Information Retrieval

WS 2017 / **2018**

Lecture 13, Tuesday January 30th, 2018

(POS-Tagging, Entity Recognition, Viterbi Algorithm)

Prof. Dr. Hannah Bast
Chair of Algorithms and Data Structures
Department of Computer Science
University of Freiburg

UNI FREIBURG

Overview of this lecture

- Organizational
  - Your experiences with ES12 — SPARQL to SQL
  - Exam registration — Deadline **TOMORROW**
- Content
  - Natural Language Processing — Brief intro + overview
Part-Of-Speech-Tagging
Entity Recognition
  - Hidden Markov Model — Definition + Example
  - Viterbi Algorithm — Recursive Formula
Application to POS-Tagging
  - **ES13:** POS-Tagging via the Viterbi Algorithm + optionally build an entity recognizer on top (easy if you have POS-tags)

Experiences with ES12

UNI FREIBURG

- Summary / excerpts
  - Many of you liked the topic / sheet / dataset a lot
  - "Programming helped to figure out how it's working"
  - "I quite enjoyed querying all kind of things with SPARQL because it is so easy to use"
  - "Nice to get to know databases and work with them so easily"
  - "The 'more than 100 results' constraints prohibited some more interesting / complex queries"
  - "Transl. would have been **so** nice to implement in Haskell"

Natural Language Processing 1/9

UNI FREIBURG

- Retrospection
  - So far in this course, we have seen a number of fundamental information retrieval techniques
  - What you have seen so far is sufficient to build a search engine with very decent functionality and result quality
  - Interestingly, none of our techniques so far actually tried to actually "understand" the language
  - Instead, they were mostly "statistical": e.g., keyword search just counts how often the words occur in the given corpus
  - It is one of the big mysteries of information retrieval (so far) that relatively "stupid" statistical methods work so well

    The same could be said about much of machine learning

Natural Language Processing 2/9

- Brief introduction to NLP
  - Natural Language Processing (NLP) is concerned with "understanding" language in a more "linguistic" sense
  - Here is a list of four fundamental NLP problems:

Part-Of-Speech Tagging:	For each word in a sentence, determine whether it is a noun, a verb, a preposition,
Named-Entity Recognition:	Recognize words in the text that refer to entities from a given knowledge base
Sentence Parsing:	Given a sentence, compute its grammatical structure (usually based on the POS tags)
Semantic Role Labeling:	Deeper semantic analysis of given sentence, e.g., the subject and object of the verb

Natural Language Processing 3/9

UNI FREIBURG

- Language
  - In this lecture, we will focus on the **English** language
  - The same techniques can be used just the same for other word-based languages
  - For character-based languages (like Chinese) different approaches are needed

Natural Language Processing 4/9

- POS-Tagging
  - Definition: given a sentence, assign to each word a tag that describes it's grammatical function in the sentence, e.g.:

NN	Noun	e.g.: time, world, …
IN	Preposition	e.g.: of, in, to, for, …
DT	Determiner	e.g.: the, a, this, …
JJ	Adjective	e.g.: new, other, same, …
NNP	Proper Noun	e.g.: Iraq, Virginia, Michael, …

  - One can make these tags more or less fine-grained

    For example, one can distinguish between **JJ** (normal adjective), **JJR** (comparative) and **JJS** (superlative)

  - For ES13, we will provide you with **36** POS- tags from the very widely used Penn Treebank Tag set

Natural Language Processing 5/9

- POS-Tagging, trivial (manual) solution
  - For each word in the (English) language, determine the POS tag with which it most frequently occurs in text

    For example, "the" is almost always a Determiner (DT)
  - However, there are many words, which can have different POS tags in different sentences, for example the word "like"

    PRP VBP DT NN
    I like this course

    EX VBZ DT NN IN DT NN
    There is no course like this one
  - Note that sometimes you need a deeper understanding of the words to compute the right POS tags, here is a classic:

    Time flies like an arrow

Natural Language Processing 6/9

- POS-Tagging, better solution
  - In the second half of the lecture, we will solve POS-Tagging with Hidden Markov Models and the Viterbi Algorithm

    This method gives an accuracy of 97% in practice

    This is close to the accuracy achieved by human experts (humans also make mistakes for such tasks)

    The naive method from the previous slide achieves an accuracy of about 90% ... and it fails badly for words which it did not see during training

    (it can only guess then, the standard is to guess NN)

Natural Language Processing 7/9

- Named-Entity Recognition and Disambiguation
  - **Named-Entity Recognition (NER):** given a sentence, recognize which words refer to a named entity
  - **Named-Entity Disambiguation (NED):** after NER, say which entity from a given knowledge base is referred to
  - Here is an example sentence:

    Armstrong was born in America and was an astronaut, engineer, and the first person to walk on the moon

    NER: just identify the underlined words

    NED (with Wikidata): Q1615, Q30, Q405

    Note: "abstract" entities like astronaut, engineer, person are typically not considered named entities

Natural Language Processing 8/9

- Simple Algorithm
  - **NER:** POS-tag the sentence and identify each sequence of words marked NNP as a named entity, for example

    DT NNP NNP RB VBZ NNP
    The Apache Project also supports Windows.
  - **NED:** take the most popular entity from the knowledge base with those words as name or synonym

    Q489709 (Apache Software Foundation)

    Q1406 (Microsoft Windows)

    Note that our dataset from ES5 (fuzzy search) already contained the names and entities of all Wikidata entities

Natural Language Processing 9/9

- Applications
  - Entity Recognition is enormously useful, two examples:

    Information Extraction

    Automatically extract knowledge base triples from text

    Identifying entities is the first (crucial) step towards this

    Entity Queries

    About 40% of web search queries refer to an entity

    Correctly identifying that entity is key for a good result

    More about that in the next (and last) lecture, where I will demonstrate some of our own search engines

Hidden Markov Model 1/4

- Markov Chain
  - Given a set S of **states**, e.g. S = {☺, 😐, ☹}
  - Given probabilities for the initial state

    Pr(☺) = **½**, Pr(😐) = **¼**, Pr(☹) = **¼**
  - Given transition probabilities between states

    Pr(☺ → ☺) = **½**, Pr(☺ → 😐) = **¼**, Pr(☺ → ☹) = **¼**

    Pr(😐 → ☺) = **¼**, Pr(😐 → 😐) = **½**, Pr(😐 → ☹) = **¼**

    Pr(☹ → ☺) = **¼**, Pr(😐 → 😐) = **¼**, Pr(☹ → ☹) = **½**
  - Each sequence of states now has a probability

    Pr(☺ → ☺ → ☺ → ☺) = ½ · ½ · ½ · ½ = **1/16**

    Pr(☺ → ☹ → ☺ → ☹) = ½ · ¼ · ¼ · ¼ = **1/128**

Hidden Markov Model 2/4

- Markov Chain, some remarks
  - The probability distribution for the current state depends only on the previous state, not on the history before that

    This is called the **Markov property**
  - The transition probabilities form an **|S| x |S|** matrix, which is **row-stochastic** = the sum of each row is **1**

    Note: there is no reason why column sums should be 1
  - Typical question in Markov Chain theory (but not in this talk) is, whether the chain has a **stationary distribution**

    = there is a probability distribution p_1, …, $p_{|S|}$ over the states such that, wherever you start, eventually each state s will be visited with probability approaching p_s

Hidden Markov Model 3/4

- Hidden Markov Model
  - Set **H** of hidden states, and set **S** of observable states

    hidden states: slept well (h^+), slept badly (h^-)

    observable states: ☺, 😐, ☹
  - We assume a Markov Model for the hidden states

    $Pr(h^+) = 3/4$ $Pr(h^-) = 1/4$

    $Pr(h^+ \rightarrow h^+) = 3/4$, $Pr(h^+ \rightarrow h^-) = 1/4$

    $Pr(h^- \rightarrow h^+) = 1/3$, $Pr(h^- \rightarrow h^-) = 2/3$
  - Observed states are conditional on the hidden states

    $Pr(☺ \mid h^+) = 1/2$, $Pr(😐 \mid h^+) = 1/4$, $Pr(☹ \mid h^+) = 1/4$

    $Pr(☺ \mid h^-) = 0$, $Pr(😐 \mid h^-) = 1/2$, $Pr(☹ \mid h^-) = 1/2$

Hidden Markov Model 4/4

- How to find the hidden states
  - Given a HMM and a sequence of **o**bserved states

    ☺ → ☺ → ☺ → 😐 → ☹ → 😐 → ☺ → ☺

  - Find the most likely sequence of **h**idden states

    $h^+ \rightarrow h^+ \rightarrow h^+ \rightarrow h^- \rightarrow h^- \rightarrow h^- \rightarrow h^+ \rightarrow h^+$

  - Formally, given sequence of observations $o_1, \ldots, o_n$ find sequence of hidden states $h_1, \ldots, h_n$ such that the probability of observing these states is maximized

    $Pr(o_1, \ldots, o_n, h_1, \ldots, h_n)$

    Note: this is again the maximum likelihood (MLE) principle, as discussed in Lecture 11 (Naive Bayes)

Viterbi Algorithm 1/14

$Pr(h_1 \mid h_0) := Pr(h_1)$
probability that
initial state = h_1

- Some preparation
  - Let us first use the HMM definition to split up the probability in question into simpler terms

    Markov property

    Bayes theorem

    $Pr(o_1, \ldots, o_n, h_1, \ldots, h_n)$

    $= Pr(o_1, h_1) \cdot Pr(o_2, h_2 \mid o_1, h_1) \cdot Pr(o_3, h_3 \mid o_2, h_2) \cdot \ldots$

    $= Pr(h_1) \cdot Pr(o_1 \mid h_1) \cdot Pr(h_2 \mid h_1) \cdot Pr(o_2 \mid h_2) \cdot \ldots$

    $= \prod_{i=1..n} Pr(h_i \mid h_{i-1}) \cdot Pr(o_i \mid h_i)$

  - So our goal is to find $h_1, \ldots, h_n$ which achieve

    $\max_{h_1,\ldots,h_n} \{ \prod_{i=1..n} Pr(h_i \mid h_{i-1}) \cdot Pr(o_i \mid h_i) \}$

Viterbi Algorithm 2/14

- Trivial approach

  So our goal is to find $h_1, \ldots, h_n$ which achieve

  $\max_{h_1,\ldots,h_n} \{ \prod_{i=1..n} Pr(h_i \mid h_{i-1}) \cdot Pr(o_i \mid h_i) \}$

  Let $H = \{ H_1, \ldots, H_k \}$ the set of possible states

  We could simply try out all possible assignments of the $h_1, \ldots, h_n$ to one of H each

  $\underbrace{k \cdot \ldots \cdot k}_{n \text{ times}} = k^n$

  How many such assignments are there?

  Infeasible, when n or k are not small

Viterbi Algorithm 3/14

- Recursive formula, first attempt

$\mathbf{P_n} := \max_{h_1,\ldots,h_n} \{ \prod_{i=1..n} \Pr(h_i \mid h_{i-1}) \cdot \Pr(o_i \mid h_i) \}$

Can we express $\mathbf{P_n}$ recursively via $\mathbf{P_{n-1}}$ and so on?

$\mathbf{P_n} = \max_{h_1,\ldots,h_n} \{ \prod_{i=1..n} \Pr(h_i \mid h_{i-1}) \cdot \Pr(o_i \mid h_i) \}$

$= \max_{h_1,\ldots,h_n} \{ \Pr(h_n \mid h_{n-1}) \cdot \Pr(o_n \mid h_n) \cdot \prod_{i=1..n-1} \Pr(h_i \mid h_{i-1}) \cdot \Pr(o_i \mid h_i) \}$

$\neq \max_{h_n} \{ \Pr(h_n \mid h_{n-1}) \cdot \Pr(o_n \mid h_n) \} \cdot \max_{h_1,\ldots,h_{n-1}} \{ \prod_{i=1..n-1} \Pr(h_i \mid h_{i-1}) \cdot \Pr(o_i \mid h_i) \}$

$= \max_{h_n} \{ \Pr(h_n \mid h_{n-1}) \cdot \Pr(o_n \mid h_n) \} \cdot \mathbf{P_{n-1}}$

This is terribly WRONG !

It doesn't even make sense (because of the h_{n-1})

Viterbi Algorithm 4/14

We assume that
$f(x), g(x) \geq 0$ for all x

- Some basic properties of max, True or False?

$\max_x \{ f(x) \cdot g(x) \} = \max_x \{ f(x) \} \cdot \max_x \{ g(x) \}$

False: only equal if f and g attain their maximum at the same value of x (which is not the case in general)

$\max_{x,y} \{ f(x) \cdot g(y) \} = \max_x \{ f(x) \} \cdot \max_y \{ g(y) \}$

True: f and g can be maximized independently, and a product of two independent factors is maximal if each factor is maximal

It's a good math exercise (and exam preparation) to prove these formally or give counterexamples

Viterbi Algorithm 5/14

we assume all maxima exist and are finite

- Basic properties of max, True or False?

$\max_{x,y} \{ f(x, y) \cdot g(y) \} = \max_{x,y} \{ f(x, y) \} \cdot \max_{y} \{ g(y) \}$

False: the best y for f may be different from the best y for g

the result is a function of x

$\max_{x,y} \{ f(x, y) \} = \max_{x} \max_{y} \{ f(x, y) \} = \max_{y} \max_{x} \{ f(x, y) \}$

True: maximizing for two variables is equivalent to first maximizing for one and then for the other

$\max_{x,y} \{ f(x) \cdot g(x, y) \} = \max_{x} \{ f(x) \cdot \max_{y} \{ g(x, y) \} \}$

True: $\max_{x,y} = \max_{x} \max_{y}$ and f(x) is independent of y and so it can be "pulled out" of the maximization over y

Viterbi Algorithm 6/14

- Recursive formula, second attempt

$$\mathbf{P_n(h_n)} := \max_{h_1,\ldots,h_{n-1}} \prod_{i=1..n} \{ \Pr(h_i \mid h_{i-1}) \cdot \Pr(o_i \mid h_i) \}$$

We can express $P_n(h_n)$ recursively via $P_{n-1}(h_{n-1})$!

$$\begin{aligned}
\mathbf{P_n(h_n)} &= \max_{h_1,\ldots,h_{n-1}} \prod_{i=1..n} \{ \Pr(h_i \mid h_{i-1}) \cdot \Pr(o_i \mid h_i) \} \\
&= \max_{h_1,\ldots,h_{n-1}} \{ \Pr(h_n \mid h_{n-1}) \cdot \Pr(o_n \mid h_n) \\
&\qquad \cdot \prod_{i=1..n-1} \Pr(h_i \mid h_{i-1}) \cdot \Pr(o_i \mid h_i) \} \\
&= \Pr(o_n \mid h_n) \cdot \max_{h_{n-1}} \{ \Pr(h_n \mid h_{n-1}) \cdot \\
&\qquad \max_{h_1,\ldots,h_{n-2}} \prod_{i=1..n-1} \Pr(h_i \mid h_{i-1}) \cdot \Pr(o_i \mid h_i) \} \\
&= \Pr(o_n \mid h_n) \cdot \max_{h_{n-1}} \{ \Pr(h_n \mid h_{n-1}) \cdot \mathbf{P_{n-1}(h_{n-1})} \}
\end{aligned}$$

This is CORRECT !

Viterbi Algorithm 7/14

- Algorithm, initial step

  Let $\mathbf{o_1, \ldots, o_n}$ be the sequence of **n** observations

  Let $\mathbf{H = \{H_1, \ldots, H_k\}}$ be the set of **k** hidden states

  Understand: the $h_1, \ldots, h_n$ from the previous slides are variables for the n hidden states we are looking for

  Each h_i takes on a value from $\{ H_1, \ldots, H_k \}$, which one is best, is exactly what we are trying to figure out

  Compute $P_1(H_1) = Pr(H_1) \cdot Pr(o_1 \mid H_1)$

  Compute $P_1(H_2) = Pr(H_2) \cdot Pr(o_1 \mid H_2)$

  …

  Compute $P_1(H_k) = Pr(H_k) \cdot Pr(o_1 \mid H_k)$

Viterbi Algorithm 8/14

- Algorithm, following steps

  Compute $P_2(H_1)$, …, $P_2(H_k)$ from $P_1(H_1)$, …, $P_1(H_k)$

  Compute $P_3(H_1)$, …, $P_3(H_k)$ from $P_2(H_1)$, …, $P_2(H_k)$

  …

  Compute $P_n(H_1)$, …, $P_n(H_k)$ from $P_{n-1}(H_1)$, …, $P_{n-1}(H_k)$

  Finally, compute $P_n = \max \{ P_n(H_1), \ldots, P_n(H_k) \}$

  This gives us the desired maximum likelihood

  The h_1, …, h_n that achieve this maximum can be retrieved by remembering, in the computation of each $P_i(H_j)$, for which $P_{i-1}(H_{j'})$ the maximum was achieved

  Similar to the sequence of operations in L5 (edit distance)

Viterbi Algorithm 9/14

- Time Complexity
  - Let **n** = number of observations
  - Let **h** = number of hidden states
  - Then altogether **n · k** values are computed (one per observation and possible hidden state for that observation)
  - For each value, the maximum over **k** values (the possible previous states) has to be computed
  - The time complexity is therefore $\mathbf{\Theta(n \cdot k^2)}$
  - The sequence of hidden states can be recovered in an additional **Θ(n)** time

Viterbi Algorithm 10/14

- Application to POS-Tagging
  - Example sentence:

    time flies like an arrow
  - Observed states = the sequence of words

    o_1 = time, o_2 = flies, o_3 = like, o_4 = an, o_5 = arrow
  - Hidden states = the sequence of POS-tags

    h_1 = ?, h_2 = ?, h_3 = ?, h_4 = ?, h_5 = ?
  - Example tag set

    H = { Noun, Verb, Other }

    Note: the tag set for ES13 is much larger

Viterbi Algorithm 11/14

- Application to POS-Tagging, example
  - Transition probabilities and Pr(word | hidden state) estimated from a large corpus ... given for ES13

Initial probab.	
Noun	1/2
Verb	0
Other	1/2

	Noun	Verb	Other	END
Noun	1/3	1/3	0	1/3
Verb	1/3	0	1/3	1/3
Other	1/2	0	1/2	0

	an	arrow	bear	flies	like	to	time
Noun	0	1/5	2/5	1/5	0	0	1/5
Verb	0	0	1/5	2/5	1/5	0	1/5
Other	2/5	0	0	0	1/5	2/5	0

Viterbi Algorithm 12/14

$P_m(h_m) = Pr(o_m | h_m) \cdot \max_{h_{m-1}} \{ Pr(h_m | h_{m-1}) \cdot P_{m-1}(h_{m-1}) \}$

Initial probab.	
Noun	1/2
Verb	0
Other	1/2

	Noun	Verb	Other	END
Noun	1/3	1/3	0	1/3
Verb	1/3	0	1/3	1/3
Other	1/2	0	1/2	0

$P_1(h_1) = Pr(o_1 | h_1) \cdot Pr(h_1)$

h	o_1 = time	o_2 = flies	o_3 = like	o_4 = an	o_5 = arrow
Noun	1/5	1/5	0	0	1/5
Verb	1/5	2/5	1/5	0	0
Other	0	0	1/5	2/5	0

h_i	$P_1(h_1)$	$P_2(h_2)$	$P_3(h_3)$	$P_4(h_4)$	$P_5(h_5)$
Noun	$\frac{1}{2}\cdot\frac{1}{5}$	$\frac{1}{5}\cdot\frac{1}{3}\cdot\frac{1}{2}\cdot\frac{1}{5}=\frac{1}{150}$	0	0	$\frac{1}{5}\cdot\frac{1}{2}\cdot\frac{1}{75^2}$ MAX
Verb	0	$\frac{2}{5}\cdot\frac{1}{3}\cdot\frac{1}{2}\cdot\frac{1}{5}=\frac{1}{75}$	$\frac{1}{5}\cdot\frac{1}{3}\cdot\frac{1}{150}$	0	0
Other	0	0	$\frac{1}{5}\cdot\frac{1}{3}\cdot\frac{1}{75}$	$\frac{2}{5}\cdot\frac{1}{2}\cdot\frac{1}{15}\cdot\frac{1}{75}$	0
	N	V	O	$=\frac{1}{75^2}$ O	N

Viterbi Algorithm 13/14

- Smoothing
  - Like for Naive Bayes, words which have not been seen during training are a problem, because $Pr(o_i \mid h) = 0$
  - When computing POS tags for a sentence with **k** new words, we therefore replace all $Pr(o_i \mid h)$ by

    $(1 - \boldsymbol{\varepsilon}) \cdot Pr(o_i \mid h) + \boldsymbol{\varepsilon} / (\#\text{vocabulary} + \mathbf{k})$

    where #vocabulary is the size of the training vocabulary, and the o_i also include the k new words

    For ES13, take $\boldsymbol{\varepsilon = 0.01}$
  - That way, all $Pr(o_i \mid h) > 0$ and for each h, the $Pr(o_i \mid h)$ are still a probability distribution, that is, sum to 1

Viterbi Algorithm 14/14

- Remark
  - The Viterbi algorithm is an instance of the principle of **dynamic programming**
  - Other prominent instances of dynamic programming:

    Edit distance computation (Lecture 5)

    Shortest path computation (Dijkstra's Algorithm)
  - The Viterbi algorithm can also be seen as solving a shortest path problem on a special graph

References

- Wikipedia
  - http://en.wikipedia.org/wiki/Markov_chain
  - http://en.wikipedia.org/wiki/Hidden_Markov_model
  - http://en.wikipedia.org/wiki/Viterbi_algorithm
  - http://en.wikipedia.org/wiki/Part-of-speech_tagging
  - http://en.wikipedia.org/wiki/Andrey_Markov
  - http://en.wikipedia.org/wiki/Andrew_Viterbi