Language Technology (2021)

Language modelling

Marco Kuhlmann
Department of Computer and Information Science

Language modelling

- **Language modelling** is the task of assigning probabilities to the sentences of some language.
- In particular, we want to predict what the next word will be, given the words so far.
- Instead of on words, language models can also be defined on characters or sounds.

Language models for language identification

- **Language identification** is the problem of detecting the main language of a text document d.
- We rank languages based on the probability of observing d as a random sample of the candidate language L.

Language models for translation

Rank candidate translations for the following Chinese sentence:

他 向 记者 介绍了 主要 内容

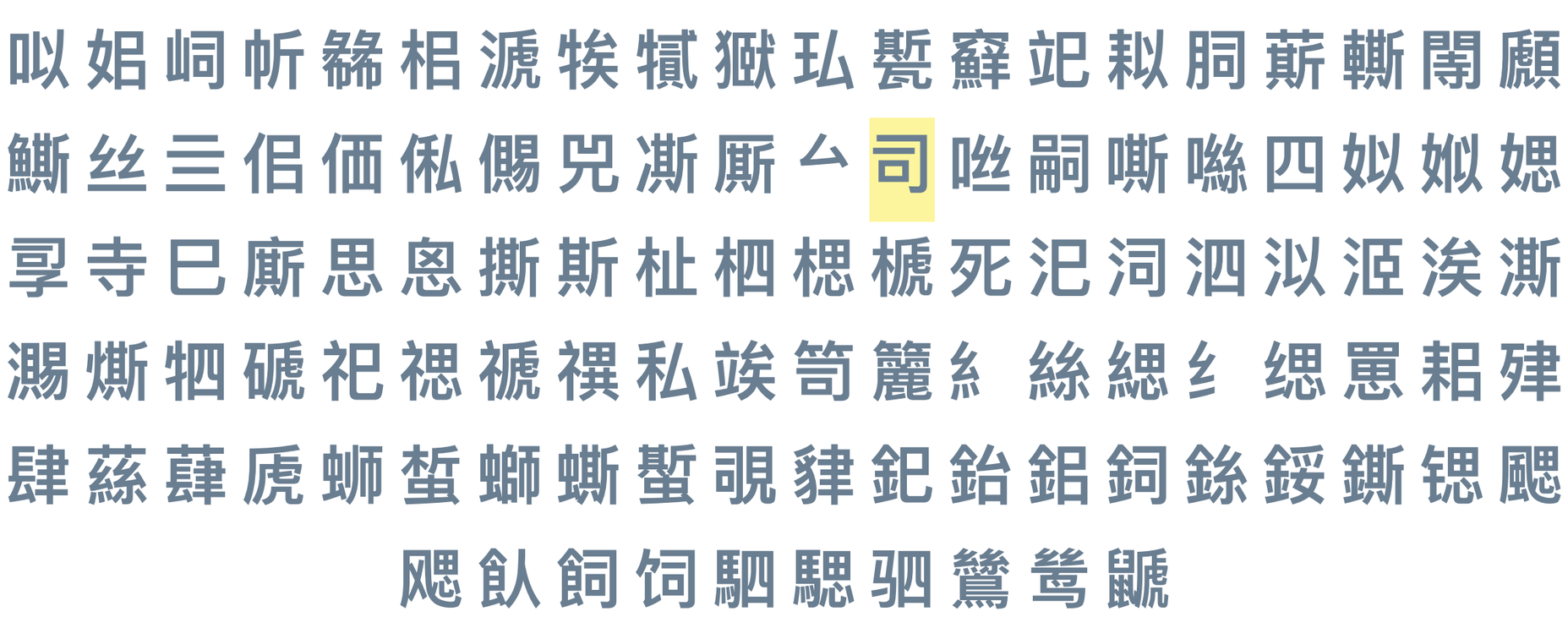
he to reporters introduced main content

- he introduced reporters to the main contents of the statement
- he briefed to reporters the main contents of the statement
- he briefed reporters on the main contents of the statement

Example from Jurafsky and Martin (2017)

Language models for predictive text input

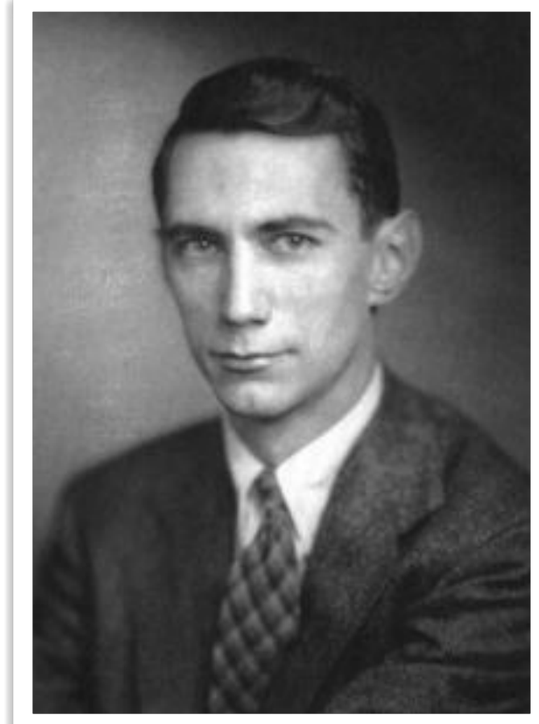
ping guo gong si → 蘋果公司



Shannon's game

Shannon's game is like Hangman except that

- It's no fun at all.
- You may only guess one character at a time.
- When moving on to the next character, all characters become available again.



Claude Shannon
(1916–2001)

Image source: Wikipedia

This lecture

- Introduction to language modelling
- Probabilistic language models
- Learning probabilistic language models
- Evaluation of language models
- Edit distance (partially advanced material)

Probabilistic language models

N-gram models

- An ***n*-gram** is a contiguous sequence of n words or characters.

  unigram (*Dear*), bigram (*Dear Sherlock*), trigram (*Dear Sherlock Holmes*)

- An ***n*-gram model** is a language model defined on n-grams – a probability distribution over sequences of n words.

- We are often interested in the conditional probability of seeing the last word in an n-gram, given the previous words:

$$P(w_n \mid w_1 \cdots w_{n-1})$$

Unigram model

A **unigram language model** is a bag-of-words model:

$$P(w_1 \cdots w_N) = \prod_{i=1}^{N} P(w_i)$$

Thus the probabilities of all words in the text are mutually independent.

Markov models

- The probability of each item depends only on the immediately preceding item.
- The probability of a sequence of items is the product of these conditional probabilities.
- For a well-defined model, we need to mark the beginning and the end of a sequence.

Андре́й Ма́рков
(1856–1922)

Image source: Wikipedia

Probability of a sequence of words

beginning-of-sentence

$$P(abc) = P(a \mid \text{BOS}) \cdot P(b \mid a) \cdot P(c \mid b) \cdot P(\text{EOS} \mid c)$$

end-of-sentence

Bigram models

A **bigram model** is a Markov model on sequences of words:

$$P(w_1 \cdots w_N) = P(w_1 \,|\, \text{BOS}) \cdot \prod_{i=2}^{N} \left[P(w_i \,|\, w_{i-1}) \right] \cdot P(\text{EOS} \,|\, w_N)$$

Thus the probability of a word
depends only on the immediately preceding word.

Formal definition of a bigram model

V a set of possible words (characters); the vocabulary

We assume that the vocabulary includes EOS.

$P(w|u)$ a probability that specifies how likely it is to observe the word w after the context unigram u

one value for each combination of a word w and a context u

Formal definition of an n-gram model

n — the model's order (1 = unigram, 2 = bigram, …)

V — a set of possible words (characters); the vocabulary

We assume that the vocabulary includes EOS.

$P(w|u)$ — a probability that specifies how likely it is to observe the word w after the context $(n-1)$-gram u

one value for each combination of a word w and a context u

Quiz

Simple uses of n-gram models

- **Prediction**

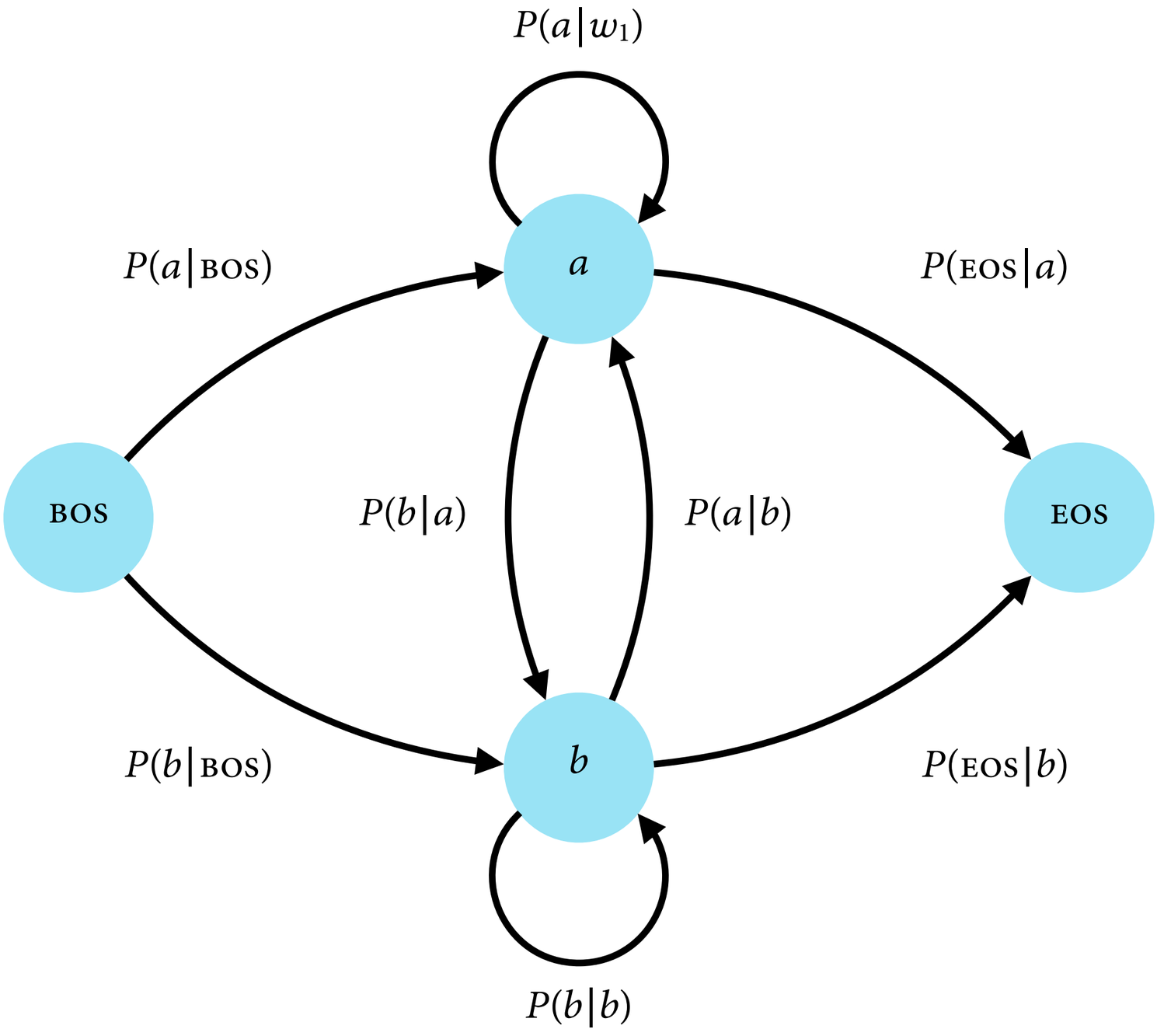
  To predict the next word, we can choose the word that has the highest probability among all possible words w:

  $$\text{predicted word} = \text{argmax}_w \, P(w | \text{preceding words})$$

- **Generation**

  We can generate a random sequence of words by sampling from the vocabulary with probability $P(w | \text{preceding words})$.

  does not produce meaningful text (without further conditioning)

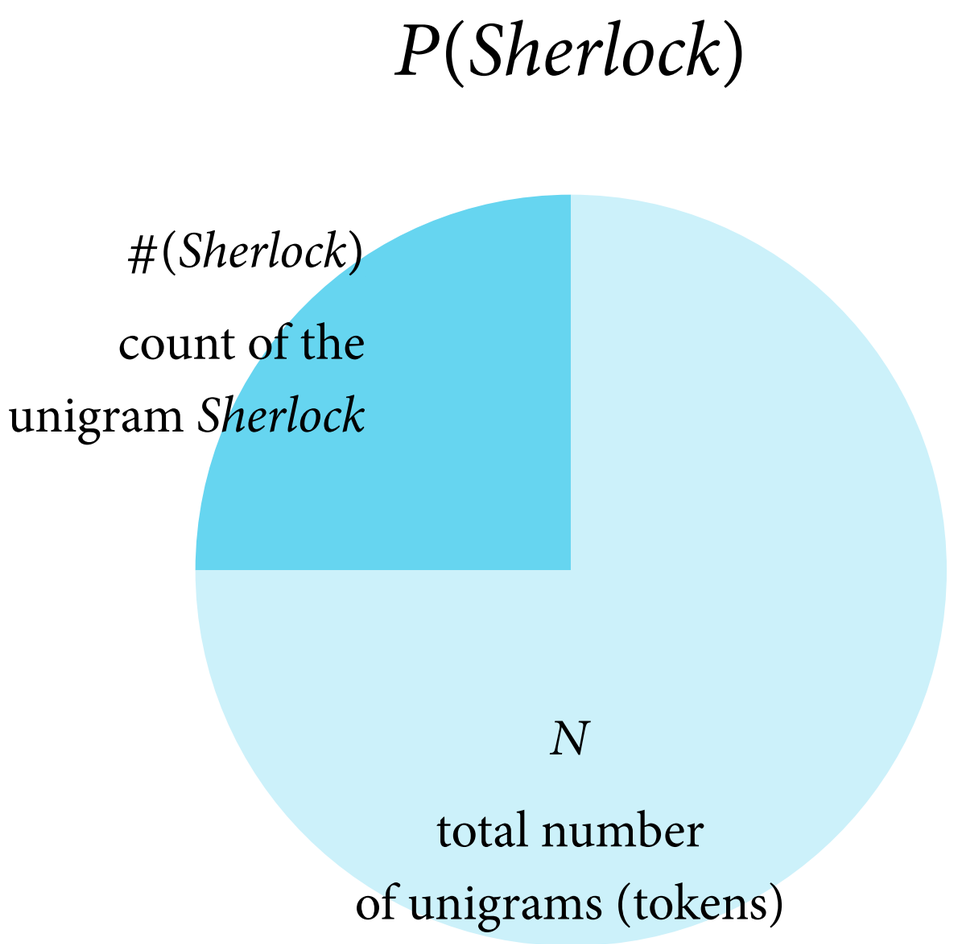

$P(a|w_1)$
$P(a|\text{BOS})$
a
$P(\text{EOS}|a)$
BOS
$P(b|a)$
$P(a|b)$
EOS
$P(b|\text{BOS})$
b
$P(\text{EOS}|b)$
$P(b|b)$

This lecture

- Introduction to language modelling
- Probabilistic language models
- Learning probabilistic language models
- Evaluation of language models
- Edit distance (partially advanced material)

Learning probabilistic language models

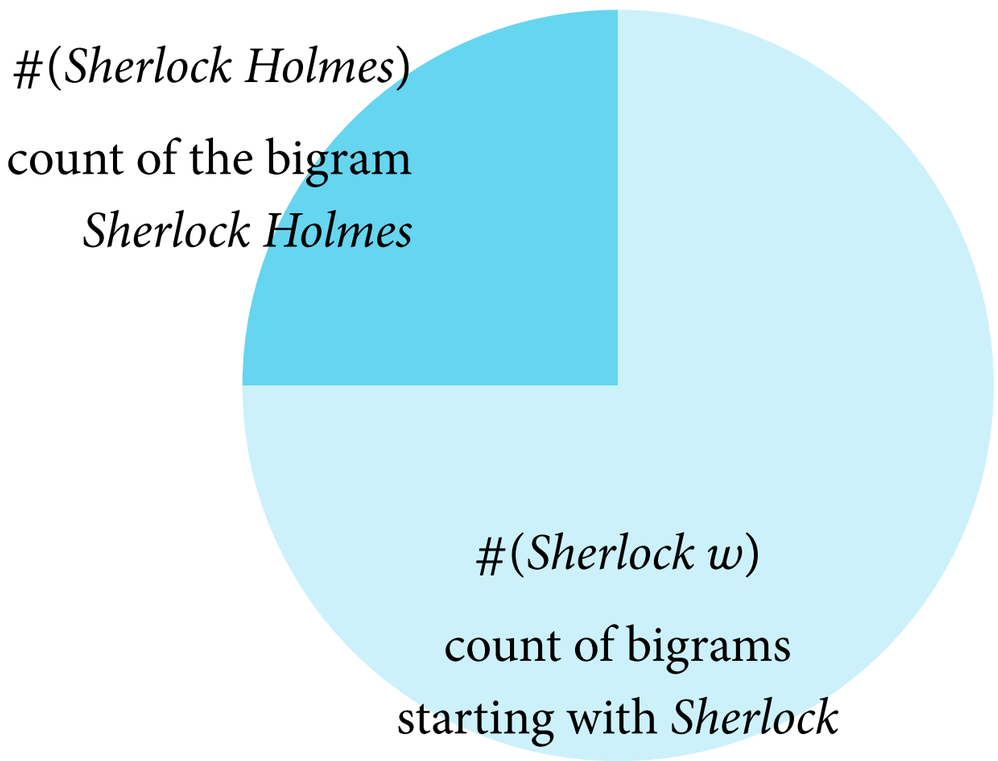
MLE of unigram probabilities



$$P(w) = \frac{\#(w)}{N}$$

MLE of bigram probabilities

$$P(Holmes \mid Sherlock)$$



$$P(w \mid u) = \frac{\#(uw)}{\#(u\bullet)}$$

$$P(w \mid u) = \frac{\#(uw)}{\#(u)}$$

Notation

N	number of word tokens (excluding BOS)
$\#(w)$	count of unigram (word) w
$\#(uw)$	count of bigram uw
$\#(u\bullet)$	count of bigrams starting with u
V	number of word types (including EOS and UNK, if used)

Source: Chen and Goodman (1998)

Sample exam problem

A Swedish corpus of 100,000 tokens contains 1,500 occurrences of the word *det*, 1,800 occurrences of the word *är*, 250 occurrences of the bigram *det är*, 10 occurrences of the word *sägs*, and 0 occurrences of the bigram *det sägs*.

Determine the following probabilities using MLE:

- the unigram probability $P(det)$
- the bigram probability $P(är|det)$
- the bigram probability $P(sägs|det)$

A problem with maximum likelihood estimation

- Shakespeare's collected works contain ca. 31,000 word types. There are 961 million different bigrams with these words.
- In his texts we only find 300,000 bigrams. This means that 99.97% of all theoretically possible bigrams have count 0.
- Under a bigram model, each sentence containing one of those bigrams will receive a probability of zero.

  Zero probabilities destroy information!

Additive smoothing

We can do add-k smoothing as for the Naive Bayes classifier:

$$P(w) = \frac{\#(w) + k}{N + kV}$$

$$P(w \mid u) = \frac{\#(uw) + k}{\#(u\bullet) + kV} = \frac{\#(uw) + k}{\#(u) + kV}$$

why?

A problem with additive smoothing

- We have only a constant amount of probability mass that we can distribute among the *n*-grams.

  probabilities still need to sum to one

- Therefore, although we are adding to the *count* of every *n*-gram, we are not adding to the *probability* of every *n*-gram.

- We actually *subtract* probability from observed *n*-grams and redistribute it equally among all possible *n*-grams.

A problem with additive smoothing

Suppose that we want to estimate a unigram model with vocabulary $V = \{A, B, C, D\}$ from the (single) training sentence 'A B C'.

Probabilities without smoothing				Probabilities with add-one smoothing			
A	B	C	D	A	B	C	D
1/3 = 0.33	1/3 = 0.33	1/3 = 0.33	0/3 = 0.00	2/7 = 0.29	2/7 = 0.29	2/7 = 0.29	1/7 = 0.14

Each observation loses 16% of its original probability mass.

Unknown words

- In addition to new bigrams, a new text may even contain completely new words. For these, smoothing will not help.
- One way to deal with this is to introduce a special word type UNK, and smooth it like any other word type in the vocabulary.

  For additive smoothing: Hallucinate k occurrences of the unknown word.

- At test time, we replace every unknown word with UNK.

This lecture

- Introduction to language modelling
- Probabilistic language models
- Learning probabilistic language models
- Evaluation of language models
- Edit distance (partially advanced material)

Evaluation of n-gram models

Intrinsic and extrinsic evaluation

- **Intrinsic evaluation**

  How does the method or model score with respect to a given evaluation measure?

  in classification: accuracy, precision, recall

- **Extrinsic evaluation**

  How much does the method or model help the application in which it is embedded?

  predictive input, machine translation, speech recognition

Intrinsic evaluation of language models, intuition

- Learn a language model from a set of training sentences and use it to compute the probability of a set of test sentences.
- If the language model is good, then the probability of the test sentences should be high.

  train on The Guardian, evaluate on The Daily Mail vs. Helsingin Sanomat

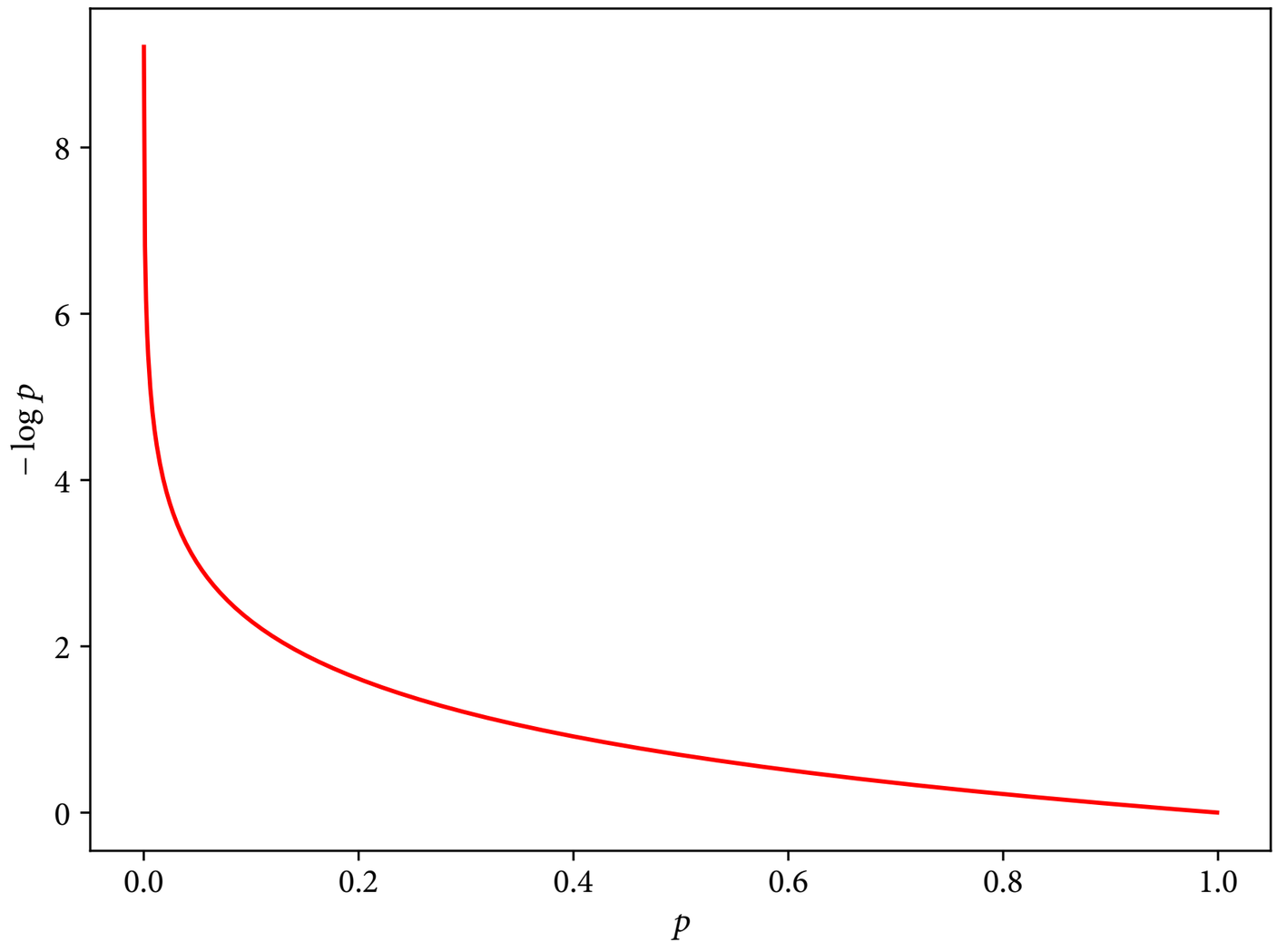
From probabilities to surprisal

- Instead of computing probabilities for the test sentences, we will compute negative log probabilities.
- Intuitively, this measures how 'surprised' we are about seeing the test sentences, given our language model.

  high probability = low surprisal

- The **entropy** of a language model on the test sentences is its average surprisal per word.

Negative log probabilities as a measure of surprisal



Entropy and perplexity

Assume that we have one long test sentence $w_1 \cdots w_N$.

The **entropy** of the language model on the test sentence is

$$-\frac{1}{N} \log_2 P(w_1 \cdots w_N)$$

The **perplexity** of the language model on the test sentence is

$$2^{\text{entropy}} = 2^{-\frac{1}{N} \log_2 P(w_1 \cdots w_N)}$$

The relation between entropy and order

- All other things being equal, a model with higher order has lower entropy than a model with lower order.
- Experiment: Train two n-gram models on the training section of the Wall Street Journal corpus, evaluate on the test section.

Unigram model	Trigram model
9.91	6.77

Entropy and smoothing

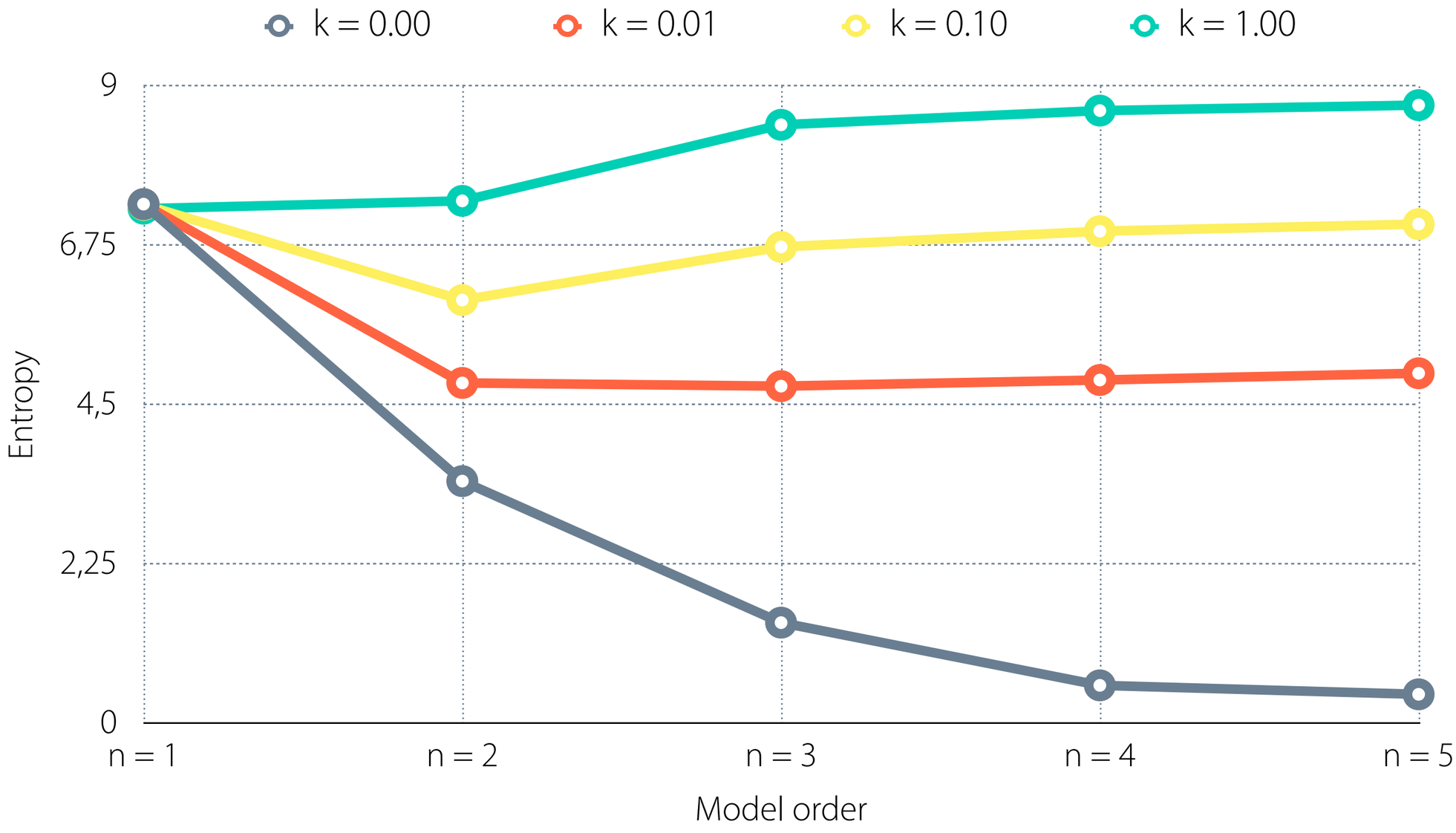
- When smoothing a language model, we are redistributing probability mass to observations we have never made.
- This leaves a smaller fraction of the probability mass to the observations that we actually did make during training.

  The training data becomes less likely.

- The more probability we are taking away from actual observations, the higher the entropy of the resulting model.

  assuming that test data and training data are similar

Entropy and smoothing



The problem with unknown words

- The test data will in general contain **unknown words** – words that we have not seen in the training data.
- Because we are multiplying probabilities, a single unknown word will bring down the probability of the test data to zero.

  Slogan: Zero probabilities destroy information

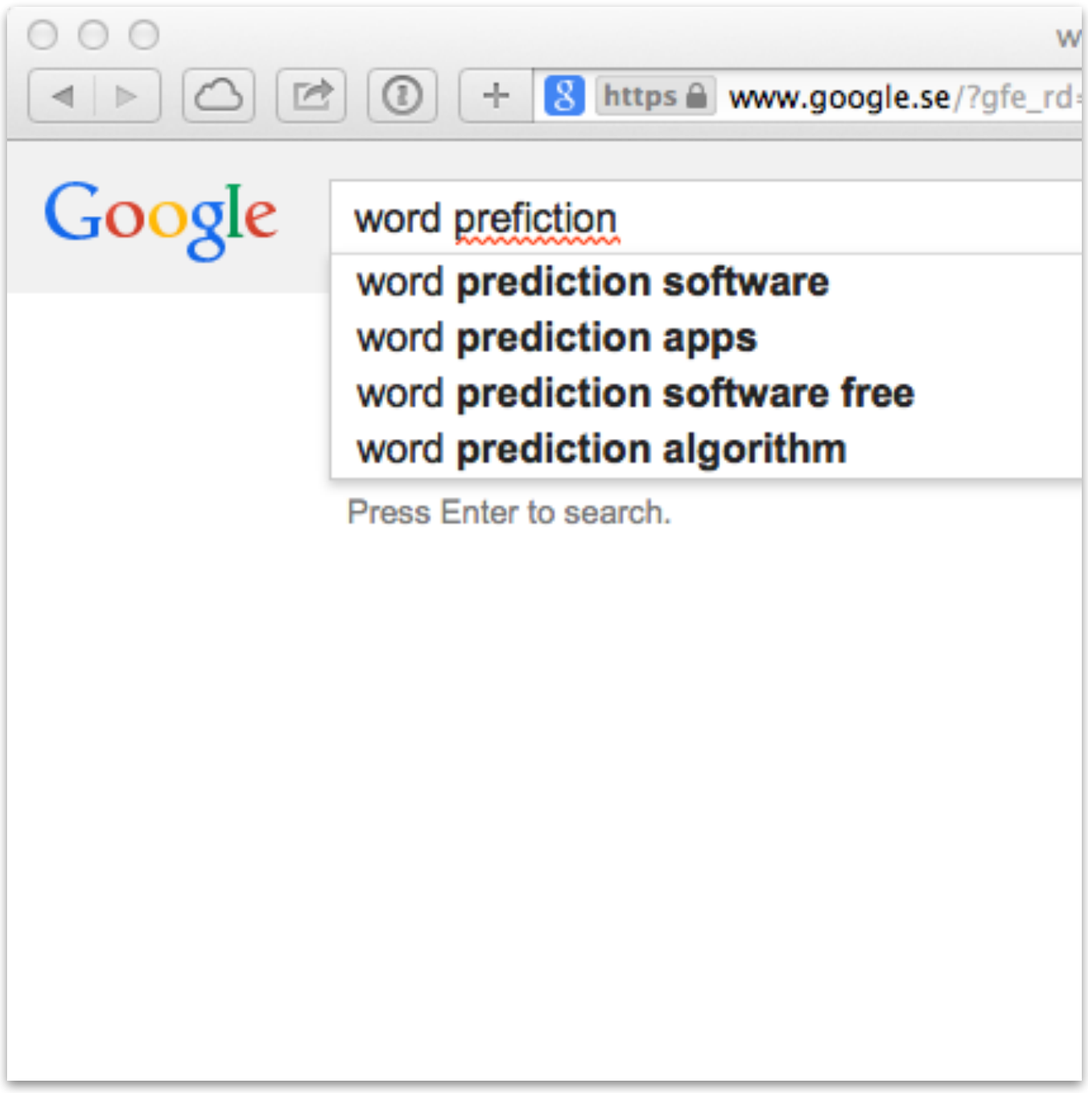
- The conclusion is that we should never compare language models with different vocabularies.

This lecture

- Introduction to language modelling
- Probabilistic language models
- Learning probabilistic language models
- Evaluation of language models
- Edit distance (partially advanced material)

Edit distance (partially advanced material)

Autocomplete and autocorrect



Edit distance

- Many misspelled words are quite similar to the correctly spelled words; there are typically only a few mistakes.

  lingvisterma, word prefiction

- Given a misspelled word, we want to identify one or several similar words and propose the most probable one.

- This idea requires a measure of the orthographic similarity between two words.

Edit operations

- We can measure the similarity between two words by the number of operations needed to transform one into the other.
- Here we assume three types of operations:

insertion	add a letter before or after another one
deletion	delete a letter
substitution	substitute a letter for another one

Edit operations, example

How many edits does it take to go from *intention* to *execution*?

	intention
delete the letter i	ntention
substitute e for n	etention
substitute x for t	exention
insert the letter c	execntion
substitute u for n	execution

Levenshtein distance

Each edit operation is assigned a cost:

- The cost for insertion and deletion is 1.
- The cost for substitution is 0 if the substituted letter is the same as the original one, and 1 in all other cases.

The **Levenshtein distance** between two words is the minimal cost for transforming one word into the other.

Computing the Levenshtein distance

- We would like to find a sequence of operations which transforms one word into the other and has minimal cost.
- The search space for this problem is huge – in fact, there are infinitely many sequences of operations!
- However, if we are only interested in sequences with *minimal* cost, we can solve the problem using dynamic programming.

  Wagner–Fischer algorithm (advanced material)

Other measures of edit distance

- Practical systems for spelling correction typically use more fine-grained weights than the ones that we use here.

  s instead of *a* is more probable than *d* instead of *a*

- We can still use the same algorithm for computing the Levenshtein distance; we only have to change the weights.

- A more realistic measure is the **Damerau–Levenshtein-distance**, which uses transposition as another edit operation.

  transposition = switching the positions of two adjacent characters

This lecture

- Introduction to language modelling
- Probabilistic language models
- Learning probabilistic language models
- Evaluation of language models
- Edit distance (partially advanced material)