# DD2418 Language Engineering:
# 3a: Statistical language models

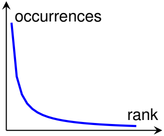
Johan Boye, KTH

# Statistical properties of language

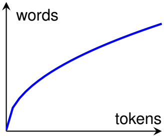
We will view words as *random events* generated by some *random process*.

Some high-level observations:

Zipf's law: The number of occurrences of a word is inversly proportional to the word's rank in a frequency table.




Heap's law: The number of unique words in a text is proportional to the square root of the number of tokens.



# Statistical language model

A (statistical) language model predicts the probability of the next word, based on the preceding words.

- *He wrote a* _______.
- *He wrote a letter* _______.
- *He wrote a letter to* _______.
- *He wrote a letter to his* _______.

# Applications of language models

- Word prediction
  - *I'm going* ______
    - to
    - for
    - on
- Spelling correction
  - *Flights form Boston.*
- Speech recognition
  - $P$(“recognize speech”) $>$ $P$(“wreck a nice beach”)
- Translation
  - $P$(“tall building”) $>$ $P$(“high building”)
- ... and many more

# Some notation

$\mathbf{P(w)}$ = the probability of the word $w$.

- Picking a random word from a text, what is the probability that that word will be $w$?

$\mathbf{P(w_1 w_2 \ldots w_n)}$ = the probability of the sequence $w_1\ w_2\ \ldots\ w_n$.

- Picking a random sequence of $i$ words from a text, what is the probability that that sequence will be $w_1\ w_2\ \ldots\ w_n$?

# Chain rule for probabilities

The *chain rule* rewrites a joint probability into a product of conditional probabilities.

$$P(A, B) = P(B|A)P(A)$$

In general:

$$P(A_1, \ldots, A_n) = \prod_{i=1}^{n} P(A_i|A_1, \ldots, A_{i-1})$$

For example:
$P(\text{I really like ants}) = P(\text{ants}|\text{I really like})P(\text{like}|\text{I really})P(\text{really}|\text{I})P(\text{I})$

We take $P(\text{ants}|\text{I really like})$ to mean
$P(w_i = \text{“ants”}|w_{i-i} = \text{“like”}, w_{i-2} = \text{“really”}, w_{i-3} = \text{“I”})$

# Markov assumption

Assume that a word only depends on the previous couple of words.

*Unigram* model: Don't consider the context.

$$P(\text{I have a unicorn}) = P(\text{I})P(\text{have})P(\text{a})P(\text{unicorn})$$

*Bigram* model: Look at the previous word.

$$P(\text{ I have a unicorn}) = \\ P(\text{I})P(\text{have}|\text{I})P(\text{a}|\text{have})P(\text{unicorn}|\text{a})$$

In general: An *n-gram model* takes the $n-1$ preceding words into account. Typically such models are estimated from a large corpus.

$$P(w_i|w_1,\dots,w_{i-1}) = P(w_i|w_{i-n+1},\dots,w_{i-1})$$

# DD2418 Language Engineering:
# 3b: Estimating n-gram models

Johan Boye, KTH

# Markov assumption

Assume that a word only depends on the previous couple of words.

*Unigram* model: Don't consider the context.

$$P(\text{I have a unicorn}) = P(\text{I})P(\text{have})P(\text{a})P(\text{unicorn})$$

*Bigram* model: Look at the previous word.

$$P(\text{ I have a unicorn}) = \\ P(\text{I})P(\text{have}|\text{I})P(\text{a}|\text{have})P(\text{unicorn}|\text{a})$$

In general: An *n-gram model* takes the $n-1$ preceding words into account. Typically such models are estimated from a large corpus.

$$P(w_i|w_1,\ldots,w_{i-1}) = P(w_i|w_{i-n+1},\ldots,w_{i-1})$$

# Maximum likelihood estimation (MLE)

How do we estimate n-gram probabilities $P(w_i|w_{i-n+1}, \ldots, w_{i-1})$?

Count word strings in a text corpus, compute fraction:

$$P(w_i|w_{i-n+1}, \ldots, w_{i-1}) = \frac{c(w_{i-n+1}, \ldots, w_{i-1}, w_i)}{c(w_{i-n+1}, \ldots, w_{i-1})}$$

In particular (bigram probabilities):

$$P(\text{to}|\text{like}) = \frac{c(\text{like to})}{c(\text{like})}$$

Unigram probabilities:

$$P(\text{like}) = \frac{c(\text{like})}{N}, \text{ where } N \text{ is the number of tokens in the corpus}$$

# Maximum likelihood estimation (MLE)

For a unigram model estimated from a corpus $w_1 \ldots w_k$, the likelihood function is given by:

$$P(w_1)P(w_2)\ldots P(w_k)$$

e.g. for the Bible:

$$P(\mathit{In})P(\mathit{the})P(\mathit{beginning})\cdots P(\mathit{Amen})P(.)$$

If $N = 790010$ and $c(\mathit{In}) = 359$ and $c(\mathit{the}) = 96660$, etc., then the above expression is **maximized** when

$$P(\mathit{In}) = \frac{359}{790010} \text{ and } P(\mathit{the}) = \frac{96660}{790010} \text{ etc.}$$

Similarly for bigrams, trigrams, etc.

# Counting bigrams

. . . $\boxed{\text{I would}}$ like to know your plans for . . .

I would: 1

# Counting bigrams

. . . I $\boxed{\text{would like}}$ to know your plans for . . .

I would: 1 would like: 1

# Counting bigrams

. . . I would $\boxed{\text{like to}}$ know your plans for . . .

I would: 1 would like: 1 like to: 1

# Bigram probabilities

The word *I* occurred 1000 times...

- ... 20 times, the next word was *like*
- ... 200 times, the next word was *am*
- ... 100 times, the next word was *have*
- etc.

From the counts, we can estimate *bigram probabilities*:

- $P(\mathit{like}|I) = 0.02$
- $P(\mathit{am}|I) = 0.2$
- $P(\mathit{have}|I) = 0.1$
- etc.

# Trigram probabilities

The sequence *I like* occurred 20 times...

- ... 5 times, the next word was *to*
- ... 4 times, the next word was *that*
- ... 1 time, the next word was *apples*
- etc.

From the counts, we can estimate *trigram probabilities*:

- $P(to|I\ like) = 0.25$
- $P(that|I\ like) = 0.2$
- $P(apples|I\ like) = 0.05$
- etc.

# Example

- Corpus with 19 tokens (including punctuation):
  *I live in Boston.*
  *I like ants.*
  *Ants like honey.*
  *Therefore I like honey too.*
- What is $P$(I like Boston) using a unigram model based on the above corpus?
- What is $P$(I like honey) using a bigram model?
- What is $P$(I like Boston) using a bigram model?

# Example

- Corpus with 19 tokens (including punctuation):

  *I live in Boston.*
  *I like ants.*
  *Ants like honey.*
  *Therefore I like honey too.*

- What is $P$(I like Boston) using a unigram model based on the above corpus? (3/19)(3/19)(1/19)
- What is $P$(I like honey) using a bigram model? (3/19)(2/3)(2/3)
- What is $P$(I like Boston) using a bigram model? (3/19)(2/3)(0/3) = 0

# Problems with Maximum likelihood estimation

*Data sparsity* is a problem for the straightforward MLE method.

E.g. $\frac{c(\text{“I have a unicorn”})}{c(\text{“I have a”})}$

- What if there are no occurrences of “I have a unicorn”?
- What if there are no occurrences of “I have a”?

Regardless of how much data you have, this will happen over and over.

# DD2418 Language Engineering:
# 3c: Zero probabilities, and what to do about them

Johan Boye, KTH

# Zero-probabilities, and what to do about them

A problem with *n*-gram models is that many sensible word sequences will have zero-probabilities.

3 techniques to solve this problems:

- Smoothing
- Backoff
- Linear interpolation

# Laplace smoothing

*Smoothing:* Transfer some of the probability mass from the seen sequences to the unseen sequences.

Easiest variant: *Laplace (add-one)* smoothing.

- Previously (Maximum Likelihood Estimation):

$$P_{MLE}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

- Now: (MLE with Laplace smoothing):

$$P_{Laplace}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

  where $V$ is the size of the vocabulary (number of unique words).

# Laplace smoothing (example)

- Corpus with 19 tokens (including punctuation):

  *I live in Boston.*
  *I like ants.*
  *Ants like honey.*
  *Therefore I like honey too.*

- What is $P$(I like honey) using a bigram model with Laplace smoothing?
- What is $P$(I like Boston) using a bigram model with Laplace smoothing?

# Laplace smoothing (example)

- Corpus with 19 tokens (including punctuation):

  *I live in Boston.*
  *I like ants.*
  *Ants like honey.*
  *Therefore I like honey too.*

- What is $P$(I like honey) using a bigram model with Laplace smoothing? $(\frac{3+1}{19+10})(\frac{2+1}{3+10})(\frac{2+1}{3+10})$
- What is $P$(I like Boston) using a bigram model with Laplace smoothing? $(\frac{3+1}{19+10})(\frac{2+1}{3+10})(\frac{0+1}{3+10})$

# Advanced smoothing

Laplace smoothing turns out to be too crude for many applications

- ...but it is useful in some machine learning contexts (more on this later in the course)

More sophisticated smoothing methods exist, e.g. the *interpolated Kneser-Ney* method.

We won't cover these in the course.

# Backoff

If a particular $n$-gram is not present in the training corpus, then use $(n-1)$-grams instead.

If the $(n-1)$-grams do not exist either, then use $(n-2)$-grams, etc.

Example: Suppose that $P(\text{ants}|\text{really like}) = 0$ in a trigram model. Then compute the probability as:

$$\hat{P}(\text{ants}|\text{really like}) = P(\text{ants}|\text{like})P(\text{like}|\text{really})$$

Note that this computation will most likely underestimate the actual probability (why?).

# Linear interpolation

Estimate $\hat{P}(w_i|w_{i-1})$ as

$$\lambda_1 P(w_i|w_{i-1}) + \lambda_2 P(w_i) + \lambda_3$$

where the $\lambda$s sum to 1.

$$\sum_i \lambda_i = 1$$

Typically $\lambda_1 = 0.99$, $\lambda_2 = 0.01 - \lambda_3$, $\lambda_3 = 10^{-6}$

This idea naturally extends to 3-grams, etc.

# Assignment 2, problem 1

Write a program that computes all bigram probabilites from a given (training) corpus, and stores it in a file. Practical issue:

We will use log-probabilities rather than probabilities

- ... using the natural logarithm (because it's simpler)
- $-11.99225$ rather than $0.0000061919939907$.

This is to avoid underflow when computing with very small probabilities.

... and we can add rather than multiply

- $\log(p_1 \times \ldots \times p_n) = \log(p_1) + \ldots + \log(p_n)$

# Probability of a sentence

When calculating the probability of a sentence, it is useful to include punctuation or boundary symbols, e.g.

$$P(\langle b \rangle \text{ I like Boston } \langle b \rangle) =$$
$$P(\text{I}|\langle b \rangle) \times P(\text{like}|\text{I}) \times P(\text{Boston}|\text{like}) \times P(\langle b \rangle|\text{Boston})$$

- $P(\text{I}|\langle b \rangle)$: Probability that "I" will be the first word of a sentence.
- $P(\langle b \rangle|\text{Boston})$: Probability that "Boston" will be the last word of a sentence.

# $n$-gram models and linguistic structure

How much linguistic structure is captured by $n$-gram-models?

- Higher $n \Rightarrow$ we capture more language structure, BUT
- Higher $n \Rightarrow$ we need more training data to get accurate probabilities.
- 4-grams and above require Google quantities of data, OR a restricted domain!

Long-distance dependencies will always be a problem (regardless of the choice of $n$):

> **The struggle** *between conservatives and socialists* **is** *being fought on many fronts.*

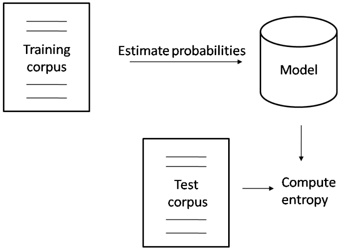
# DD2418 Language Engineering:
# 3d: Evaluation of language models

Johan Boye, KTH

# Evaluation of *n*-gram models

*Extrinsic evaluation:* Put your *n*-gram to use in an application, e.g. a machine translation system. Measure the performance.

*Intrinsic evaluation:* Compute the *entropy* of the model.



# Information

The *information* of an outcome having probability $p$ is

$$-\log_2 p$$

Information is measured in *bits*.

# Entropy

The *entropy* of a random variable $X$ is the expected value of the information.

$$H(X) = -\sum_{i=1}^{n} P(X = x_i) \log_2 P(X = x_i)$$

Special case: We assume that $0 \log_2 0 = 0$.

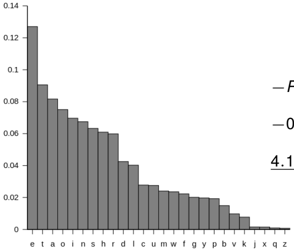
The entropy of $X$ is a measure of the difficulty of predicting the value of $X$.

# Entropy examples

Entropy for a−z using a uniform distribution:

$$-\sum_{i=1}^{26} \frac{1}{26} \log_2 \frac{1}{26} = -\log_2 \frac{1}{26} = \log_2 26 = \underline{4.7}$$

Entropy for a−z using probabilities for English:




$$-P(\mathrm{a}) \log_2 P(\mathrm{a}) \quad -P(\mathrm{b}) \log_2 P(\mathrm{b}) \quad - \ldots \quad -P(\mathrm{z}) \log_2 P(\mathrm{z}) =$$

$$-0.082 \log_2 0.082 \quad -0.015 \log_2 0.015 \quad - \ldots \quad -0.00074 \log_2 0.00074 =$$

$\underline{4.18}$

Entropy if $P(\mathrm{a}) = 1$: $-1 \log_2 1 - 0 \log_2 0 - \ldots 0 \log_2 0 = \underline{0}$

# Basic properties of entropy

More predictability $\Leftrightarrow$ lower entropy.

The entropy of a certain event is 0.

Maximum entropy is obtained if all outcomes are equally probable.

# Entropy and language

What is the entropy at different points in a sentence?

`The recent` ①

# Entropy and language

What is the entropy at different points in a sentence?

`The recent ① results mean that ②`

# Entropy and language

What is the entropy at different points in a sentence?

```
The recent ① results mean that ②
scientists now need to ③
```

# Entropy and language

What is the entropy at different points in a sentence?

```
The recent ① results mean that ②
scientists now need to ③ go back to
the drawing ④
```

# Entropy and language

What is the entropy at different points in a sentence?

```
The recent ① results mean that ②
scientists now need to ③ go back to
the drawing ④ board.
```

# Cross-entropy

Suppose `a,b,c,d` is distributed according to $p$:

$$p_{\mathtt{a}} = P(\mathtt{a}) = 0.5,\ p_{\mathtt{b}} = 0.2,\ p_{\mathtt{c}} = 0.2,\ p_{\mathtt{d}} = 0.1$$

but we *believe* it is distributed according to $q$:

$$q_{\mathtt{a}} = P(\mathtt{a}) = 0.1,\ q_{\mathtt{b}} = 0.2,\ q_{\mathtt{c}} = 0.2,\ q_{\mathtt{d}} = 0.5$$

The *cross-entropy of p on q* is then computed as:

$$H(X) = -\sum_{i=\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d}} p_i \log_2 q_i$$

Cross-entropy measures how difficult it is to predict the symbol under this belief.

# Entropy as an evaluation metric

Entropy can be used as an evaluation metric for language models.

Given a model P estimated from a *training corpus*, one can approximate the entropy as:

$$-\frac{1}{N}\log_2 P(w_1, w_2, \ldots, w_N)$$

where $w_1, w_2, \ldots, w_N$ is a very long sequence of words from a *test corpus*.

The above computation really approximates the *cross-entropy* of the test set on $P$, which is an upper bound of the entropy of $P$.

# Entropy as an evaluation metric

The *lower* the entropy of the test corpus, the *better* the language model learned from the training corpus.

The tacit assumption here is that the test corpus is representative of actual data.

# Assignment 2, problem 3

- Write a program that evaluates a language model (a model constructed with your program in (a)) on a given test set.
- Build a number of models and evaluate them on different test sets.
- Draw conclusions.