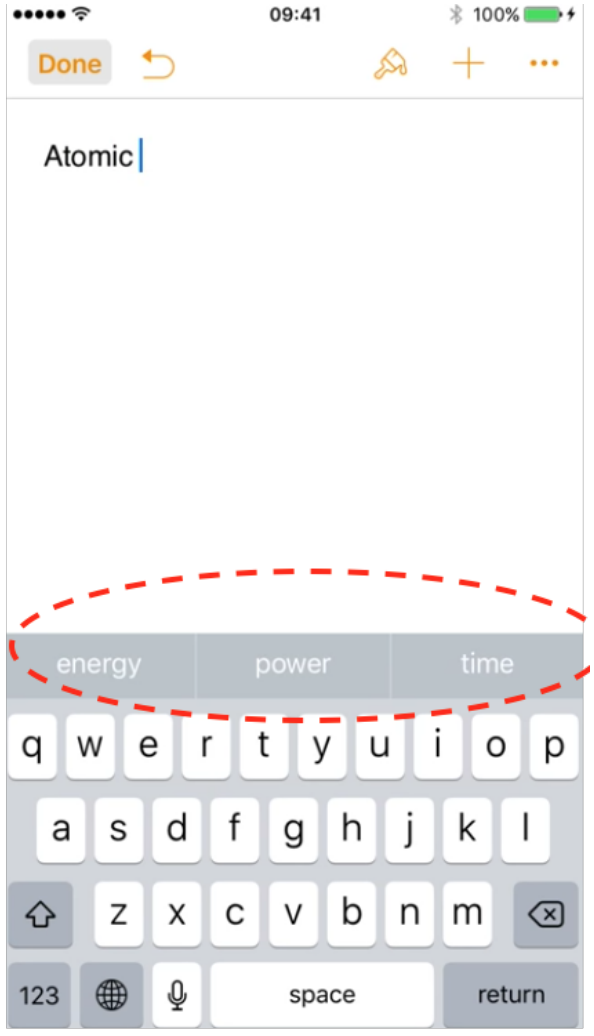
Information overflow

Sparsity as feature of natural language

N-Gram Language Model

Excursion: Part of Speech Tagging

Autocomplete Function



Lets have a look at how an n-gram language model.

Autocomplete Function

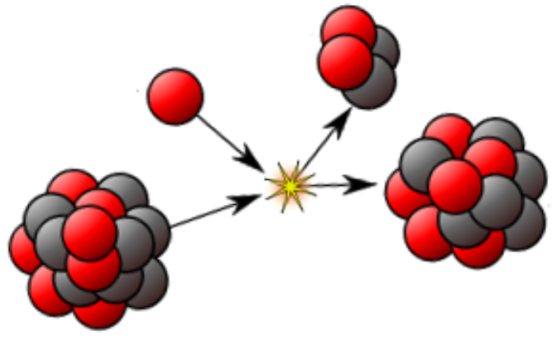
Atomic Energy will have been made available to a single source

Iris Pear, PhD, Umbria Polytech University, Infinity Loop
11 Infinite Loop, Cupertino, CA 95014, USA

Abstract

Atomic Physics and I shall not have the same problem with a separate section for a very long long way. Nuclear weapons will not have to come out the same day after a long time of the year he added the two sides will have the two leaders to take the same way to bring up to their long ways of the same as they will have been a good place for a good time at home the united front and she is a great place for a good time. The atoms of a better universe will have the right for the same as you are the way we shall have to be a great place for a great time to enjoy the day you are a wonderful person to your great time to take the fun and take a great time and enjoy the great day you will be a wonderful time for your parents and kids. Molecular diagnostics will have been available for the rest by a single day and a good day to the rest have a wonderful time and aggravation for the rest day at home time for the two of us will have a great place for the rest to be great for you tomorrow and tomorrow after all and I am a very happy boy to the great day and I hope he is wonderful.

Image



Lets have a look at how an n-gram language model.

Uses of Probabilistic Language Models

- Spelling correction
- Auto complete
- Language detection (classification)
- Other classification tasks

Probabilistic Language Models

What is the likely next word?

```
Make America ...
```

N-gram language models see sentences as sequences of words, the occurrence of each word is a function of the likelihood of the sequence of words.

```
Make America Great Again
```

The predicted next word naturally depends on the corpus on which a language model was trained on and a range of other factors. Lets formalize things a bit.

Probabilistic Language Models

What is the probability of:

$$P(\text{again}|\text{Make America great})$$

We can express this probability as:

$$P(\text{again}|\text{Make America great}) = \frac{P(\text{Make America great again})}{P(\text{Make America great})}$$

which we may be inclined to estimate as

$$\hat{P}(\text{again}|\text{Make America great}) = \frac{C(\text{Make America great again})}{C(\text{Make America great})}$$

where the $C(.)$ indicates the raw counts of the text fragments.

Make America ...

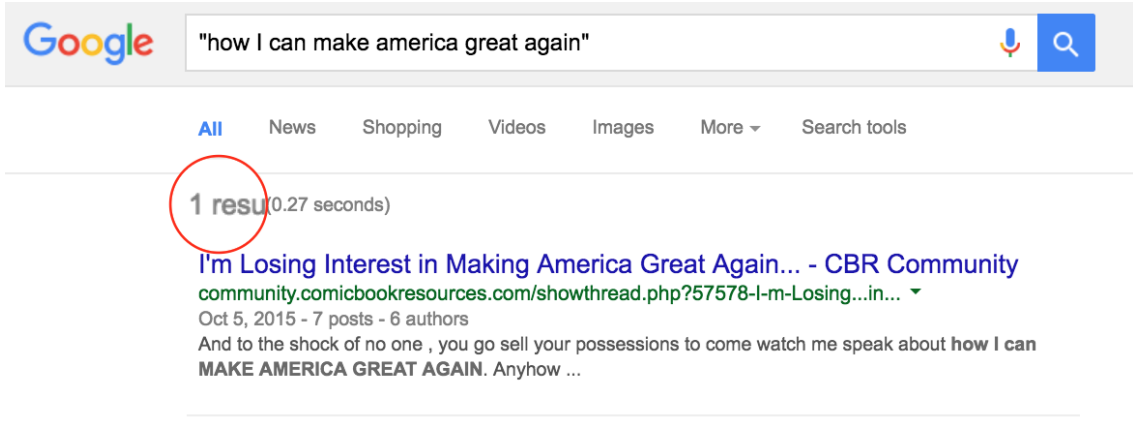
```
library(ngram)
TRUMP <- readLines(con = "../../Data/Trump-Speeches.txt")
# concatenate into one massive string, remove empty lines
TRUMP <- TRUMP[-grep("^SPEECH|^$", TRUMP)]
TRUMP <- gsub(" +", " ", TRUMP)
TRUMP <- paste(TRUMP, collapse = " ")
TRUMP <- preprocess(TRUMP, remove.punct = TRUE)

p1 <- length(strsplit(TRUMP, "make america great again")[[1]])
p1

## [1] 45

p2 <- length(strsplit(TRUMP, "make america great")[[1]])
p2

## [1] 48

p1/p2

## [1] 0.938
```

So here estimating the conditional probability is possible as the sentence is rather short. However, for longer sentences this becomes much more difficult.

Longer Sentences

For longer sentences, it becomes much less likely that we will observe sufficient number of raw counts.

`how I can make America Great again`



Meaning the estimate

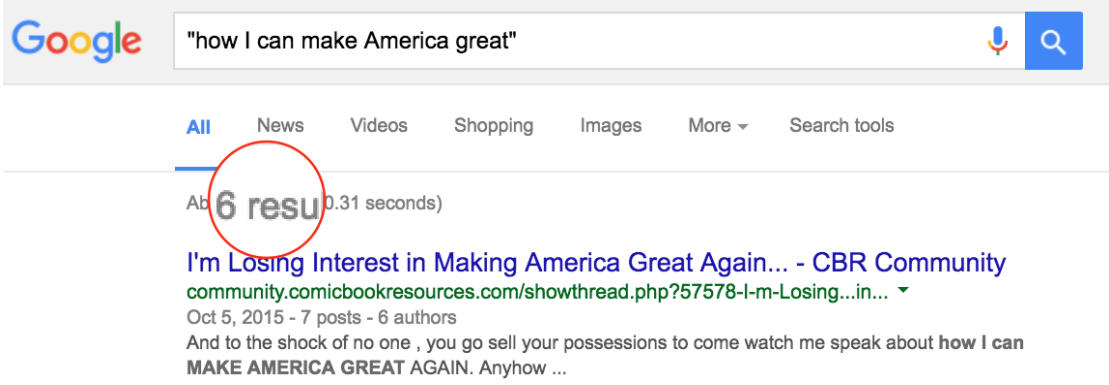
$$\hat{P}(\text{again}|\text{how I can make America great}) = \frac{C(\text{how I can make America great again})}{C(\text{how I can make America great})}$$

is very imprecise. For longer sentence, the counts in numerator and denominator would be exactly zero.

Longer Sentences

For longer sentences, it becomes much less likely that we will observe sufficient number of raw counts.

`how I can make America Great again`



Meaning the estimate

$$\hat{P}(\text{again}|\text{how I can make America great}) = \frac{C(\text{how I can make America great again})}{C(\text{how I can make America great})}$$

is very imprecise. For longer sentence, the counts in numerator and denominator would be exactly zero.

A bit of notation...

What is the joint probability of observing a sequence of words $w_1, ..., w_n$?

$$
\begin{aligned}
P(w_1, .., w_n) &= P(w_1)P(w_2, ..., w_n|w_1) \\
&= P(w_1)P(w_2|w_1)P(w_3, ..., w_n|(w_1, w_2)) \\
&= P(w_1)P(w_2|w_1)P(w_3|(w_1, w_2))P(w_4, ..., w_n|(w_1, w_2, w_3)) \\
&... \\
&= \prod_{k=1}^{n} P(w_k|(w_1, ..., w_{k-1})
\end{aligned}
$$

iteratively applying the **Chain Rule of Probability**.

Curse of Dimensionality

We can compute probability of a sentence

$$P(w_1, .., w_n) \quad = \quad \prod_{k=1}^{n} P(w_k | (w_1, ..., w_{k-1})$$

by multiplying a sequence of conditional probabilities.

- We can not estimate each individual conditional probability because its highly unlikely that a stable estimate does exist.
- Similarly, it would be computationally infeasible.
- Parameter space (number of conditional probabilities that need to be estimated) grows exponentially in n.

N-gram language model

Given the computational issues, the `n-gram` statistical language model assumes that we can approximate the probability of a word w given a history h by looking back just at the last N words in the history.
The simplest case is "not to look back", i.e. approximate the probability of a word w with an empty history, $h = \emptyset$. This yields the **unigram** language model.
So we would approximate $P(w_k|(w_1, ..., w_{k-1})) \approx P(w_k)$, which yields:

$$P(w_1, .., w_n) = \prod_{k=1}^{n} P(w_k)$$

This assumes that a sequence of words can be best approximated by the unconditional probabilities of an individual word appearing or not appearing.
The factorization implies that we assume that words are *stochastically independently drawn* from one another.

An unigram babbler

Some random sequences created with a unigram Trump babbler

```
## [1] "prosperity not talking about the game and more you mean but ill tell you k
## [2] "o terminate obamacare with a lot its deadly totally destabilized the worst
## [3] "ated me so im "
## [1] "statements theyve never ever see what that stupid your life obviously you
## [2] "aid i talked about results yesterday i wrote the way i order televisions s
## [3] "somebody we order "
## [1] "jeep holding a lot of affection and the hell wants i dont think its really
## [2] "ing to even talk about the way if i dont have said recently he sent out th
```

Unigrams create poor results, because natural linguistic dependencies encoded in word collocations (e.g. verb follows an object, prepositions preceede locations, etc.) are ignored.
Adding more **context**, by looking and history of preceeding words.

N-gram models

A bigram model defined as

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

This assumption that the probability of a word depends only on the previous word is called the **Markov** assumption. Here we approximate $P(w_k|(w_1, ..., w_{k-1})) \approx P(w_k|w_{k-1})$
Markov models are a class of probabilistic models that assume that the future can be predicted without looking *too far* into the past.
We can generalize to N-gram models

$$P(w_1, ..., w_n) = \prod_{k=1}^{n} P(w_k | w_{k-1}, ..., w_{k-N+1})$$

where $P(w_k|(w_1, ..., w_{k-1})) \approx P(w_k|w_{k-1}, w_{k-2}, ...w_{k-N+1})$

A Mini Example

Suppose you have a small corpus

```
<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>
```

the <s> indicate start and end tags, they are to be treated just as words. It is important to include them to ensure that the probability estimates that we are extracting are well behaved.

$P(\text{I}|\text{<s>}) = \frac{2}{3} = .67$ $P(\text{Sam}|\text{<s>}) = \frac{1}{3} = .33$ $P(\text{am}|\text{I}) = \frac{2}{3} = .67$

$P(\text{</s>}|\text{Sam}) = \frac{1}{2} = 0.5$ $P(\text{Sam}|\text{am}) = \frac{1}{2} = .5$ $P(\text{do}|\text{I}) = \frac{1}{3} = .33$

Bigram illustration

`I want to make America great again`

$$P(\text{I}|<\text{start}>)P(\text{want}|\text{I})P(\text{to}|\text{want})P(\text{make}|\text{to})P(\text{America}|\text{make})$$
$$P(\text{great}|\text{America})P(\text{again}|\text{great})P(<\text{end}>|\text{again})$$

Assume $P(\text{I}|<\text{start}>) = 25\%$ and $P(<\text{end}>|\text{again}) = 25\%$.

Bigram illustration

```
##       ngrams estprop         ngrams  estprop
##  1:   to be   0.1108  make america  0.186782
##  2: want to   0.6382        make a  0.146552
##  3:  i dont   0.0797      make our  0.140805
##  4:   to do   0.0696       make it  0.137931
##  5: i think   0.0723 america great  0.269663
##  6:  i mean   0.0614 america first  0.117978
##  7:  i said   0.0611    america is  0.044944
##  8:  i have   0.0609 america shower 0.005618
##  9: to have   0.0403   great again  0.107715
## 10:  to get   0.0386  great people  0.045124
## 11:   i was   0.0420    great with  0.034934
## 12:  i want   0.0404     great and  0.032023
## 13: to make   0.0299      a shower  0.000279
## 14:  to the   0.0271  shower again  1.000000
```

$$P(\text{I}|<\text{start}>)P(\text{want}|\text{I})P(\text{to}|\text{want})P(\text{make}|\text{to})P(\text{America}|\text{make})$$
$$P(\text{great}|\text{America})P(\text{again}|\text{great})P(<\text{end}>|\text{again})$$
$$= 0.25x0.04x0.64x0.03x0.19x0.27x0.11x0.25 = 0.0000002708$$

$$P(\text{I}|<\text{start}>)P(\text{want}|\text{I})P(\text{to}|\text{want})P(\text{make}|\text{to})P(\text{America}|\text{make})$$
$$P(\textbf{shower}|\text{America})P(\text{again}|\textbf{shower})P(<\text{end}>|\text{again})$$
$$= 0.25x0.04x0.64x0.03x0.19x0.0056179775x1x0.25 = 0.00000005123$$

What do we learn?

- N-gram models capture syntatic features and general knowledge (here, knowledge about the underlying speeker)
- It turns out that linguistic features such as word sequences "I want" are reasonably frequent and they capture linguistic features: verbs tend to follow subject as indicated by "I".
- "america shower" is much more rare in Trump's speeches compared to "america great".
- N-gram models can be trained by counting and normalization

Illustration of estimating N-Gram probabilities.

We will use Maximum Likelihood estimation, illustrate and proof what is the maximum likelihood estimator using a unigram model

$$P(w_1, ..., w_n) = \prod_{k=1}^{n} P(w_k)$$

This can think of this as a sequence of independent *Bernoulli* trials with success probability p_k for word k.
Suppose you observe a sample of size N of word sequences of length n, $\{W^1, ..., W^N\}$. So each $W^i = (w_{i1}, ..., w_{in})$, where $w_{ij} = 1$ in case word w_j is present in sequence i.
What is the likelihood of observing a specific sequence?

$$P(W^i) = \prod_{k=1}^{n} p_k^{w_{ik}} (1 - p_k)^{1 - w_{ik}}$$

What is the likelihood of observing the whole sample?

$$\prod_{i=1}^{N} P(W^i) = \prod_{i=1}^{N} \prod_{k=1}^{n} p_k^{w_{ik}} (1 - p_k)^{1 - w_{ik}}$$

Log Likelihood

Taking logs

$$\sum_{i=1}^{N}\sum_{k=1}^{n} w_{ik} log(p_k) + (1 - w_{ik}) log(1 - p_k)$$

We want to find optimal $p_1, ..., p_n$, so take FOC. Notice that everything is additive and there are no interactions between individual p_k. Take FOC with respect to p_k.

$$\sum_{i=1}^{N} \frac{1}{p_k} w_{ik} - \sum_{i=1}^{N} (1 - w_{ik}) \frac{1}{1 - p_k} = 0$$

This yields

$$p_k = \frac{\sum_{i=1}^{N} w_{ik}}{N} \quad \forall k$$

where the numerator is just the number of word sequences that contain the word and N is just the sample size.

In case of Bigram model

For Bigram model, the intuitive way to estimate the probability $P(w_k|w_{k-1})$ is to get the counts of word sequences $C(w_{k-1}, w_k)$ from a corpus and normalize this by the counts of word that share the same first word, i.e. we estimate

$$P(w_k|w_{k-1}) = \frac{C(w_{k-1}, w_k)}{\sum_w C(w_{k-1}, w)} = \frac{C(w_{k-1}, w_k)}{C(w_{k-1})}$$

Note that $\sum_w C(w_{k-1}, w) = C(w_{k-1})$
I.e. the number of word pairs that share the starting word w_{k-1} should simply add up to the number of times the word w_{k-1} appears.

A Bigram Example

I want to make America great again

```
##           i want  to make america great again
## i         2  196   0    3       0     0     0
## want      5    0 485    0       0     0     0
## to        5    4   0  163       0     1     0
## make      0    0   0    0      65    17     0
## america   1    0   4    0       0    48     1
## great    16    0   7    0       0    14    74
## again    19    0   3    0       1     0     0
```

Normalization: divide each row's counts by appropriate unigram counts for w_{n-1}

```
##    [,1]   [,2]   [,3]   [,4]     [,5]   [,6]       [,7]
## X1 "to"   "want" "i"    "great" "make" "america" "again"
## V1 "5460" " 760" "4857" " 687"  " 348" " 178"    " 246"
```

A Bigram Example

```
##                 i     want     to     make america    great   again
## i       0.000412 0.040354 0.0000 0.000618 0.00000 0.000000 0.00000
## want    0.006579 0.000000 0.6382 0.000000 0.00000 0.000000 0.00000
## to      0.000916 0.000733 0.0000 0.029853 0.00000 0.000183 0.00000
## make    0.000000 0.000000 0.0000 0.000000 0.18678 0.048851 0.00000
## america 0.005618 0.000000 0.0225 0.000000 0.00000 0.269663 0.00562
## great   0.023290 0.000000 0.0102 0.000000 0.00000 0.020378 0.10771
## again   0.077236 0.000000 0.0122 0.000000 0.00407 0.000000 0.00000
```

- Ratio of $\frac{C(w_n, w_{n-1})}{C(w_{n-1})}$ is a maximum Likelihood estimate for $P(w_n|w_{n-1})$
- We observe that many raw counts are zero: the matrix is sparse.
- The larger N, the more sparse will these matrices get.
- However, larger N generally results in better performance as more history is incorporated.

Trade-off: Higher order N-gram versus lower order N-grams

Unigram Trump Babbler

```
## [1] "prosperity not talking about the game and more you mean but ill tell you know shes going t"
## [2] "o terminate obamacare with a lot its deadly totally destabilized the worst human being tre"
## [3] "ated me so im not smart and tell you get on trade agreement you look we have to end "
```

Bigram Trump Babbler

```
## [1] "from pakistan and he was talking to some place it costs 3 billion and i said ok then i hav"
## [2] "e to happen with these two nations and must regard them with their families we mourn as on"
## [3] "e united people with force purpose and determination but the muslims living in this "
```

Trigram Trump Babbler

```
## [1] "permanently admits more than 100000 immigrants from the middle east our government has bee"
## [2] "n admitting ever growing numbers year after year without any effective plan for our own se"
## [3] "curity in fact clintons state department was in charge of admissions and the admissions pr"
## [4] "ocess for people applying to enter from overseas "
```

Quadrigram Trump Babbler

```
## [1] "tough as nails hes going to be your champion im going to be good for womens health issues "
## [2] "its very important to me it was instructional when i did the art of the deal pac after the"
## [3] " book they have all these pacs and the money comes in and its "
```

as N increases, the number of parameters to be estimated explodes. In addition, as we have seen, the matrices are very sparse - many zeroes!

Curse of Dimensionality of N-gram model

Suppose you have a vocabulary of size $|V|$. Assuming no constraints imposed by language structure. How many different conditional probabilites are there to estimate?

- There are $|V|$ sentences, containing exactly 1 words.
- There are $|V| \times |V|$ sentences containing 2 words
- There are $|V| \times |V| \times |V|$ sentences containing 3 words

In total there are $|V|^N$ parameters in an n-gram for vocabulary size $|V|$.

Babbling Donald Trump

```
library(ngram)
TRUMP <- readLines(con = "../../Data/Trump-Speeches.txt")
# concatenate into one massive string, remove empty lines
TRUMP <- TRUMP[-grep("^SPEECH|^$", TRUMP)]
TRUMP <- gsub(" +", " ", TRUMP)
TRUMP <- paste(TRUMP, collapse = " ")
TRUMP <- ngram(TRUMP, n = 3)
str_break(babble(TRUMP, genlen = 35, seed = 130))
```

```
## [1] "I have as big a heart as anybody. We want to win Iowa, folks. Because look, I love the poo"
## [2] "rly educated. We're the smartest people, we're the most loyal people by far. Everybody say"
## [3] "s it. "
```

Making N-gram Models Better

Basic N-gram models can be greatly improved using **smoothing** and **backoff** techniques to deal with data sparsity and unseen sequences. Smoothing Techniques adjust probability estimates to avoid assigning zero probability to unseen N-grams.

- **Additive (Laplace) Smoothing**:

$$P_{\text{laplace}}(w_k \mid w_{k-1}) = \frac{C(w_{k-1}, w_k) + \alpha}{C(w_{k-1}) + \alpha \cdot |V|}$$

  where α is typically set to 1.

- **Kneser-Ney Smoothing**: A more advanced method that considers not just frequency but the diversity of contexts a word appears in.

Making N-gram Models Better

Backoff Models

If a trigram is unseen, back off to a bigram; if that's unseen, back off to a unigram.

Interpolation

Combine N-gram probabilities across different orders:

$$P(w_k \mid w_{k-1}, w_{k-2}) = \lambda_3 P_3 + \lambda_2 P_2 + \lambda_1 P_1$$

where λ_i are weights that sum to 1.

Why it matters

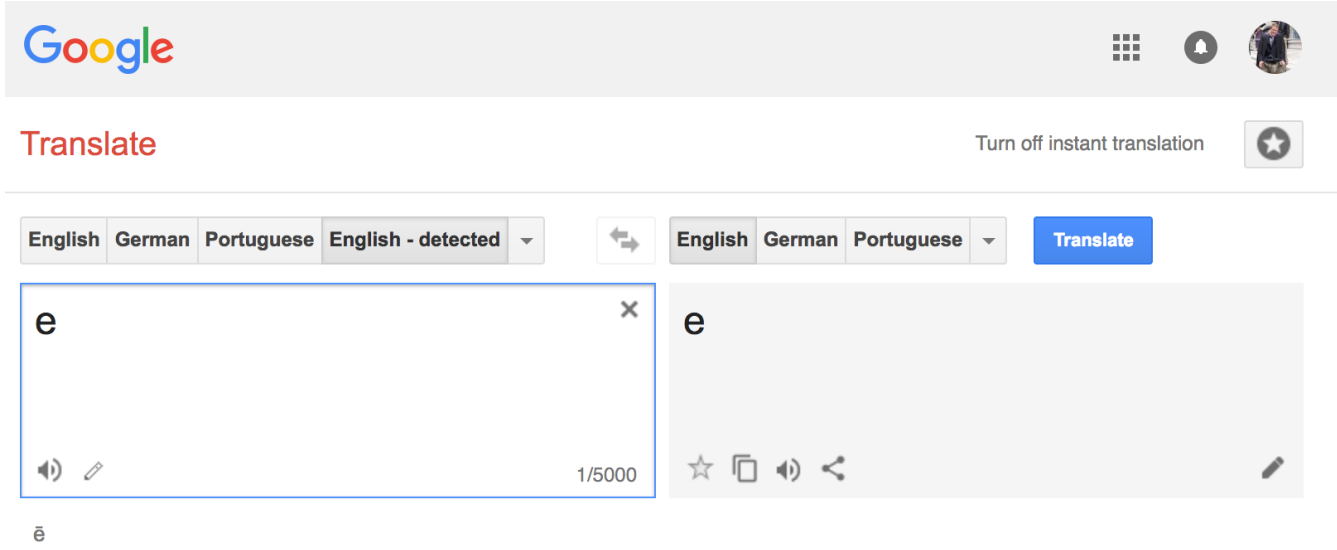
These enhancements let N-gram models perform much better, especially on sparse data or rare sequences.

Issues with N-gram based language models

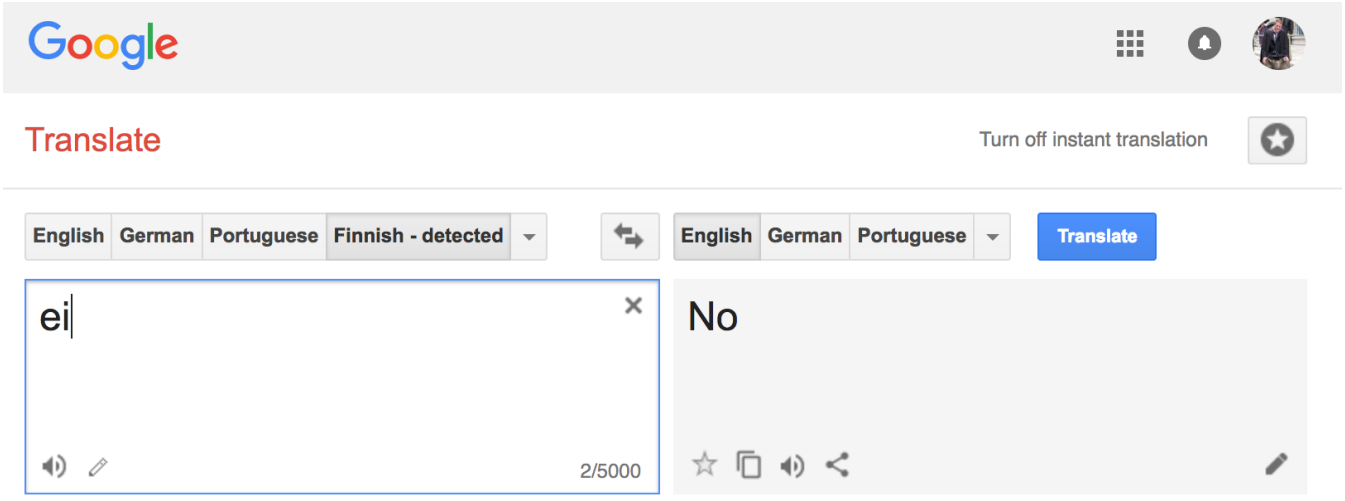
- Sparsity: Even big corpora miss many combinations.
- No generalization: “he walks quickly” “he runs fast”
- No notion of meaning or similarity between words.

“What if we could teach the model that ‘walk’ and ‘run’ are related?”

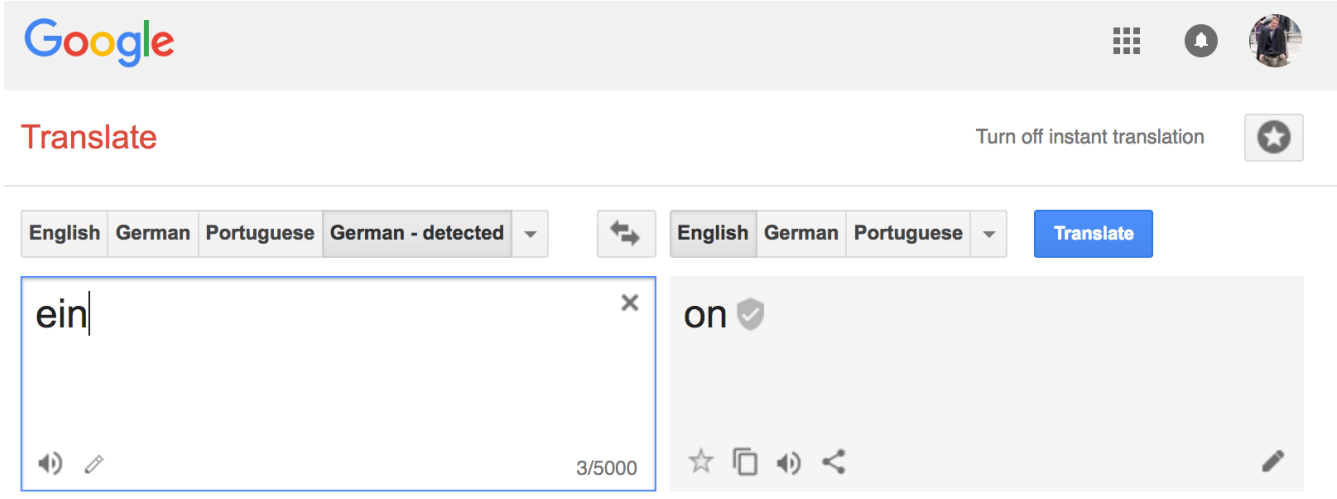
N-gram based language categorization



N-gram based language categorization



N-gram based language categorization



N-gram based language categorization

- The simplest n-grams are single- and multi character n-grams.
- For example the word "TEXT" can be tokenized into the following
  - bi-grams: _T, TE, EX, XT, T_
  - tri-grams: _TE, TEX, EXT, XT_, T_ _
  - quad-grams: _TEX, TEXT, EXT_, XT_ _, T_ _ _
- Often times do different human languages or documents covering different (technical) topics exhibit different *ngram frequency profiles*.
- Remember Zipf's law, which stated that the
  The nth most common word in a human language text occurs with a frequency inversely proportional to n.

$\Rightarrow$ can use n-gram frequency profiles across different human languages to build a simple classifier of human language

N-gram based language categorization

```
library(textcat)
## ngram top character profiles for english
TC_char_profiles[["english"]][1:20]

##     _     e     t     o     n     i     a     s     r     h    e_     d    _t     c     l
## 20326  6617  4843  3834  3653  3602  3433  2945  2921  2507  2000  1816  1785  1639  1635
##    th    he   _th     u     f
##  1535  1351  1333  1309  1253

## ngram top character profiles for english
TC_char_profiles[["german"]][1:20]

##     _     e     n     i     r     t     s     a     h     d    er    en     u     l    n_
## 31586 15008  9058  7299  6830  5662  5348  4618  4176  4011  3415  3412  3341  3266  2848
##     c    ch     g     o    e_
##  2636  2460  2407  2376  2208

## ngram top character profiles for english
TC_char_profiles[["spanish"]][1:20]

##     _     e     a     o     s     n     i     r     l     d     c     t     u    a_    e_
## 25044  7830  7437  5102  4394  4358  4065  3998  3634  3118  2931  2834  2316  2269  2211
##    s_    de     p    _d     m
##  1862  1679  1673  1644  1447
```

Cavnar, W. B., Trenkle, J. M., & Mi, A. A. (1994). N-Gram-Based Text Categorization. In Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval, 161–175.
implemented in the **textcat** package in R.

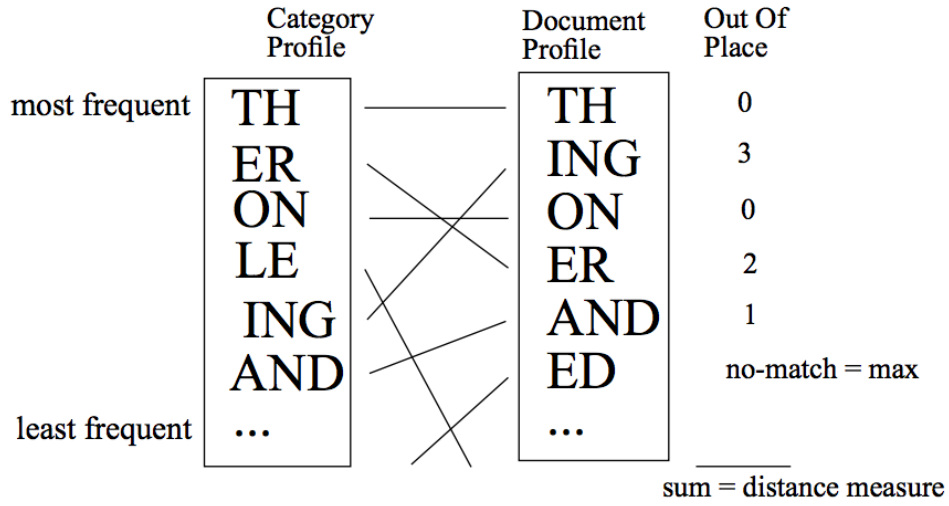
N-gram based language categorization

```
library(textcat)

textcat("Let see whether we can confuse it with fake latin")

## [1] "english"

textcat("Lorem ipsum dolor sit amet, et dapibus nunc sit gravida, augue vitae, felis massa est augue vehicula"

## [1] "latin"
```

Cavnar, W. B., Trenkle, J. M., & Mi, A. A. (1994). N-Gram-Based Text Categorization. In Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval, 161–175.
implemented in the **textcat** package in R.

N-gram based language categorization

Computation of distances using an "out-of-place" statistic - alternatives for rank-ordered



Note: These profiles are for explanatory purposes only and do not reflect real N-gram frequency statistics.

Plan

Information overflow

Sparsity as feature of natural language

N-Gram Language Model

Excursion: Part of Speech Tagging

Part of Speech Tagging

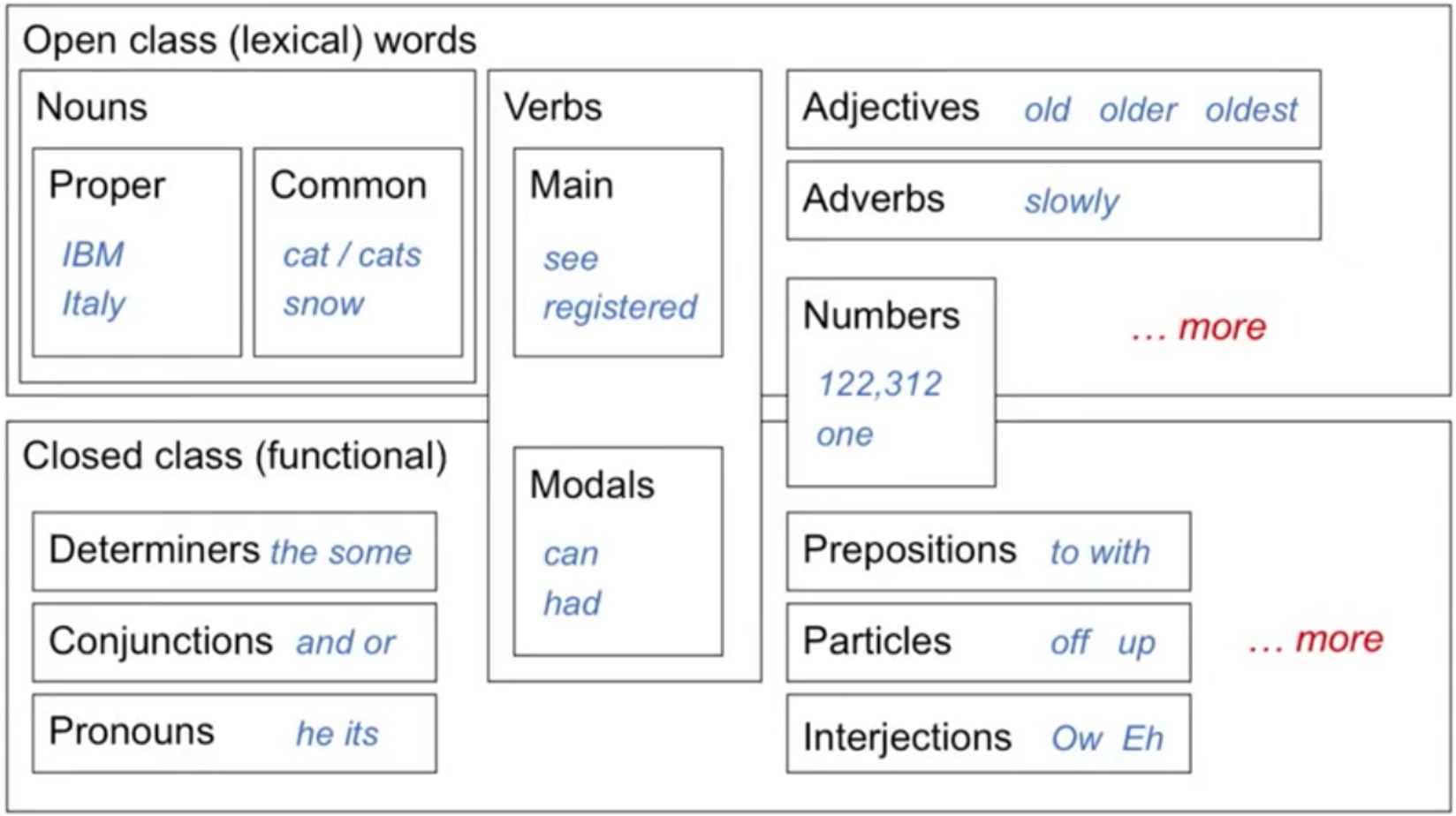
Classical *Part's of speech* are: nouns, verbs, pronouns, preopositions, adverbs, conjunctions, participles and articles.

Modern Transformer architecture based language modeling does not require traditional Part of Speech tagging anymore. But they nevertheless are helpful in understanding the building blocks to modern generative large language models and showcase the use of Hidden Markov Models that may help introduce the concept of a latent unobservable state or *memory* that is relevant for Recurrent Neural Networks.

- dimensionality reduction (removing words)
- word sense disambiguation
- Named Entity Recognition
- information extraction

$\Rightarrow$ we will introduce the formalization of common tools for POS tagging and introduce use pipelines in *R*.

Part of Speech Tagging



Open Versus Closed Class

Typically two types of high level groups are defined

- **Closed class**: considered as closed as the set of closed class words hardly changes over time.
  - determiners: a, an, the
  - pronouns: I, he, she, they
  - prepositions: over, under, near
- **Open class**: New entries into classes of all types, think of proper nouns becoming verbs - such as "Google" and "to google".

Word Class Ambiguity makes this a challenging task

Part of speech tagging is challenging as words can be members of multiple classes, depending on the *context* of use.

- `get/VB off/IN my/PRP$ back/NN`
- `win/VB the/DT voters/NNS back/RB`
- `I/PRP promise/VBP to/TO back/VB the/DT bill/NN ./.`

Part of speech tagging task is a relatively *easy* task. For every word that has ambiguity, there is a constrained set of ambiguous tags to chose from. Most POS implementation work off the information contained by the word itself and on information contained by small *windows* around the word.

The Penn Treebank Part-of-Speech Tagset

There are many lists of parts-of-speech, most modern language processing on English uses the 45-tag Penn Treebank tagset.

Tag	Description	Example	Tag	Description	Example
CC	coordin. conjunction	*and, but, or*	SYM	symbol	+,%, &
CD	cardinal number	*one, two*	TO	"to"	*to*
DT	determiner	*a, the*	UH	interjection	*ah, oops*
EX	existential 'there'	*there*	VB	verb base form	*eat*
FW	foreign word	*mea culpa*	VBD	verb past tense	*ate*
IN	preposition/sub-conj	*of, in, by*	VBG	verb gerund	*eating*
JJ	adjective	*yellow*	VBN	verb past participle	*eaten*
JJR	adj., comparative	*bigger*	VBP	verb non-3sg pres	*eat*
JJS	adj., superlative	*wildest*	VBZ	verb 3sg pres	*eats*
LS	list item marker	*1, 2, One*	WDT	wh-determiner	*which, that*
MD	modal	*can, should*	WP	wh-pronoun	*what, who*
NN	noun, sing. or mass	*llama*	WP$	possessive wh-	*whose*
NNS	noun, plural	*llamas*	WRB	wh-adverb	*how, where*
NNP	proper noun, sing.	*IBM*	$	dollar sign	*$*
NNPS	proper noun, plural	*Carolinas*	#	pound sign	*#*
PDT	predeterminer	*all, both*	"	left quote	' or "
POS	possessive ending	*'s*	"	right quote	' or "
PRP	personal pronoun	*I, you, he*	(	left parenthesis	[, (, {, <
PRP$	possessive pronoun	*your, one's*	)	right parenthesis	],), }, >
RB	adverb	*quickly, never*	,	comma	,
RBR	adverb, comparative	*faster*	.	sentence-final punc	. ! ?
RBS	adverb, superlative	*fastest*	:	mid-sentence punc	: ; ... – -
RP	particle	*up, off*			

There are other tagsets T with anything between 8 to 1,200, but this is the most commonly used

A naive POS tagger

The *baseline* POS tagger uses a simple tag-allocation rule: assign a tag $t^* \in T$ to a word w_j if

$$t^* = argmax P(t_i|w_j)$$

This tag allocation rule assigns the tag t to a word w_j that has the highest likelihood for that word. These conditional likelihoods can be estimated from some tagged *training* data.

It achieves accuracy of around 90%.

Reason for surprising high accuracy is due to fact that a lot of *stopwords* that make up the bulk of the quantity of tokens of text have mostly unambigous tags.

Accuracy of Naive POS

For example for the word `well`:

- `Get/VB well/RB soon/RB !/.`
- `This/DT oil/NN well/NN is/VBZ profitable/JJ ./.`

For the word `well`, a training corpus suggests

Part-of-Speech	Total	(over Absolute Total)	Probability
adv	237,644,762	337,697,034	81.03%
adj	38,018,925	"	11.26%
x	20,818,507	"	6.16%
noun	4,839,300	"	1.43%
verb	296,019	"	0.09%
pron	42,918	"	0.01%
.	17,877	"	0.01%
det	12,313	"	0
num	3,822	"	0
prt	2,270	"	0
adp	31	"	0
Totals	**337,697,034**		**100%**

This suggests that the naive model would suggest that the most likely class for the word is `RB` - adverb form.

A (shallow) deep dive: Hidden Markov Models

One direction to improve on baseline POS tagger is to use information contained in structure around a word. So suppose you have a word sequence $w_1, ..., w_n$ (like a sentence), then the optimization problem that you want to solve is to assign a sequence of tags $t_1, ..., t_n$ to these words, that maximizes the probability

$$(t_1, ..., t_n)* = argmax P((t_1, ..., t_n)|(w_1, ..., w_n))$$

The underlying (hidden) true states of the world is the correct tag sequence $t_1, ..., t_n$.
It is impractical (impossible) to estimate $P((t_1, ..., t_n)|(w_1, ..., w_n))$ directly from training data due to the sparsity. So we employ a simplifying assumption.

A (shallow) deep dive: Hidden Markov Models

We can apply *Bayes Rule*, so the optimization problem becomes

$$(t_1, ..., t_n)^* = argmax \frac{P((w_1, ..., w_n)|(t_1, ..., t_n))P(t_1, ..., t_n)}{P((w_1, ..., w_n))}$$

This optimization problem is equivalent to solving [why?]

$$(t_1, ..., t_n)^* = argmax P((w_1, ..., w_n)|(t_1, ..., t_n))P(t_1, ..., t_n)$$

A (shallow) deep dive: Hidden Markov Models

For Hidden Markov POS models, we make two additional assumptions

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

and

$$P((t_1, ..., t_n)) = \prod_{i=1}^{n} P(t_i|t_{i-1})$$

This allows us to rewrite the optimization problem as

$$(t_1, ..., t_n)^* = argmax \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1})$$

An example: Hidden Markov Models

Consider the sentence

`Janet will back the bill`

which is correctly tagged as

`Janet/NNP will/MD back/VB the/DT bill/NN`

We obtain the following information from a tagged training corpus.

An example: Hidden Markov Models

	Janet	will	back	the	bill
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0.000097	0
NN	0	0.000200	0.000223	0.000006	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

Displaying $P(w_i|t_i)$ and $P(t_i|t_{i-1})$.

An example: Hidden Markov Models

	NNP	MD	VB	JJ	NN	RB	DT
$<s>$	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096	0.0176	0.0014	0.0086	0.1216	0.0177	0.0068
RB	0.0068	0.0102	0.1011	0.1012	0.0120	0.0728	0.0479
DT	0.1147	0.0021	0.0002	0.2157	0.4744	0.0102	0.0017

Displaying $P(w_i|t_i)$ and $P(t_i|t_{i-1})$.

An example: Finding optimal path

	Janet	will	back	the	bill
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0.000097	0
NN	0	0.000200	0.000223	0.000006	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

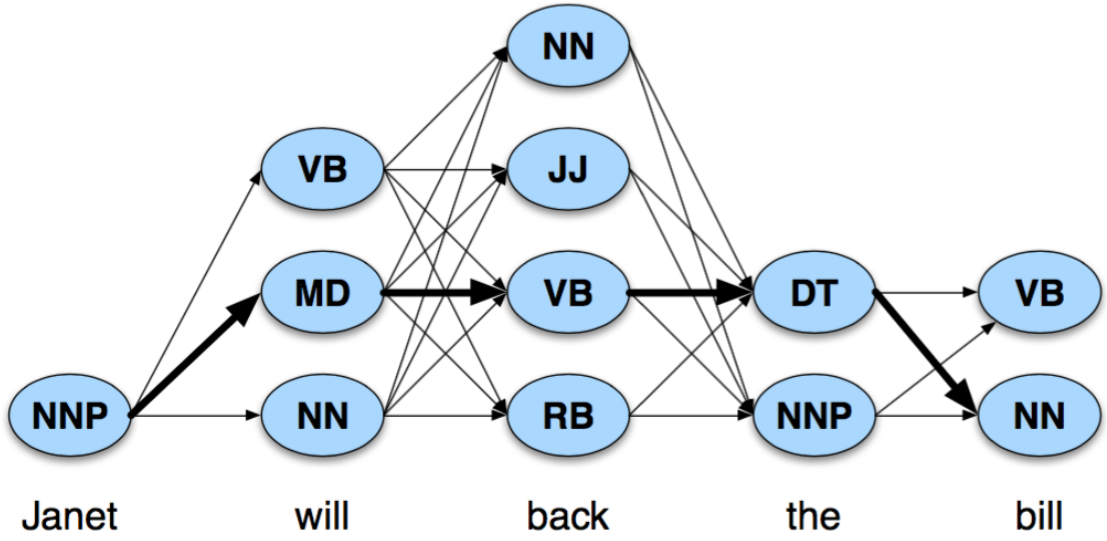
- In the corpus, the word `Janet` only appears with tag `NNP`.
- the word `will` has three possible tags `MD`, `VB`, `NN`.
- The probability that a random word of type modal (MD) is the word `will` is 0.31 = $P(\texttt{will}|\texttt{MD})$

An example: Finding optimal path

	NNP	MD	VB	JJ	NN	RB	DT
$<s>$	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096	0.0176	0.0014	0.0086	0.1216	0.0177	0.0068
RB	0.0068	0.0102	0.1011	0.1012	0.0120	0.0728	0.0479
DT	0.1147	0.0021	0.0002	0.2157	0.4744	0.0102	0.0017

- Transition matrix presents estimated $P(t_i|t_{i-1})$.
- the row sums should add to 1 - they dont since not the whole tagset is displayed.
- $P(\texttt{VB}||\texttt{MD}) = 0.79$, probability that `MD` is followed by tag `VB`.

An example: Finding optimal path



- The optimization problem can be modelled as an optimization problem on a *directed path*.
- We want to find the path that has highest likelihood.
- Brute forcing - the computation of all possible values for $\prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1})$ is computationaly extremely inefficient, and becomes infeasible very fast.
- *Viterbi algorithm* is a dynamic programming algorithm that solves this program efficiently.

Part of Speech Tagging in *R*

In *R* an easily accessible POS tagging tool that performs very well is accessible through the packages `OpenNLP`, which makes Apache's Open NLP platform accessible (`https://opennlp.apache.org/`).

It is a bit slow and the Apache NLP package is memory intensive (requests around)

Rather than working with a hidden markov model, its a maximum entropy classifier - which is just a fancy way of saying "logistic regression". We will introduce logistic regression for simple classification tasks.

Part of Speech Tagging in *R*

We will work with a developmental extension called the `tagger` package. Speed is an issue with NLP pipelines, OpenNLP extension takes arund 0.1 seconds per "sentence".

```
library(NLP)
library(openNLP)
# this is developmental, can be installed with the next two lines of code.
library(tagger)

## this installs 'pacman' which is a package to load developmental R extensions
if (!require("pacman")) install.packages("pacman")
pacman::p_load_gh(c("trinker/termco", "trinker/tagger"))

temp <- tag_pos("Janet will back the bill")

temp[[1]]

##     NNP      MD      VB      DT      NN
## "Janet"  "will"  "back"   "the"  "bill"

data.frame(tokens = temp[[1]], tags = names(temp[[1]]))

##     tokens tags
## NNP  Janet  NNP
## MD    will   MD
## VB    back   VB
## DT     the   DT
## NN    bill   NN
```

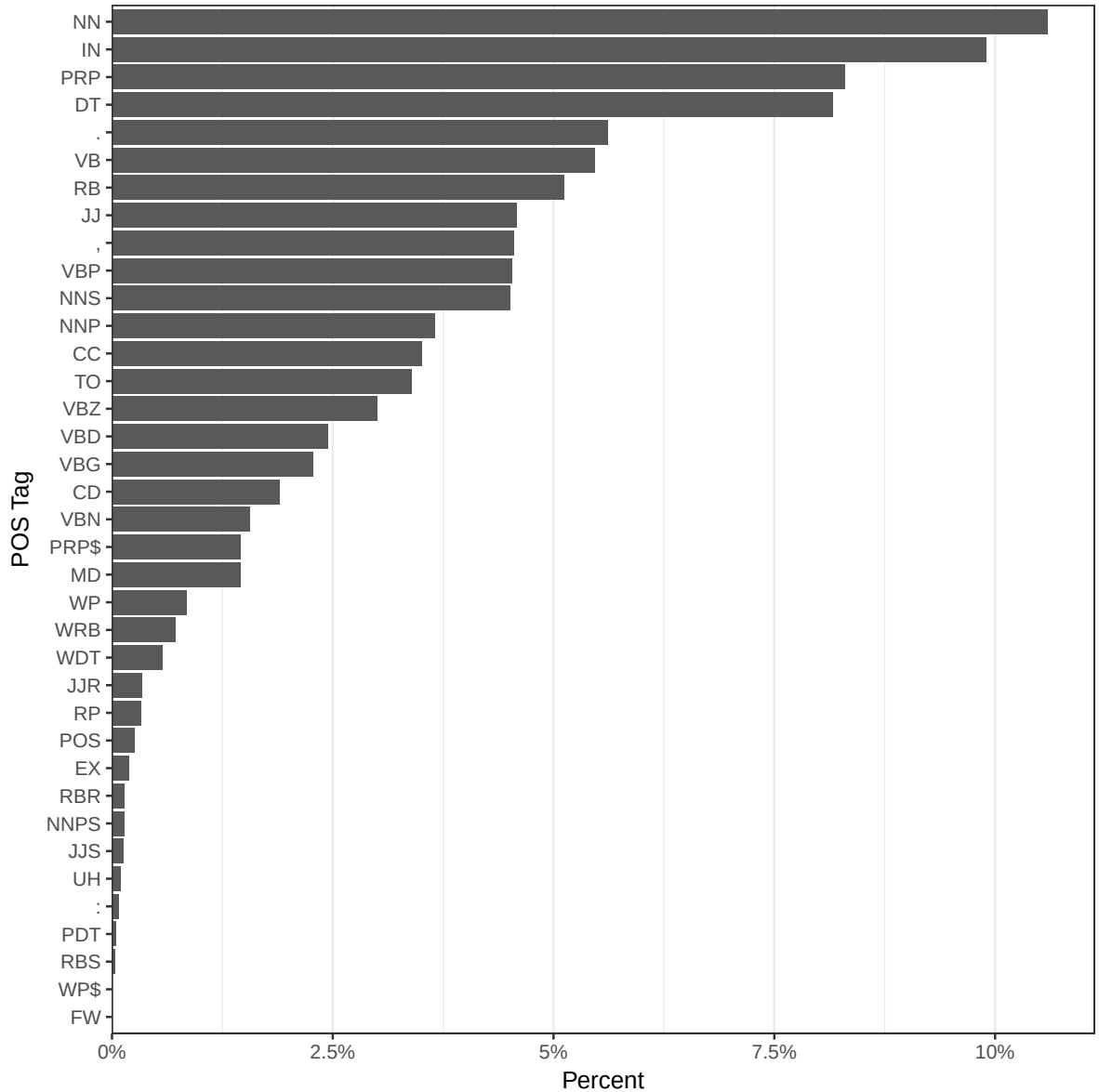
Part of Speech Tagging in *R*

```
data(presidential_debates_2012)

TAGGED <- tag_pos(presidential_debates_2012$dialogue)

head(TAGGED)
```

```
## [[1]]
##            PRP             MD             VB             IN             RB             IN
##           "We"           "'ll"         "talk"        "about" "specifically"        "about"
##             NN             NN             IN             DT             NN              .
##       "health"         "care"           "in"            "a"       "moment"            "."
##
## [[2]]
##         CC         WP        VBP        PRP         VB         DT         NN         NN
##      "But"     "what"       "do"      "you"  "support"      "the"  "voucher"   "system"
##          ,        NNP          .
##        "," "Governor"        "?"
##
## [[3]]
##         WP        PRP        VBP        VBZ         DT         NN         IN         JJ
##     "What"        "I"  "support"       "is"       "no"   "change"      "for"  "current"
##        NNS         CC         IN        NNS         TO        NNP          .
## "retirees"      "and"     "near" "retirees"       "to" "Medicare"        "."
##
## [[4]]
##          CC          DT          NN         VBZ         VBG          NN          CD
##       "And"       "the" "president"  "supports"    "taking"    "dollar"     "seven"
##          CD          CD          CD          IN          IN          DT          NN
##   "hundred"   "sixteen"   "billion"       "out"        "of"      "that"   "program"
##           .
##         "."
##
## [[5]]
##         CC         WP         IN         DT        NNS          .
##      "And"      "what"    "about"      "the" "vouchers"        "?"
```

Part of Speech Tagging in *R*

```
plot(TAGGED)
```



Going forward

Modern LLMs don't use POS tags explicitly, but they do learn internal representations that encode POS-like information implicitly.

POS tagging could still be useful as a structured inductive bias or an auxiliary signal during training or evaluation.