WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Afrikaans
Čeština
Deutsch
Español
فارسی
Français
Italiano
Қазақша
日本語
Polski
Русский
Svenska
Українська
中文
Edit links

Viterbi algorithm

From Wikipedia, the free encyclopedia

The **Viterbi algorithm** is a dynamic programming algorithm for finding the most likely sequence of hidden states – called the **Viterbi path** – that results in a sequence of observed events, especially in the context of Markov information sources and hidden Markov models.

The algorithm has found universal application in decoding the convolutional codes used in both CDMA and GSM digital cellular, dial-up modems, satellite, deep-space communications, and 802.11 wireless LANs. It is now also commonly used in speech recognition, speech synthesis, diarization,[1] keyword spotting, computational linguistics, and bioinformatics. For example, in speech-to-text (speech recognition), the acoustic signal is treated as the observed sequence of events, and a string of text is considered to be the "hidden cause" of the acoustic signal. The Viterbi algorithm finds the most likely string of text given the acoustic signal.

Contents [hide]

History [edit]

The Viterbi algorithm is named after Andrew Viterbi, who proposed it in 1967 as a decoding algorithm for convolutional codes over noisy digital communication links.[2] It has, however, a history of multiple invention, with at least seven independent discoveries, including those by Viterbi, Needleman and Wunsch, and Wagner and Fischer.[3]

"Viterbi (path, algorithm)" has become a standard term for the application of dynamic programming algorithms to maximization problems involving probabilities.[3] For example, in statistical parsing a dynamic programming algorithm can be used to discover the single most likely context-free derivation (parse) of a string, which is commonly called the "Viterbi parse".[4][5][6]

Algorithm [edit]

Suppose we are given a hidden Markov model (HMM) with state space S, initial probabilities π_i of being in state i and transition probabilities $a_{i,j}$ of transitioning from state i to state j. Say we observe outputs $y_1,\dots,y_T$. The most likely state sequence $x_1,\dots,x_T$ that produces the observations is given by the recurrence relations:[7]

$$\begin{array}{rcl} V_{1,k} &=& \mathrm{P}\big(y_1 \ | \ k \big) \cdot \pi_k \\ V_{t,k} &=& \max_{x \in S} \left(\mathrm{P}\big(y_t \ | \ k \big) \cdot a_{x,k} \cdot V_{t-1,x}\right) \end{array}$$

Here $V_{t,k}$ is the probability of the most probable state sequence responsible for the first t observations that have k as its final state. The Viterbi path can be retrieved by saving back pointers that remember which state x was used in the second equation. Let $\mathrm{Ptr}(k,t)$ be the function that returns the value of x used to compute $V_{t,k}$ if $t > 1$, or k if $t = 1$. Then:

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

Here we're using the standard definition of arg max.

The complexity of this algorithm is $O(T\times\left|{S}\right|^2)$.

Example [edit]

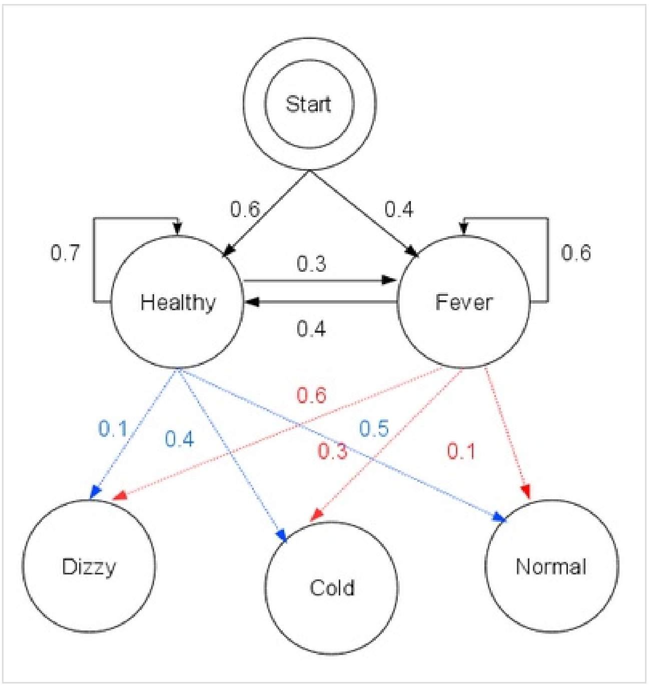
Consider a village where all villagers are either healthy or have a fever and only the village doctor can determine whether each has a fever. The doctor diagnoses fever by asking patients how they feel. The villagers may only answer that they feel normal, dizzy, or cold.

The doctor believes that the health condition of his patients operate as a discrete Markov chain. There are two states, "Healthy" and "Fever", but the doctor cannot observe them directly, they are *hidden* from him. On each day, there is a certain chance that the patient will tell the doctor he/she is "normal", "cold", or "dizzy", depending on her health condition.

The *observations* (normal, cold, dizzy) along with a *hidden* state (healthy, fever) form a hidden Markov model (HMM), and can be represented as follows in the Python programming language:

```
states = ('Healthy', 'Fever')

observations = ('normal', 'cold', 'dizzy')

start_probability = {'Healthy': 0.6, 'Fever': 0.4}

transition_probability = {
   'Healthy' : {'Healthy': 0.7, 'Fever': 0.3},
   'Fever' : {'Healthy': 0.4, 'Fever': 0.6}
   }

emission_probability = {
   'Healthy' : {'normal': 0.5, 'cold': 0.4, 'dizzy': 0.1},
   'Fever' : {'normal': 0.1, 'cold': 0.3, 'dizzy': 0.6}
   }
```

In this piece of code, `start_probability` represents the doctor's belief about which state the HMM is in when the patient first visits (all he knows is that the patient tends to be healthy). The particular probability distribution used here is not the equilibrium one, which is (given the transition probabilities) approximately `{'Healthy': 0.57, 'Fever': 0.43}`. The `transition_probability` represents the change of the health condition in the underlying Markov chain. In this example, there is only a 30% chance that tomorrow the patient will have a fever if he is healthy today. The `emission_probability` represents how likely the patient is to feel on each day. If he is healthy, there is a 50% chance that he feels normal; if he has a fever, there is a 60% chance that he feels dizzy.



The patient visits three days in a row and the doctor discovers that on the first day she feels normal, on the second day she feels cold, on the third day she feels dizzy. The doctor has a question: what is the most likely sequence of health conditions of the patient that would explain these observations? This is answered by the Viterbi algorithm.

```
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]],
y0) for y0 in states)
            V[t][y] = prob
            newpath[y] = path[state] + [y]

        # Don't need to remember the old paths
        path = newpath
    n = 0           # if only one element is observed max is sought in the
initialization values
    if len(obs) != 1:
        n = t
    print_dptable(V)
    (prob, state) = max((V[n][y], y) for y in states)
    return (prob, path[state])

# Don't study this, it just prints a table of the steps.
def print_dptable(V):
    s = "    " + " ".join(("%7d" % i) for i in range(len(V))) + "\n"
    for y in V[0]:
        s += "%.5s: " % y
        s += " ".join("%.7s" % ("%f" % v[y]) for v in V)
        s += "\n"
    print(s)
```

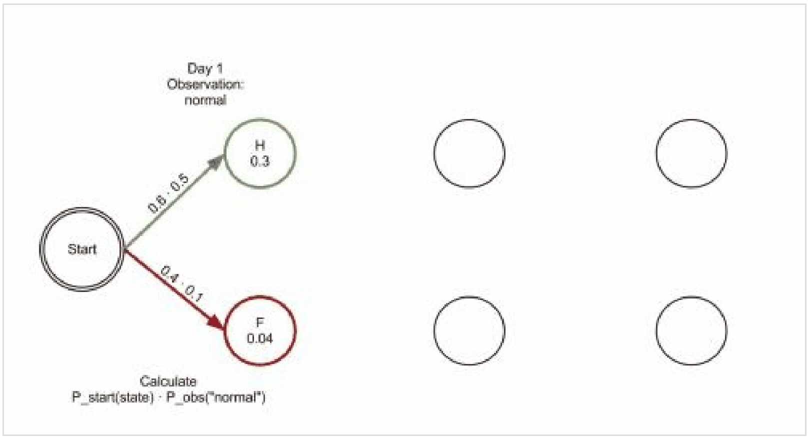
The function `viterbi` takes the following arguments: `obs` is the sequence of observations, e.g. `['normal', 'cold', 'dizzy']`; `states` is the set of hidden states; `start_p` is the start probability; `trans_p` are the transition probabilities; and `emit_p` are the emission probabilities. For simplicity of code, we assume that the observation sequence `obs` is non-empty and that `trans_p[i][j]` and `emit_p[i][j]` is defined for all states i,j.

In the running example, the forward/Viterbi algorithm is used as follows:

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

This reveals that the observations `['normal', 'cold', 'dizzy']` were most likely generated by states `['Healthy', 'Healthy', 'Fever']`. In other words, given the observed activities, the patient was most likely to have been healthy both on the first day when she felt normal as well as on the second day when she felt cold, and then she contracted a fever the third day.

The operation of Viterbi's algorithm can be visualized by means of a trellis diagram. The Viterbi path is essentially the shortest path through this trellis. The trellis for the clinic example is shown below; the corresponding Viterbi path is in bold:



When implementing Viterbi's algorithm, it should be noted that many languages use floating point arithmetic - as p is small, this may lead to underflow in the results. A common technique to avoid this is to take the logarithm of the probabilities and use it throughout the computation, the same technique used in the Logarithmic Number System. Once the algorithm has terminated, an accurate value can be obtained by performing the appropriate exponentiation.

Extensions [edit]

A generalization of the Viterbi algorithm, termed the *max-sum algorithm* (or *max-product algorithm*) can be used to find the most likely assignment of all or some subset of latent variables in a large number of graphical models, e.g. Bayesian networks, Markov random fields and conditional random fields. The latent variables need in general to be connected in a way somewhat similar to an HMM, with a limited number of connections between variables and some type of linear structure among the variables. The general algorithm involves *message passing* and is substantially similar to the belief propagation algorithm (which is the generalization of the forward-backward algorithm).

With the algorithm called iterative Viterbi decoding one can find the subsequence of an observation that matches best (on average) to a given HMM. This algorithm is proposed by Qi Wang et al.[8] to deal with turbo code. Iterative Viterbi decoding works by iteratively invoking a modified Viterbi algorithm, reestimating the score for a filler until convergence.

An alternative algorithm, the Lazy Viterbi algorithm, has been proposed recently.[9] For many codes of practical interest, under reasonable noise conditions, the lazy decoder (using Lazy Viterbi algorithm) is much faster than the original Viterbi decoder (using Viterbi algorithm). This algorithm works by not expanding any nodes until it really needs to, and usually manages to get away with doing a lot less work (in software) than the ordinary Viterbi algorithm for the same result - however, it is not so easy to parallelize in hardware.

Pseudocode [edit]

Given the observation space $O = \{o_1, o_2, \dots, o_N\}$, the state space $S = \{s_1, s_2, \dots, s_K\}$, a sequence of observations $Y = \{y_1, y_2, \dots, y_T\}$, transition matrix A of size $K \times K$ such that A_{ij} stores the transition probability of transiting from state s_i to state s_j, emission matrix B of size $K \times N$ such that B_{ij} stores the probability of observing o_j from state s_i, an array of initial probabilities π of size K such that π_i stores the probability that $x_1 == s_i$.We say a path $X = \{x_1, x_2, \dots, x_T\}$ is a sequence of states that generate the observations $Y = \{y_1, y_2, \dots, y_T\}$.

In this dynamic programming problem, we construct two 2-dimensional tables T_1, T_2 of size $K \times T$. Each element of T_1, $T_1[i,j]$, stores the probability of the most likely path so far $\hat{X} = \{\hat{x}_1, \hat{x}_2, \dots, \hat{x}_j\}$ with $\hat{x}_j = s_i$ that generates $Y = \{y_1, y_2, \dots, y_j\}$. Each element of T_2, $T_2[i,j]$, stores $\hat{x}_{j-1}$ of the most likely path so far $\hat{X} = \{\hat{x}_1, \hat{x}_2, \dots, \hat{x}_{j-1}, \hat{x}_j\}$ for $\forall j, 2 \le j \le T$

We fill entries of two tables $T_1[i,j], T_2[i,j]$ by increasing order of $K \cdot j + i$.

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

Note that B_{iy_j} does not need to appear in the latter expression, as it's constant with *i* and *j* and does not affect the argmax.

```
   INPUT:  The observation space $O = \{o_1, o_2, \dots, o_N\}$,
           the state space $S = \{s_1, s_2, \dots, s_K\}$,
           a sequence of observations  $Y = \{y_1, y_2, \dots, y_T\}$ such that $y_t == i$ if
the
             observation at time $t$ is $o_i$,
           transition matrix $A$ of size $K \cdot K$ such that $A_{ij}$ stores the transition
             probability of transiting from state $s_i$ to state $s_j$,
           emission matrix $B$ of size $K \cdot N$ such that $B_{ij}$ stores the probability
of
             observing $o_j$ from  state $s_i$,
           an array of initial probabilities $\pi$ of size $K$ such that $\pi_i$ stores the
probability
             that $x_1 == s_i$
   OUTPUT: The most likely hidden state sequence $X = \{x_1, x_2, \dots, x_T\}$
function VITERBI( O,  S,π,Y,A,B ) : X
    for each state $s_i$ do
        $T_1[i,1] \leftarrow \pi_i \cdot B_{iy_1}$
        $T_2[i,1] \leftarrow 0$
    end for
    for $i \leftarrow 2,3,\dots,T$ do
        for each 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
    $z_T \leftarrow \arg\max_k (T_1[k,T])$
    $x_T \leftarrow s_{z_T}$
    for $i \leftarrow T, T-1, \dots, 2$ do
        $z_{i-1} \leftarrow T_2[z_i, i]$
        $x_{i-1} \leftarrow s_{z_{i-1}}$
    end for
    return X
end function
```

See also [edit]

- Expectation–maximization algorithm
- Baum–Welch algorithm
- Forward-backward algorithm
- Forward algorithm
- Error-correcting code
- Soft output Viterbi algorithm
- Viterbi decoder
- Hidden Markov model
- Part-of-speech tagging

Notes [edit]

1. ^ Xavier Anguera et Al, "Speaker Diarization: A Review of Recent Research", retrieved 19. August 2010, IEEE TASLP
2. ^ 29 Apr 2005, G. David Forney Jr: The Viterbi Algorithm: A Personal History
3. ^ [a] [b] Daniel Jurafsky; James H. Martin. *Speech and Language Processing*. Pearson Education International. p. 246.
4. ^ Schmid, Helmut (2004). *Efficient parsing of highly ambiguous context-free grammars with bit vectors* (PDF). Proc. 20th Int'l Conf. on Computational Linguistics (COLING). doi:10.3115/1220355.1220379.
5. ^ Klein, Dan; Manning, Christopher D. (2003). *A* parsing: fast exact Viterbi parse selection* (PDF). Proc. 2003 Conf. of the North American Chapter of the Association for Computational Linguistics on Human Language Technology (NAACL). pp. 40–47. doi:10.3115/1073445.1073461.
6. ^ Stanke, M.; Keller, O.; Gunduz, I.; Hayes, A.; Waack, S.; Morgenstern, B. (2006). "AUGUSTUS: Ab initio prediction of alternative transcripts". *Nucleic Acids Research* **34**: W435. doi:10.1093/nar/gkl200.
7. ^ Xing E, slide 11
8. ^ Qi Wang; Lei Wei; Rodney A. Kennedy (2002). "Iterative Viterbi Decoding, Trellis Shaping,and Multilevel Structure for High-Rate Parity-Concatenated TCM". *IEEE Transactions on Communications* **50**: 48–55.

doi:10.1109/26.975743.

9. ^ *A fast maximum-likelihood decoder for convolutional codes* (PDF). Vehicular Technology Conference. December 2002. pp. 371–375. doi:10.1109/VETECF.2002.1040367.

References [edit]

- Viterbi AJ (April 1967). "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm". *IEEE Transactions on Information Theory* **13** (2): 260–269. doi:10.1109/TIT.1967.1054010. (note: the Viterbi decoding algorithm is described in section IV.) Subscription required.
- Feldman J, Abou-Faycal I, Frigo M (2002). "A Fast Maximum-Likelihood Decoder for Convolutional Codes". *Vehicular Technology Conference* **1**: 371–375. doi:10.1109/VETECF.2002.1040367.
- Forney GD (March 1973). "The Viterbi algorithm". *Proceedings of the IEEE* **61** (3): 268–278. doi:10.1109/PROC.1973.9030. Subscription required.
- Press, WH; Teukolsky, SA; Vetterling, WT; Flannery, BP (2007). "Section 16.2. Viterbi Decoding". *Numerical Recipes: The Art of Scientific Computing* (3rd ed.). New York: Cambridge University Press. ISBN 978-0-521-88068-8.
- Rabiner LR (February 1989). "A tutorial on hidden Markov models and selected applications in speech recognition". *Proceedings of the IEEE* **77** (2): 257–286. doi:10.1109/5.18626. (Describes the forward algorithm and Viterbi algorithm for HMMs).
- Shinghal, R. and Godfried T. Toussaint, "Experiments in text recognition with the modified Viterbi algorithm," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-l, April 1979, pp. 184–193.
- Shinghal, R. and Godfried T. Toussaint, "The sensitivity of the modified Viterbi algorithm to the source statistics," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-2, March 1980, pp. 181–185.

Implementations [edit]

- Susa signal processing framework provides the C++ implementation for Forward error correction codes and channel equalization here.
- C#
- Java
- Perl
- Prolog
- Haskell

External links [edit]

- Implementations in Java, F#, Clojure, C# on Wikibooks
- Tutorial on convolutional coding with viterbi decoding, by Chip Fleming
- The history of the Viterbi Algorithm, by David Forney
- A Gentle Introduction to Dynamic Programming and the Viterbi Algorithm
- A tutorial for a Hidden Markov Model toolkit (implemented in C) that contains a description of the Viterbi algorithm

Categories: Error detection and correction | Dynamic programming | Markov models