

Z Wikipedie, otevřené encyklopedie

Viterbiho algoritmus je [algoritmus dynamického programování](#) pro hledání/nalezení nejpravděpodobnější posloupnosti skrytých stavů – nazývané **Viterbiho cesta** – jehož výsledkem je posloupnost pozorovaných událostí, ~~pravděpodobnější~~ v kontextu [Markovových informačních zdrojů](#) a [skrytých Markovových modelů](#).

Pojmy „Viterbiho cesta“ a „Viterbiho algoritmus“ se používají i pro další podobné algoritmy dynamického programování, které hledají nejpravděpodobnější vysvětlení určitého pozorování. Například algoritmus [dynamického programování pro statistické posuvátky](#) nebo počítání hrázek v [markovském bezkontextovém odvození](#) nebo [textu](#), [řetězce](#), [stavu](#) se někdy nazývá „Viterbiho posuvání“.

Algoritmus našel Andrew Viterbi v roce 1961 po dešifrování [konvolučních kódů](#) na digitálních komunikačních linkách se řízením [1]. Od té doby se používá při dešifrování [konvolučních kódů](#) používaných v mobilních sítích [CDMA](#) a [GSM](#) (v běžících [telefonních modemech](#)). Po komunikaci se setkává s kódem číselní sondou vzdáleného vesmíru i v pozemních sítích podle standardu [802.11](#). Často se používá při [rozpoznávání](#) a [syntéze](#) řeči, v [počítačové lingvistice](#) pro [vyhledávání klíčových slov](#) a v [bioinformatici](#). Nejčastěji při [rozpoznávání](#) řeči se zvukový signál povídává za posloucháče posloupnost událostí, a [textový řetězec](#) je slyšen přímo zvukového signálu. Viterbiho algoritmus hledá nejpravděpodobnější řetězec textu k danému slyšovému signálu.

Algoritmus

Předpokládejme, že je dán [skrytý Markovov model](#) (HMM) se stavovým množstvem S a se zadanou řetězkou pozorování y_1, \dots, y_T (zadanou číselnou sítí). Cílem je najít řetězec stavů x_1, \dots, x_T , který má největší pravděpodobnost $p(x_1, \dots, x_T | y_1, \dots, y_T)$ (předpokládáme pravděpodobnosti). Potud posuzujeme výstupu při posloupnosti y_1, \dots, y_T , pak nejpravděpodobnější posloupnost stavů x_1, \dots, x_T , která produkuje pozorovaný výstup, t. e. dánou rekvizitními výrazy.^[2]

$$\begin{aligned} V_{1,k} &= \text{P}(y_1 \mid k) \cdot \pi_k \\ V_{t,k} &= \text{P}(y_t \mid k) \cdot \max_{x \in S} (a_{x,k} \cdot V_{t-1,x}) \end{aligned}$$

kde $V_{t,k}$ je pravděpodobnost nejpravděpodobnější posloupnosti stavů odpovědné za prvních t pozorování, jejíž koncový stav je k . Pro získání Viterbiho cesty lze používat zpětné ukazatele, které zachycují, jaký stav x byl použit ve druhé rovnici. Nechť $\text{Ptr}(k, t)$ je funkce, která vrací hodnotu x použitou pro výpočet $V_{t,k}$ pokud $t > 1$, nebo k pokud $t = 1$. Pak:

$$\begin{aligned} x_T &= \arg \max_{x \in S} (V_{T,x}) \\ x_{t-1} &= \text{Ptr}(x_t, t) \end{aligned}$$

(používáme standardní definici [arg max](#)).

Složitost tohoto algoritmu je $O(T \times |S|^2)$.

Pseudokód

Pokud je dán prostor pozorování $O = \{o_1, o_2, \dots, o_N\}$, stavový prostor $S = \{s_1, s_2, \dots, s_K\}$, posloupnost pozorování $Y = \{y_1, y_2, \dots, y_T\}$, matice přechodů A velikosti $K \times K$ tak, že A_{ij} obsahuje přechodovou

$$T_1[i, j] = \max_k (T_1[k, j-1] \cdot A_{ki} \cdot B_{iy_j}), \text{ a}$$

$$T_2[i, j] = \arg \max_k (T_1[k, j-1] \cdot A_{ki} \cdot B_{iy_j})$$

VSTUP: Prostor pozorování $O = \{o_1, o_2, \dots, o_N\}$, stavový prostor $S = \{s_1, s_2, \dots, s_m\}$

10

ani v case i, je o:

pozorov

```

T(S, S, π, Y, A, B): X
    state  $s_i$  do
         $\pi_i \leftarrow \pi_i \cdot B_i y_i$ 
         $\pi_i \leftarrow 0$ 
    T do
        h state  $s_j$  do
             $T_1[j, i] \leftarrow \max_k (T_1[k, i-1] \cdot A_{kj} \cdot B_{jy_i})$ 
             $T_2[j, i] \leftarrow \arg \max_k (T_1[k, i-1] \cdot A_{kj} \cdot B_{jy_i})$ 
        end for
    end for
     $(T_1[k, T])$ 
     $z_T \leftarrow \arg \max_k$ 
     $x_T \leftarrow s_{z_T}$ 
    for  $i \leftarrow T, T-1, \dots, 2$  do
         $z_i \leftarrow z_{T-i}$ 
         $x_i \leftarrow s_{z_i}$ 
    end for
    return X
end function

```

Příklad

Představte si lékaře, který má pečovat o ženu císaře trpící neustále se vracející nemocí. Projevy nemoci lze léčit; tato léčba je nepříjemná, ale nemocné uleví. Problém je, že lékař císařovnu nemůže sám vyšetřit, dostává pouze každý třetí den lísteček s informací, jak se císařovna cítí (výborně, slabě, na umření). Na základě těchto informací má lékař posoudit, zda je císařovna zdravá nebo nemocná a má být podrobena léčbě.

Lékař se domnívá, že zdravotní stav císařovny se chová jako diskrétní [Markovův řetězec](#). Situaci, kdy lékař nemůže přímo zkoumat zdravotní stav císařovny, lze popsat jako [skrytý Markovův model](#) (HMM).

[Python](#)

Markovově řetězci. V tomto příkladě je jenom 30% pravděpodobnost, že za tři dny bude císařovna nemocná, když je dnes zdravá. `emission_probability` reprezentuje pravděpodobnosti jednotlivých informací. Pokud je císařovna zdravá, je 50% pravděpodobnost, že se cítí výborně; pokud je nemocná, je 60% pravděpodobnost, že se cítí na umření.

Na obrázcích jsou použity názvy z původního anglického příkladu (skryté zdravotní stavы jsou Healthy = Zdravá, Fever = Nemocná; oznámené pocity jsou Dizzy = na umření, Cold = slabě, Normal = výborně)

Grafická reprezentace zadaného HMM

Grafická reprezentace zadaného HMM

Lékař dostal s postupně tři zprávy o tom, jak se císařovna cítí, první zpráva byla výborně, druhá slabě, třetí na umření a chce zjistit, jaká je nejpravděpodobnější posloupnost zdravotních stavů císařovny, která by vysvětlila tato pozorování? Odpověď poskytne Viterbiho algoritmus:

```
# Vizualizace Viterbiho algoritmu.

def print_dptable(V):
    print("      "),
    for i in range(len(V)): print("%7d" % i),
    print()

    for y in V[0].keys():
        print("%.5s: " % y),
        for t in range(len(V)):
            print("%.7s" % ("%.f" % V[t][y])),
        print()

def viterbi(obs, states, start_p, trans_p, emit_p):
    V = [{}]
    path = {}

    # Initialize base cases (t == 0)
    for y in states:
        V[0][y] = start_p[y] * emit_p[y][obs[0]]
        path[y] = [y]
```

```

# Run Viterbi for t > 0
for t in range(1,len(obs)):
    V.append({})
    newpath = {}

    for y in states:
        (prob, state) = max([(V[t-1][y0] * trans_p[y0][y] * emit_p[y][obs[t]]) for y0 in states])
        V[t][y] = prob
        newpath[y] = path[state] + [y]

    # Don't need to remember the old paths
    path = newpath

print_dptable(V)
(prob, state) = max([(V[len(obs) - 1][y], y) for y in states])
return (prob, path[state])

```

Argumenty funkce viterbi jsou: obs je posloupnost pozorování, např. ['výborně', 'slabě', 'na umření']; states je množina skrytých stavů; start_p je start pravděpodobnost; trans_p jsou přechodové pravděpodobnosti; a emit_p jsou výstupní pravděpodobnosti. Pro jednoduchost kódu předpokládáme, že posloupnost pozorování obs je neprázdná a že trans_p[i][j] a emit_p[i][j] jsou definované pro všechny stavy i,j.

v našem příkladě se aopředný Viterbiho algoritmus používá takto:

```

def example():
    return viterbi(observations,
                   states,
                   start_probability,
                   transition_probability,
                   emission_probability)
print(example())

```

To ukazuje, že pozorování ['výborně', 'slabě', 'na umření'] byla s největší pravděpodobností generována posloupností stavů ['Zdravá', 'Zdravá', 'Nemocná']. Jinými slovy, na základě pozorovaných dat byla císařovna s největší pravděpodobností při odeslání první a druhé zprávy

zdravá (poprvé se cítila výborně, podruhé slabě), a při odeslání třetí byla nemocná.

Funkci Viterbiho algoritmu lze vizualizovat pomocí [trellis diagramu](#). Viterbiho cesta je v zásadě nejkratší cesta tímto trellisem. Trellis pro příklad s císařovnou je níže; odpovídající Viterbiho cesta je tučně:

[Animace trellis diagramu Viterbiho algoritmu Po třetí informaci o stavu je nejpravděpodobnější cesta \['Zdravá', 'Zdravá', 'Nemocná'\]](#)

Animace trellis diagramu Viterbiho algoritmu. Po třetí informaci o stavu je nejpravděpodobnější cesta ['Zdravá' , 'Zdravá' , 'Nemocná']

Při implementaci Viterbiho algoritmu je nutné zmínit, že mnoho jazyků používá aritmetiku s [pohyblivou řádovou čárkou](#) – pokud jsou hodnoty pravděpodobností malé, může dojít k [podtečení výsledku](#). Obvyklá technika, jak se tomu vyhnout, je používat během celého výpočtu [logaritmus pravděpodobnosti](#), tatáž technika použitá v [Logarithmic Number System](#). Po skončení algoritmu lze získat správnou hodnotu pomocí [exponenciální funkce](#).

Rozšíření

Zobecnění Viterbiho algoritmu nazývané *max-sum algoritmus* (nebo *max-product algoritmus*) lze použít pro nalezení nejpravděpodobnějšího přiřazení všech nebo určitých podmnožinách [skrytých proměnných](#) ve velkém množství [grafických modelů](#), např. [bayesovské sítě](#), [Markov náhodná pole](#) a [podmíněná náhodná pole](#). Skryté proměnné musí být obecně propojeny nějakým způsobem na HMM, s omezeným počtem spojení mezi proměnnými a určitým typem lineární struktury mezi proměnnými. Obecný algoritmus využívá mechanismus [předávání zpráv](#) a v zásadě se podobá algoritmu [belief propagation](#) (který je zobecněním [forward-backward algoritmu](#)).

Pomocí algoritmu nazývaného [iterativní Viterbiho dekódování](#) lze najít podposloupnost pozorování, která vyhovuje nejlépe (v průměru) dané HMM. Tento algoritmus navrhl Qi Wang, etc.^[3] pro zpracování [turbo kódů](#). Iterativní Viterbi dekódování pracuje iterativně vyvoláním modifikovaného Viterbiho algoritmu, znova odhadnutím skóre pro výplňku při konvergenci.

Nedávno byl navržen alternativní algoritmus, [líny Viterbiho algoritmus](#)^[4]. Pro mnoho kódů používaných v praxi, při rozumném šumu, je [dekodér](#) používající líny Viterbiho algoritmus mnohem rychlejší než tradiční Viterbiho dekodér^[5]. Líny Viterbiho algoritmus neexpanduje uzly, dokud to není opravdu nutné, a obvykle vyžaduje mnohem méně výpočtů, aby došel ke stejnemu výsledku jako normální Viterbiho algoritmus – není ho však snadné hardwarově [paralelizovat](#).

Existuje rozšíření Viterbiho algoritmu, aby pracoval s deterministickým konečným automatem pro zlepšení rychlosti při stochastické konverzi písmen na fonémy^[6].

Související články

- [Baum-Welchův algoritmus](#)
- [Forward-backward algoritmus](#)
- [Forward algoritmus](#)
- [Samoopravný kód](#)
- [Viterbiho algoritmus s měkkým výstupem](#)
- [Viterbiho dekodér](#)
- [Markovův model i Skrytý Markovův model](#)
- [Part-of-speech tagging](#)

Literatura

- Viterbi AJ. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*. 1967, s. 260–269. [Dostupné online](#). doi:[10.1109/TIT.1967.1054010](https://doi.org/10.1109/TIT.1967.1054010).
(note: the Viterbi decoding algoritmus je described in section IV.)
Subscription required.
- Feldman J, Abou-Faycal I, Frigo M. A Fast Maximum-Likelihood Decoder for Convolutional Codes. *Vehicular Technology Conference*. 2002, s. 371–375. doi:[10.1109/VETECF.2002.1040367](https://doi.org/10.1109/VETECF.2002.1040367).
- Forney GD. The Viterbi algorithm. *Proceedings of the IEEE*. 1973, s. 268–278. [Dostupné online](#). doi:[10.1109/PROC.1973.9030](https://doi.org/10.1109/PROC.1973.9030). Subscription required.

- PRESS, WH; TEUKOLSKY, SA; VETTERLING, WT; FLANNERY, BP. *Numerical Recipes: The Art of Scientific Computing*. 3rd. vyd. [s.l.]: Cambridge University Press, 2007. [ISBN 978-0-521-88068-8](#). Kapitola [Section 16.2. Viterbi Decoding](#).
- Rabiner LR. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*. 1989, s. 257–286. [doi:10.1109/5.18626](#). (Describes the forward algorithmus and Viterbi algorithm for HMMs).
- Shinghal, R. a [Godfried T. Toussaint](#), "Experiments in text recognition with the modified Viterbi algoritmus," *IEEE Transactions on Pattern Analysis a Machine Intelligence*, Vol. PAMI-1, April 1979, pp. 184–193.
- Shinghal, R. a [Godfried T. Toussaint](#), "The sensitivity of the modified Viterbi algoritmus to the source statistics," *IEEE Transactions on Pattern Analysis a Machine Intelligence*, vol. PAMI-2, March 1980, pp. 181–185.

Reference

V tomto článku byl použit [překlad](#) textu z článku [Viterbi algorithm](#) na anglické Wikipedii.

1. ↑ 29 Apr 2005, G. David Forney Jr: [The Viterbi Algorithm: A Personal History](#)
2. ↑ Xing E, slide 11
3. ↑ Qi Wang, Lei Wei; Rodney A. Kennedy, Iterative Viterbi Decoding, Trellis Shaping, a Multilevel Structure for High-Rate Parity-Concated TCM. *IEEE TRANSACTIONS ON COMMUNICATIONS*. 2002, s. 48–55.
4. ↑ (December 2002) "A fast maximum-likelihood decoder for convolutional codes" (PDF) in *Vehicular Technology Conference*.: 371–375. [doi:10.1109/VETECF.2002.1040367](#).
5. ↑ (December 2002) "A fast maximum-likelihood decoder for convolutional codes" (PDF) in *Vehicular Technology Conference*. doi: [10.1109/VETECF.2002.1040367](#).
6. ↑ Luk, R.W.P., R.I. Damper. Computational complexity of a fast Viterbi decoding algoritmus for stochastic letter-phoneme transduction. *IEEE Trans. Speech a Audio Processing*. 1998, s. 217–225. [doi:10.1109/89.668816](#).

Implementace

- [C a Jazyk symbolických adres](#)
- [C](#)
- [C++](#)
- [C++ a Boost](#) autor: Antonio Gulli
- [C#](#)
- [F#](#)
- [Java](#)
- [Perl](#)
- [Prolog](#)
- [VHDL](#)

Externí odkazy

- Obrázky, zvuky či videa k tématu [Viterbiho algoritmus](#) na Wikimedia Commons
- [Implementace v jazyce Java, F#, Clojure, C# na Wikibooks](#)
- [Učební text](#) [nedostupný zdroj] o konvolučním kódování s Viterbiho dekódováním, autor: Chip Fleming
- [Historie Viterbiho algoritmu](#), autor: David Forney
- [Jemný úvod do dynamického programování a Viterbiho algoritmu](#)
- [Učební text o sadě nástrojů pro modelování skrytého Markovova modelu \(implementace v jazyce C\) který obsahuje popis Viterbiho algoritmu.](#)

https://cs.wikipedia.org/w/index.php?title=Viterbiho_algoritmus&oldid=25431057
Kategorie:

[Detekce a oprava chyb](#)
[Dynamické programování](#)
[Markovovy modely](#)

[Údržba:Články s dočasně použitou šablonou](#)
[Monitoring:Články přeložené z enwiki](#)
[Údržba:Články obsahující odkazy na nedostupná zdroje](#)

- [Monitoring:Autoritní kontrola s 0 identifikátory](#)

Hledání

Speciální:Hledání

Hledat

Viterbiho algoritmus

18 jazyků

[Přidat téma](#)