Introduction

Goals:

- Estimate the probability that a given sequence of words occurs in a specific language.
- Model the most probable next word for a given sequence of words.

Jurafsky & Martin (2015), chapter 4, and Chen & Goodman (1999)

Table of contents

Motivation

Examples from Jurafsky & Martin (2015)

- (1) Please turn your homework ...
  What is a probable continuation? Rather *in* or *over* and not *refrigerator*.

- (2) a. all of a sudden I notice three guys standing on the sidewalk
  b. on guys all I of notice sidewalk three a sudden standing the
  Which of the two word orders is better?

Language model (LM): Probabilistic model that gives $P(w_1 \dots w_n)$ and $P(w_n | w_1 \dots w_{n-1})$

Motivation

Applications:

- Tasks in which we have to identify words in noisy, ambiguous input: **speech recognition**, **handwriting recognition**, ...
- **spelling correction**

Example

(3) a. their is only one written exam in this class
    b. there is only one written exam in this class

- **machine translation**: among a series of different word orders in the target language, one has to choose the best one.

Example

(4) a. Das Fahrrad wird er heute reparieren.
    b. The bike will he today repair
    c. The bike he will today repair.
    d. The bike he will repair today.

N-grams

Notation: $w_1^m = w_1 \ldots w_m$.
Question: How can we compute $P(w_1^m)$?

$$\begin{aligned} P(w_1^m) &= P(w_1)P(w_2|w_1)P(w_3|w_1^2)\ldots P(w_m|w_1^{m-1}) \\ &= \prod_{k=1}^{m} P(w_k|w_1^{k-1}) \end{aligned}$$

But: computing $P(w_k|w_1^{k-1})$ for a large k is not feasible.

Approximation of $P(w_k|w_1^{k-1})$: *N-grams*, i.e., look at just the $n-1$ last words, $P(w_k|w_{k-n+1}^{k-1})$.

Special cases:

- *unigrams*: $P(w_k)$
- *bigrams*: $P(w_k|w_{k-1})$
- *trigrams*: $P(w_k|w_{k-2}w_{k-1})$

N-grams

With n-grams, we get

1. Probability of a sequence of words:

$$P(w_1^l) \approx \prod_{k=1}^{l} P(w_k | w_{k-n+1}^{k-1})$$

2. Probability of a next word:

$$P(w_l | w_1^{l-1}) \approx P(w_l | w_{l-n+1}^{l-1})$$

These are strong independence assumptions called *Markov* assumptions. E.g. with bigrams

Example

$P(\text{einfach}|\text{die Klausur war nicht}) \approx P(\text{einfach}|\text{nicht})$

Maximum likelihood estimation (MLE)

Question: How do we estimate the n-gram probabilities?

Maximum likelihood estimation (MLE): Get n-gram counts from a corpus and normalize so that the values lie between 0 and 1.

$$P(w_k|w_{k-n+1}^{k-1}) = \frac{C(w_{k-n+1}^{k-1} w_k)}{C(w_{k-n+1}^{k-1})}$$

In the bigram case, this amounts to

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

We augment sentences with an initial $\langle \texttt{s} \rangle$ and a final $\langle \texttt{/s} \rangle$

Maximum likelihood estimation (MLE)

Example from Jurafsky & Martin (2015)

Training data:

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

Some bigram probabilities:

$$P(\texttt{I}|\texttt{< s >}) = \frac{2}{3} \qquad P(\texttt{Sam}|\texttt{< s >}) = \frac{1}{3} \qquad P(\texttt{am}|\texttt{I}) = \frac{2}{3}$$
$$P(\texttt{< /s >}|\texttt{Sam}) = \frac{1}{2} \qquad P(\texttt{Sam}|\texttt{am}) = \frac{1}{2} \qquad P(\texttt{do}|\texttt{I}) = \frac{1}{3}$$

Maximum likelihood estimation (MLE)

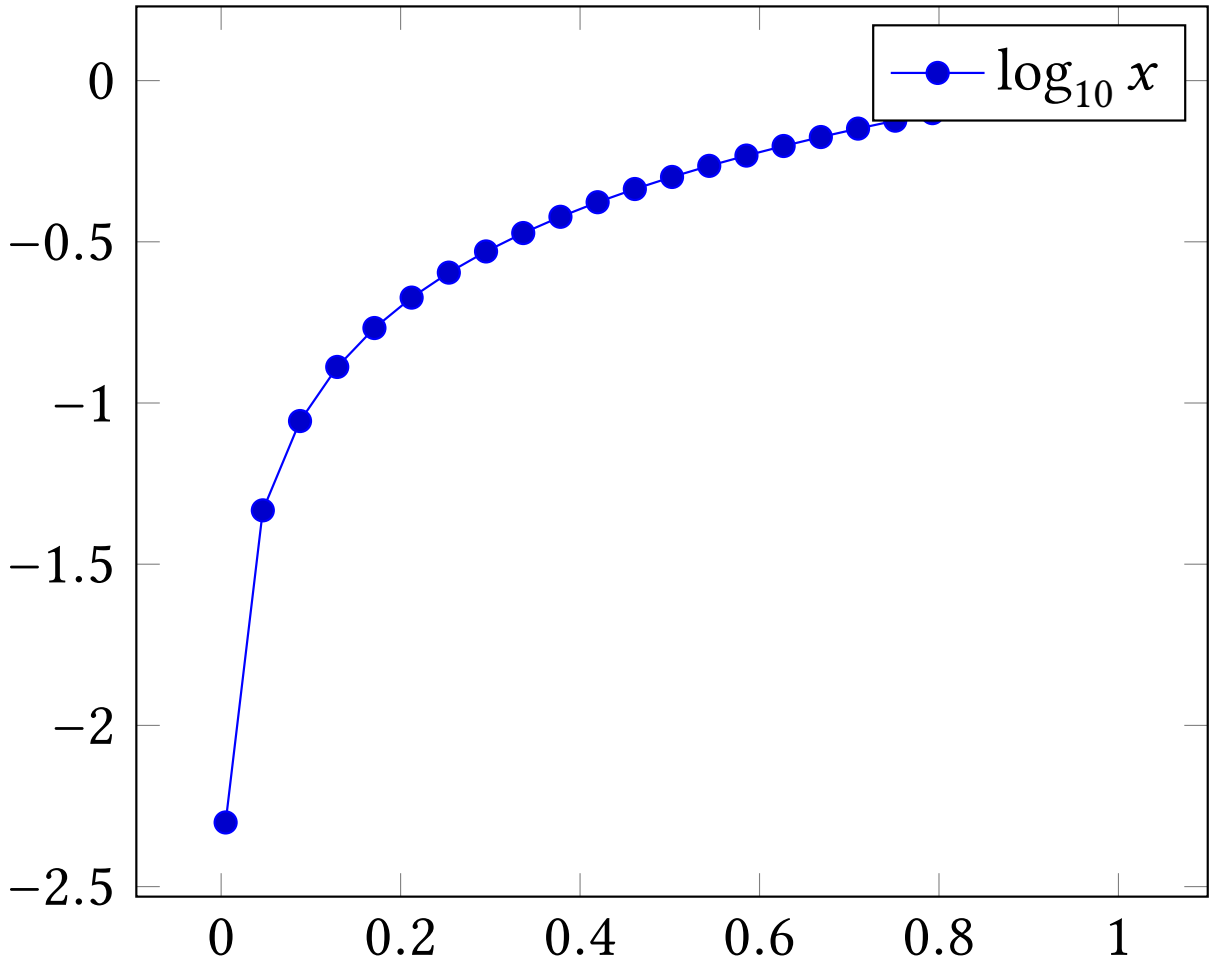
Practical issues:

- In practice, n is mostly between 3 and 5, i.e., we use trigrams, 4-grams or 5-grams.
- LM probabilities are always represented as *log probabilities*. Advantage: Adding replaces multiplying and numerical overflow is avoided.

$$p_1 \cdot p_2 \cdot \ldots p_l = \exp(\log p_1 + \log p_2 + \cdots + \log p_l)$$

Maximum likelihood estimation (MLE)

Reminder: $\log 1 = 0, \log 0 = -\infty$



Evaluating language models

The data is usually separated into

- a training set (80% of the data),
- a test set (10% of the data),
- and sometimes a development set (10% of the data).

The model is estimated from the training set. The higher the probability of the test set, the better the model.
Instead of measuring the probability of the test set, LMs are usually evaluated with respect to the *perplexity* of the test set. The higher the probability, the lower the perplexity.

Evaluating language models

The perplexity of a test set $W = w_1 w_2 \ldots w_N$ is defined as

$$\begin{aligned} PP(W) &= P(W)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(W)}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}} \\ &= \sqrt[N]{\frac{1}{\prod_{k=1}^{N} P(w_k | w_1^{k-1})}} \end{aligned}$$

With our n-gram model, we get then for the perplexity:

$$PP(W) = \sqrt[N]{\frac{1}{\prod_{k=1}^{N} P(w_k | w_{k-n+1}^{k-1})}}$$

Evaluating language models

A different way to think about perplexity: it measures the *weighted average branching factor* of a language.

Example

$L = \{a, b, c, d\}^*$. Frequencies are such that $P(a) = P(b) = P(c) = P(d) = \frac{1}{4}$ (independent from the context).
For any $w \in L$, given this model, we obtain

$$PP(w) = \sqrt[|w|]{\frac{1}{\prod_{k=1}^{|w|} \frac{1}{4}}} = \sqrt[|w|]{\frac{1}{\frac{1}{4}^{|w|}}} = \sqrt[|w|]{4^{|w|}} = 4$$

The perplexity of any $w \in L$ under this model is 4.

Evaluating language models

Example

$L = \{a, b, c, d\}^*$. Words in the language contain three times as many as as they contain bs, cs or ds. $P(a) = \frac{1}{2}$ and $P(b) = P(c) = P(d) = \frac{1}{6}$. For any $w \in L$ with these frequencies and with $|w| = 6n$:

$$PP(w) = \sqrt[6n]{\frac{1}{\prod_{k=1}^{n} \frac{1}{2 \cdot 2 \cdot 2 \cdot 6 \cdot 6 \cdot 6}}} = \sqrt[6n]{2^{6n} \cdot \sqrt{3}^{6n}} = 2\sqrt{3} = 3.46$$

Assume that we use the same model but test it on a W with equal numbers of as, bs, cs and ds, $|W| = 4n$. Then we get

$$PP(W) = \sqrt[4n]{\frac{1}{\prod_{k=1}^{n} \frac{1}{2 \cdot 6 \cdot 6 \cdot 6}}} = \sqrt[4n]{2^{4n} \cdot 3^{3n}} = 2\sqrt[4n]{3^{\frac{3}{4}4n}} = 2\sqrt[4]{27} = 4.56$$

Unknown words

Problem: New text can contain

- unknown words; or
- unseen n-grams.

In these cases, with the algorithm seen so far, we would assign probability 0 to the entire text. (And we would not be able to compute perplexity at all.)

Example from (Jurafsky & Martin, 2015)

Words following the bigram *denied the* in WSJ Treebank 3 with counts:

denied the allegations	5
denied the speculation	2
denied the rumors	1
denied the report	1

If the test set contains *denied the offer* or *denied the loan*, the model would estimate its probability as 0.

Unknown words

Unknown or *out of vocabulary* words:

- Add a pseudo-word ⟨UNK⟩to your vocabulary.
- Two ways to train the probabilities concerning ⟨UNK⟩:
  1. Choose a vocabulary V fixed in advance. Any word $w \notin V$ in the training set is converted to ⟨UNK⟩. Then estimate probabilities for ⟨UNK⟩as for all other words.
  2. Replace the first occurrence of every word w in the training set with ⟨UNK⟩. Then estimate probabilities for ⟨UNK⟩as for all other words.

Smoothing

Unseen n-grams: To avoid probabilities 0, we do *smoothing*: Take off some probability mass from the events seen in training and assign it to unseen events.

Laplace Smoothing (or *add-one* smoothing):

- Add 1 to the count of all n-grams in the training set before normalizing into probabilities.
- Not so much used for n-grams but for other tasks, for instance text classification.

- For unigrams, if N is the size of the training set and $|V|$ the size of the vocabulary, we replace $P(w) = \frac{C(w)}{N}$ with $P_{Laplace}(w) = \frac{C(w)+1}{N+|V|}$.
- For bigrams, we replace $P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$ with $P_{Laplace}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+|V|}$.

Smoothing

Smoothing methods for n-grams that use the $(n-1)$-grams, $(n-2)$-grams etc.:

- *Backoff*: use the trigram if it has been seen, otherwise fall back to the bigram and, if this has not been seen either, to the unigram.
- *Interpolation*: Use always a weighted combination of the trigram, bigram and unigram probabilities.

Linear interpolation:

$$\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1 P(w_n|w_{n-2}w_{n-1}) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n)$$

with $\sum_i \lambda_i = 1$.

References

Chen, Stanley F. & Joshua Goodman. 1999. An empirical study of smoothing techniques for language modeling. *Computer Speech and Language* 13. 359–394.

Church, K. W. & W. A. Gale. 1991. A comparison of the enhanced Good-Turing and deleted estimation methods for estimating probabilities of English bigrams. *Computer Speech and Language* 5. 19–54.

Jurafsky, Daniel & James H. Martin. 2015. Speech and language processing. an introduction to natural language processing, computational linguistics, and speech recognition. Draft of the 3rd edition.

Kneser, R. & H. Ney. 1995. Improved backing-off for m-gram language modeling. In *Proceedings of the IEEE international conference on acoustics, speech and signal processing*, vol. 1, 181–184. Detroit, MI.

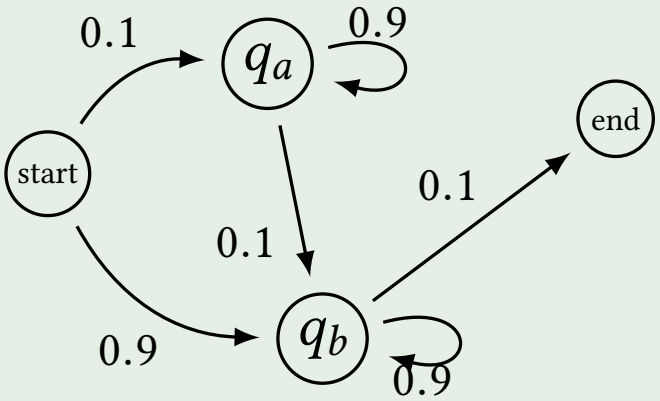
Motivation

Hidden Markov Models (**HMMs**)

- are weighted finite state automata
- with states emitting outputs.
- Transitions and emissions are weighted.

Motivation

HMM



in q_a, a is emitted with probability 0.9;
in q_a, b is emitted with probability 0.1;
in q_b, b is emitted with probability 0.9;
in q_b, a is emitted with probability 0.1;

Given a sequence bb, what is the most probable path traversed by the automaton, resulting in this output?

The states in the automaton correspond to classes, i.e., the search for the most probable path amounts to the search for the most probable class sequence.

Example

POS Tagging: the classes are POS tags, the emissions are words

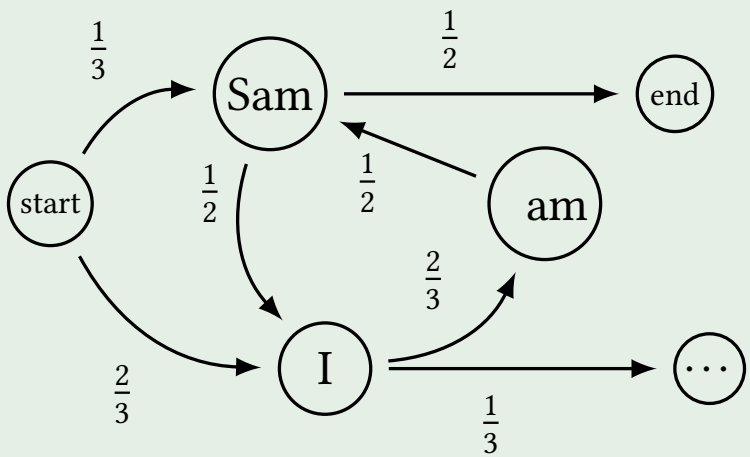
Hidden Markov Models

A **Markov chain** is a weighted automaton in which

- weights are probabilities, i.e., all weights are between 0 and 1 and the sum of the weights of all outgoing edges of a state is 1, and
- the input sequence uniquely determines the states the automaton goes through.

A Markov chain is actually a bigram language model.

Markov Chain for LM example slide 9



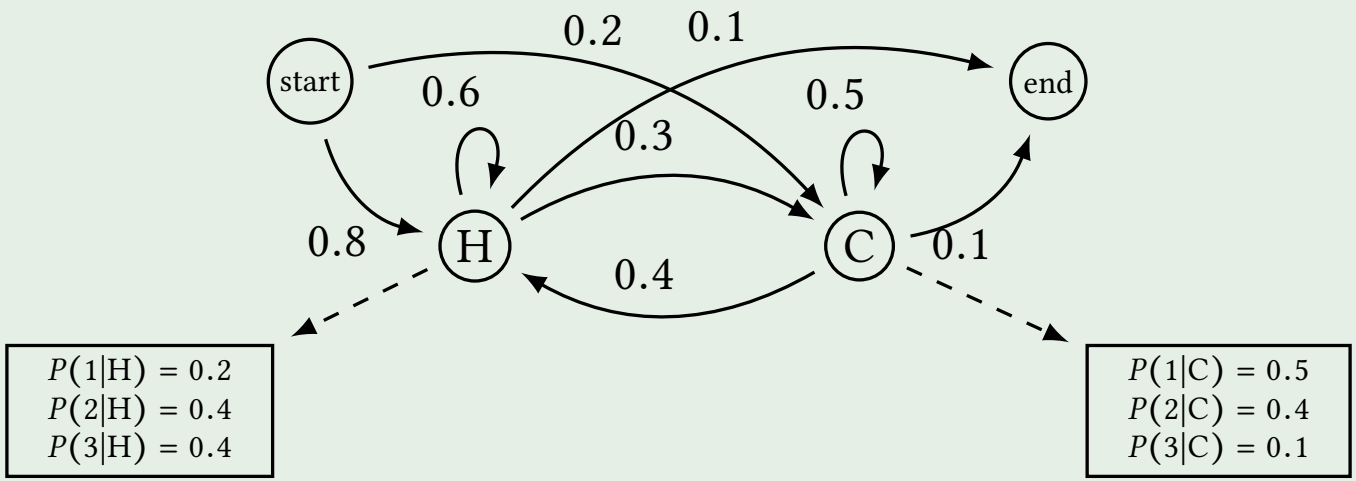
Hidden Markov Models

Markov chains are useful when we want to compute the probability for a sequence of events that we can observe.

Hidden Markov Models compute probabilities for a sequence of hidden events, given a sequence of observed events.

Example from Jurafsky & Martin (2015), after Eisner (2002)

HMM for inferring weather (hot = H, cold = C) from numbers of ice creams eaten by Jason.



Hidden Markov Models

HMM

A HMM is a tuple $\langle Q, A, O, B, q_0, q_F \rangle$, with

- $Q = \{q_1, \ldots, q_N\}$ a finite set of states;
- A a $|Q| \times |Q|$ matrix, the transition probability matrix, with $0 \leq a_{ij} \leq 1$ for all $1 \leq i \leq |Q|, 1 \leq j \leq |Q|$.
- O a finite set of observations;
- a sequence B of $|Q|$ functions $b_i : O \to \mathbb{R}$, the emission probabilities with $\sum_{o \in O} b_i(o) = 1$ for all $1 \leq i \leq |Q|$
- $q_0, q_F \notin Q$ are special start and end states, associated with transition probabilities a_{0i} and a_{iF} ($0 \leq a_{0i}, a_{iF} \leq 1$) for all $1 \leq i \leq |Q|$.

For all i, $0 \leq i \leq |Q|$, it holds that $\sum_{j=1}^{|Q|} a_{ij} + a_{iF} = 1$.

Hidden Markov Models

First order HMMs make the following simplifying assumptions:

1. **Markov assumption**: In a sequence of states $q_{i_1} \ldots q_{i_n}$, we assume

$$P(q_{i_n}|q_{i_1} \ldots q_{i_{n-1}}) = P(q_{i_n}|q_{i_{n-1}}) (= a_{i_{n-1} i_n})$$

2. **Output independence**: In a sequence of states $q_{i_1} \ldots q_{i_n}$ with the associated output sequence $o_{i_1} \ldots o_{i_n}$, we assume

$$P(o_{i_j}|q_{i_1} \ldots q_{i_j} \ldots q_{i_{n-1}}, o_{i_1} \ldots o_{i_j} \ldots o_{i_{n-1}}) = P(o_{i_j}|q_{i_j}) (= b_{i_j(o_{i_j})})$$

Hidden Markov Models

The following three fundamental problems have to be solved:

1. **Likelihood**: Given an HMM and an observation sequence, determine the likelihood of the observation sequence.
2. **Decoding**: Given an HMM and an observation sequence, discover the best hidden state sequence leading to these observations.
3. **Learning**: Given the set of states of an HMM and an observation sequence, learn the HMM parameters A and B.

Likelihood computation

Given an HMM and an observation $\mathbf{o} = \mathbf{o}_1 \dots \mathbf{o}_n$, determine the likelihood of the observation sequence $\mathbf{o}$.

For a specific state sequence $\mathbf{q} = \mathbf{q}_1 \dots \mathbf{q}_n \in Q^n$, we have

$$P(\mathbf{o}, \mathbf{q}) = P(\mathbf{o}|\mathbf{q})P(\mathbf{q})$$

and with our independence assumptions

$$P(\mathbf{o}, \mathbf{q}) = \prod_{i=1}^{n} P(\mathbf{o}_i|\mathbf{q}_i) \prod_{i=1}^{n} P(\mathbf{q}_i|\mathbf{q}_{i-1})$$

For the probability of $\mathbf{o}$, we have to sum over all possible state sequences:

$$P(\mathbf{o}) = \sum_{\mathbf{q} \in Q^n} \prod_{i=1}^{n} P(\mathbf{o}_i|\mathbf{q}_i) \prod_{i=1}^{n} P(\mathbf{q}_i|\mathbf{q}_{i-1}) P(q_F|\mathbf{q}_n)$$

Likelihood computation

Computing each element of the sum independently from the others would lead to an exponential time complexity.

Instead, we adopt the **forward algorithm** that implements some **dynamic programming**.

Idea: We fill a $n \times |Q|$ matrix α such that

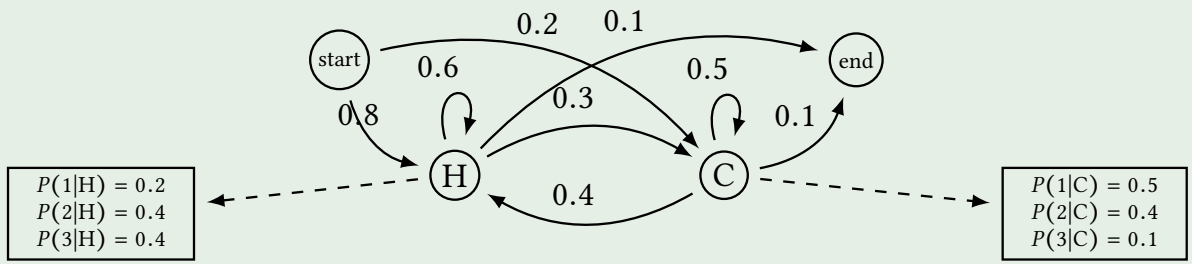
$$\alpha_{ij} = P(\mathbf{o}_1 \dots \mathbf{o}_i, \mathbf{q}_i = q_j)$$

Forward algorithm

1. $\alpha_{1j} = a_{0j} b_j(\mathbf{o}_1)$ for $1 \leq j \leq |Q|$
2. $\alpha_{ij} = \sum_{k=1}^{|Q|} \alpha_{i-1k} a_{kj} b_j(\mathbf{o}_i)$ for $1 \leq i \leq n$ and $1 \leq j \leq |Q|$
3. $P(\mathbf{o}) = \sum_{k=1}^{|Q|} \alpha_{nk} a_{kF}$

Likelihood computation

Ice cream weather example continued



$P(313)$:

H	0.32	$3.92 \cdot 10^{-2}$	$1.79 \cdot 10^{-2}$
C	$2 \cdot 10^{-2}$	$5.3 \cdot 10^{-2}$	$3.81 \cdot 10^{-3}$
	3	1	3

$P(313) = 2.17 \cdot 10^{-3}$

$$\alpha_{1j} = a_{0j} b_j(\mathbf{o}_1)$$
$$\alpha_{ij} = \sum_{k=1}^{|Q|} \alpha_{i-1k} a_{kj} b_j(\mathbf{o}_i)$$
$$P(\mathbf{o}) = \sum_{k=1}^{|Q|} \alpha_{nk} a_{kF}$$

Likelihood computation

Alternatively, we can also compute the **backward** probability β.

For a given observation $\mathbf{o}_i$ in our observation sequence, the forward probability considers the part from $\mathbf{o}_1$ to $\mathbf{o}_i$ while the backward probability considers the part from $\mathbf{o}_{i+1}$ to $\mathbf{o}_n$.

Idea: We fill a $n \times |Q|$ matrix β such that

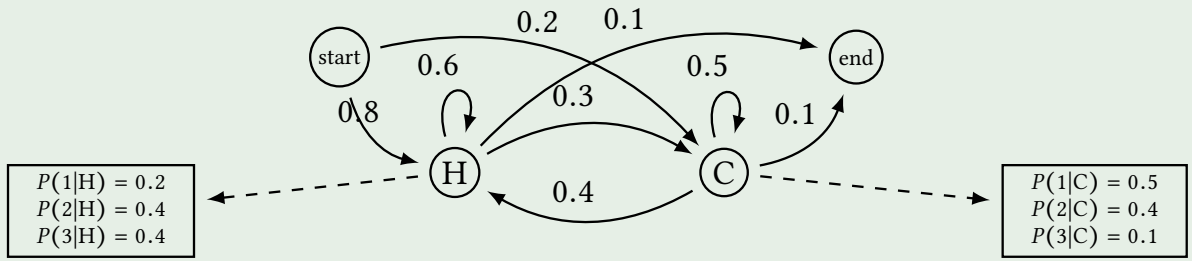
$$\beta_{ij} = P(\mathbf{o}_{i+1} \dots \mathbf{o}_n, \mathbf{q}_i = q_j)$$

Backward algorithm

1. $\beta_{nj} = a_{jF}$ for $1 \leq j \leq |Q|$
2. $\beta_{ij} = \sum_{k=1}^{|Q|} \beta_{i+1k} a_{jk} b_k(\mathbf{o}_{i+1})$ for $1 \leq i < n$ and $1 \leq j \leq |Q|$
3. $P(\mathbf{o}) = \sum_{k=1}^{|Q|} a_{0k} b_k(\mathbf{o}_1) \beta_{1k}$

Likelihood computation

Ice cream weather example continued



observed sequence 313:

H	$6.38 \cdot 10^{-3}$	$2.7 \cdot 10^{-2}$	0.1
C	$7.4 \cdot 10^{-3}$	$2.1 \cdot 10^{-2}$	0.1
	3	1	3

$P(313) = 2.17 \cdot 10^{-3}$

$$\beta_{nj} = a_{jF}$$
$$\beta_{ij} = \sum_{k=1}^{|Q|} \beta_{i+1k} a_{jk} b_k(\mathbf{o}_{i+1})$$
$$P(\mathbf{o}) = \sum_{k=1}^{|Q|} a_{0k} b_k(\mathbf{o}_1) \beta_{1k}$$

Decoding

Given an HMM and an observation $\mathbf{o} = \mathbf{o}_1 \dots \mathbf{o}_n$, determine the most probable state sequence $\mathbf{q} = \mathbf{q}_1 \dots \mathbf{q}_n \in Q^n$ for $\mathbf{o}$.

$$\arg\max_{\mathbf{q} \in Q^n} P(\mathbf{o}, \mathbf{q}) = \arg\max_{\mathbf{q} \in Q^n} P(\mathbf{o}|\mathbf{q}) P(\mathbf{q})$$

and with our independence assumptions

$$\arg\max_{\mathbf{q} \in Q^n} P(\mathbf{o}, \mathbf{q}) = \arg\max_{\mathbf{q} \in Q^n} \prod_{i=1}^{n} P(\mathbf{o}_i|\mathbf{q}_i) \prod_{i=1}^{n} P(\mathbf{q}_i|\mathbf{q}_{i-1})$$

Decoding

For the computation, we use the **viterbi** algorithm, which is very similar to the forward algorithm except that, instead of computing sums, we compute maxs.

We fill a $n \times |Q|$ matrix v such that

$$v_{ij} = \max_{\mathbf{q} \in Q^{i-1}} P(\mathbf{o}_1 \dots \mathbf{o}_i, \mathbf{q}_1 \dots \mathbf{q}_{i-1}, \mathbf{q}_i = q_j)$$

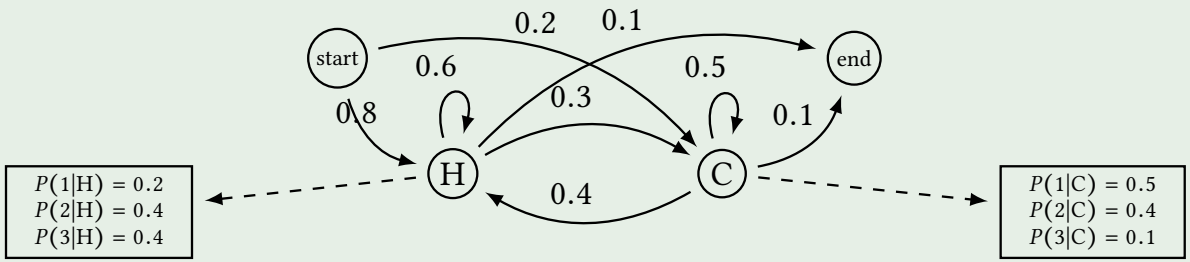
Viterbi algorithm

1. $v_{1j} = a_{0j} b_j(\mathbf{o}_1)$ for $1 \leq j \leq |Q|$
2. $v_{ij} = \max_{1 \leq k \leq |Q|} v_{i-1k} a_{kj} b_j(\mathbf{o}_i)$ for $1 \leq i \leq n$ and $1 \leq j \leq |Q|$
3. $\max_{\mathbf{q} \in Q^n} P(\mathbf{o}, \mathbf{q}) = \max_{1 \leq k \leq |Q|} v_{nk} a_{kF}$

In addition, in order to retreive the best state sequence, each entry v_{ij} is equipped with a backpointer to the state that has lead to the maximum.

Decoding

Ice cream weather example continued



output sequence 313:

H	$32 \cdot 10^{-2}$, start	$384 \cdot 10^{-4}$, H	$9216 \cdot 10^{-6}$, H
C	$2 \cdot 10^{-2}$, start	$480 \cdot 10^{-4}$, H	$24 \cdot 10^{-4}$, C
	3	1	3

$v_{1j} = a_{0j} b_j(\mathbf{o}_1)$ for $1 \leq j \leq |Q|$
$v_{ij} = \max_{1 \leq k \leq |Q|} v_{i-1k} a_{kj} b_j(\mathbf{o}_i)$ for $1 \leq i \leq n$ and $1 \leq j \leq |Q|$
$\max_{\mathbf{q} \in Q^n} P(\mathbf{o}, \mathbf{q}) = \max_{1 \leq k \leq |Q|} v_{nk} a_{kF}$

most probable weather sequence is HHH with probability $9216 \cdot 10^{-7}$

Training

Supervised HMM Training: Given a sequence of observations $\mathbf{o}$ and a corresponding sequence of states $\mathbf{q}$, learn the parameters A and B of an HMM.
MLE via the counts of $q_i q_j$ sequences and of pairs q_i, o_j of states and observations in our training data, eventually with some smoothing.

More challenging: Unsupervised HMM Training: Given an observation sequence $\mathbf{o}$ and a state set Q, estimate A and B.

Example

- ice cream – weather case: we have a sequence of observations $\mathbf{o} \in \{1, 2, 3\}^n$ and we know that the possible states are $Q = \{H, C\}$.
- POS tagging: we have a sequence of words $\mathbf{o} \in \{w_1, w_2, \ldots, w_{|V|}\}^n$ and we know that the possible POS tags (= states) are $Q = \{NN, NNS, VBD, IN, \ldots\}$.

Task: estimate A and B of the corresponding HMMs.

Training

We use the **forward-backward** or **Baum-Welch** algorithm (Baum, 1972), a special case of the **EM** algorithm (Dempster et al., 1977).

Underlying ideas:

- We estimate parameters iteratively: we start with some parameters and use the estimated probabilities to derive better and better parameters.
- We get our estimates by computing forward probabilities and then dividing the probability mass among all the different state sequences that have contributed to a forward probability.
- Put differently, in each iteration loop, we count all possible state sequences for the given observation sequence. But, instead of counting each of them once, we use their probabilities according to the most recent parameters as counts. We perform some kind of frequency-based MLE with these **weighted** or **fractional** counts.

References

Baum, L. E. 1972. An inequality and ssociated maximization technique in statistical estimation for jprobabilistic functions of markov processes. In O. Shisha (ed.), *Inequalities iii: Proceedigns of the 3rd symposiom on inequalities*, 1–8. University of California, Los Angeles: Academic Press.

Dempster, A. P., N. M. Laird & D. B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society* 39(1). 1–21.

Eisner, Jason. 2002. An interactive spreadsheet for teaching the forward-backward algorithm. In *Proceedings of the acl workshop on effective tools and methodologies for teaching nlp and cl*, 10–18.

Jurafsky, Daniel & James H. Martin. 2015. Speech and language processing. an introduction to natural language processing, computational linguistics, and speech recognition. Draft of the 3rd edition.

Table of contents

Motivation

- Naive Bayes is a generative classifier assigning a single class to a single input.
- MaxEnt is a discriminative classifier assigning a single class to a single input.
- HMM is a generative sequence classifier assigning sequences of classes to sequences of input symbols.
- CRF is a discriminative sequence classifier assigning sequences of classes to sequences of input symbols.

	single classification	sequence classification
generative	naive Bayes	HMM
discriminative	MaxEnt	CRF

Motivation

- **Generative classifiers** compute the probability of a class y given an input x as

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} = \frac{P(x,y)}{P(x)}$$

- HMM is a generative classifier:

$$P(\mathbf{q}|\mathbf{o}) = \frac{P(\mathbf{o},\mathbf{q})}{P(\mathbf{o})}$$

  For the classification, we have to compute

$$\arg\max_{\mathbf{q}\in Q^n} P(\mathbf{o},\mathbf{q}) = \arg\max_{\mathbf{q}\in Q^n} P(\mathbf{o}|\mathbf{q})P(\mathbf{q})$$

- The computation of the joint probability $P(x,y)$ (here $P(\mathbf{o},\mathbf{q})$) is a complex task.
- In contrast, MaxEnt classifiers directly compute the conditional probability $P(y|x)$ that has to be maximized.

Motivation

The move from generative to discriminative sequence classification (HMM to CRF) has two advantages:

- We do not need to compute the joint probability any longer.
- The strong independence assumptions of HMMs can be relaxed since features in a discriminative approach can capture dependencies that are less local than the n-gram based featues of HMMs.
- Feature weights need not be probabilities, i.e., can have values lower than 0 or greater than 1.

Conditional Random Fields

Goal: determine the best sequence $\mathbf{y} \in C^n$ of classes, given an input sequence $\mathbf{x}$ of length n.

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in C^n} P(\mathbf{y}|\mathbf{x})$$

CRF Applications

Sample applications are:

- POS tagging Ratnaparkhi (1997)
- shallow parsing Sha & Pereira (2003)
- Named Entity Recognition (Stanford NER)[a] Finkel et al. (2005)
- language identification Samih & Maier (2016)

[a] `http://nlp.stanford.edu/software/CRF-NER.shtml`

Conditional Random Fields

The probability of a class sequence for an input sequence depends on features (so-called **potential functions**).

Features refer to the potential class of some input symbol $\mathbf{x}_i$ and to the classes of some other input symbols.

Features are usually indicator functions that will be weighted.

Sample features

$$t(\mathbf{y}_{i-1}, \mathbf{y}_i, \mathbf{x}, i) = \begin{cases} 1 \text{ if } \mathbf{x}_i = \text{“September”} \text{ and } \mathbf{y}_{i-1} = \text{IN} \text{ and } \mathbf{y}_i = \text{NNP} \\ 0 \text{ otherwise} \end{cases}$$

(taken from Wallach (2004))

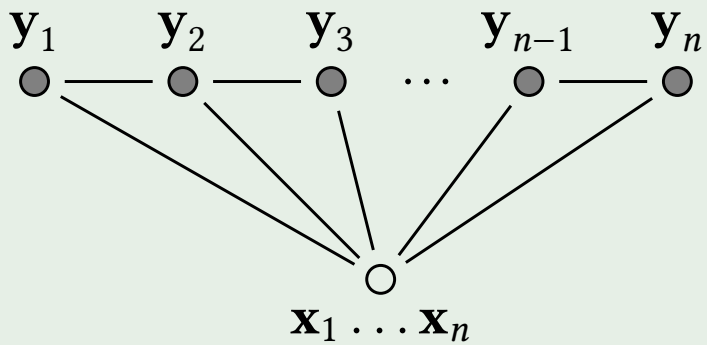
$$s(\mathbf{y}_i, \mathbf{x}, i) = \begin{cases} 1 \text{ if } \mathbf{x}_i = \text{“to”} \text{ and } \mathbf{y}_i = \text{TO} \\ 0 \text{ otherwise} \end{cases}$$

Conditional Random Fields

The dependencies that are expressed within the features can be captured in a graph.

CRF graph

If we have only transition features applying to $\mathbf{y}_{i-1}, \mathbf{y}_i, \mathbf{x}, i$ and state features applying to $\mathbf{y}_i, \mathbf{x}, i$, we get a chain-structured CRF:



Conditional Random Fields

In order to compute the probability of a class sequence for an input sequence, we

- extract the corresponding features,
- combine them linearly (= multiplying each by a weight and adding them up)
- and then applying a function to this linear combination, exactly as in the MaxEnt case.

In the following, we assume that we have only transition features and state features where the latter can also be considered a transition feature (that gives the same value for all preceding states). I.e., every features has the form $f(\mathbf{y}_{i-1}, \mathbf{y}_i, \mathbf{x}, i)$.

Conditional Random Fields

The f weights for a fixed f but for different input positions receive all the same weight. Therefore we can sum them up before weighting.

From class features to sequence features

$$F(\mathbf{y}, \mathbf{x}) = \sum_{i=1}^{n} f(\mathbf{y}_{i-1}, \mathbf{y}_i, \mathbf{x}, i)$$

Assume that we have features $f_1, \ldots, f_k$, which yield sequence features $F_1, \ldots, F_k$.

We weight these and apply them exactly as in the MaxEnt case:

Conditional class sequence probability

Let λ_j be the weight of features F_j.

$$P(\mathbf{y}|\mathbf{x}) = \frac{\mathrm{e}^{\sum_{i=1}^{k} \lambda_i F_i(\mathbf{y},\mathbf{x})}}{\sum_{\mathbf{y}' \in C^n} \mathrm{e}^{\sum_{i=1}^{k} \lambda_i F_i(\mathbf{y}',\mathbf{x})}}$$

Conditional Random Fields

CRF – POS tagging

Assume that we have POS tags Det, N, Adv, V and features

$$f_1(\mathbf{y}_{i-1}, \mathbf{y}_i, \mathbf{x}, i) = \begin{cases} 1 \text{ if } \mathbf{x}_i = \text{“chief”} \text{ and } \mathbf{y}_{i-1} = \text{Det} \text{ and } \mathbf{y}_i = \text{Adj} \\ 0 \text{ otherwise} \end{cases}$$

$$f_2(\mathbf{y}_{i-1}, \mathbf{y}_i, \mathbf{x}, i) = \begin{cases} 1 \text{ if } \mathbf{x}_i = \text{“chief”} \text{ and } \mathbf{y}_i = \text{N} \\ 0 \text{ otherwise} \end{cases}$$

$$f_3(\mathbf{y}_{i-1}, \mathbf{y}_i, \mathbf{x}, i) = \begin{cases} 1 \text{ if } \mathbf{x}_i = \text{“talks”} \text{ and } \mathbf{y}_{i-1} = \text{Det} \text{ and } \mathbf{y}_i = \text{N} \\ 0 \text{ otherwise} \end{cases}$$

$$f_4(\mathbf{y}_{i-1}, \mathbf{y}_i, \mathbf{x}, i) = \begin{cases} 1 \text{ if } \mathbf{x}_i = \text{“talks”} \text{ and } \mathbf{y}_{i-1} = \text{Adj} \text{ and } \mathbf{y}_i = \text{N} \\ 0 \text{ otherwise} \end{cases}$$

$$f_5(\mathbf{y}_{i-1}, \mathbf{y}_i, \mathbf{x}, i) = \begin{cases} 1 \text{ if } \mathbf{x}_i = \text{“talks”} \text{ and } \mathbf{y}_{i-1} = \text{N} \text{ and } \mathbf{y}_i = \text{V} \\ 0 \text{ otherwise} \end{cases}$$

$$f_6(\mathbf{y}_{i-1}, \mathbf{y}_i, \mathbf{x}, i) = \begin{cases} 1 \text{ if } \mathbf{x}_i = \text{“the”} \text{ and } \mathbf{y}_i = \text{Det} \\ 0 \text{ otherwise} \end{cases}$$

Weights: $\lambda_1 = 2$, $\lambda_2 = 5$, $\lambda_3 = 9$, $\lambda_4 = 8$, $\lambda_5 = 7$, $\lambda_6 = 20$.

Conditional Random Fields

CRF – POS tagging

Assume that we have a sequence "the chief talks". Which of the following probabilities is higher? $P(Det\ N\ V \mid \text{the chief talks})$ or $P(Det\ Adj\ N \mid \text{the chief talks})$?

Weighted feature sums for both:

1. *Det N V*: $20 + 5 + 7 = 32$
2. *Det Adj N*: $20 + 2 + 8 = 30$

Consequently, *Det N V* has a slightly higher probability.

(In real applications, we have of course many more features.)