WIKIPEDIA
The Free Encyclopedia

Viterbi algorithm

The **Viterbi algorithm** is a dynamic programming algorithm that finds the most likely sequence of hidden events that would explain a sequence of observed events. The result of the algorithm is often called the **Viterbi path**. It is most commonly used with hidden Markov models (HMMs). For example, if a doctor observes a patient's symptoms over several days (the observed events), the Viterbi algorithm could determine the most probable sequence of underlying health conditions (the hidden events) that caused those symptoms.

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 also commonly used in speech recognition, speech synthesis, diarization,[1] keyword spotting, computational linguistics, and bioinformatics. For instance, in speech-to-text (speech recognition), the acoustic signal is the observed sequence, and a string of text is the "hidden cause" of that signal. The Viterbi algorithm finds the most likely string of text given the acoustic signal.

History

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] It was introduced to natural language processing as a method of part-of-speech tagging as early as 1987.

Viterbi path and *Viterbi algorithm* have become standard terms 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] Another application is in target tracking, where the track is computed that assigns a maximum likelihood to a sequence of observations.[7]

Algorithm

Given a hidden Markov model with a set of hidden states S and a sequence of T observations $o_0, o_1, \dots, o_{T-1}$, the Viterbi algorithm finds the most likely sequence of states that could have produced those observations. At each time step t, the algorithm solves the subproblem where only the observations up to o_t are considered.

Two matrices of size $T \times |S|$ are constructed:

- $P_{t,s}$ contains the maximum probability of ending up at state s at observation t, out of all possible sequences of states leading up to it.
- $Q_{t,s}$ tracks the previous state that was used before s in this maximum probability state sequence.

Let π_s and $a_{r,s}$ be the initial and transition probabilities respectively, and let $b_{s,o}$ be the probability of observing o at state s. Then the values of P are given by the recurrence relation[8]

$$P_{t,s} = \begin{cases} \pi_s \cdot b_{s,o_t} & \text{if } t = 0, \\ \max_{r \in S} \left(P_{t-1,r} \cdot a_{r,s} \cdot b_{s,o_t} \right) & \text{if } t > 0. \end{cases}$$

The formula for $Q_{t,s}$ is identical for $t > 0$, except that $\max$ is replaced with $\arg\max$, and $Q_{0,s} = 0$. The Viterbi path can be found by selecting the maximum of P at the final timestep, and following Q in reverse.

Pseudocode

```
function Viterbi(states, init, trans, emit, obs) is
    input states: S hidden states
    input init: initial probabilities of each state
    input trans: S × S transition matrix
    input emit: S × O emission matrix
    input obs: sequence of T observations

    prob ← T × S matrix of zeroes
    prev ← empty T × S matrix
    for each state s in states do
        prob[0][s] = init[s] * emit[s][obs[0]]

    for t = 1 to T - 1 inclusive do // t = 0 has been dealt with already
        for each state s in states do
            for each state r in states do
                new_prob ← prob[t - 1][r] * trans[r][s] * emit[s][obs[t]]
                if new_prob > prob[t][s] then
                    prob[t][s] ← new_prob
                    prev[t][s] ← r

    path ← empty array of length T
    path[T - 1] ← the state s with maximum prob[T - 1][s]
    for t = T - 2 to 0 inclusive do
        path[t] ← prev[t + 1][path[t + 1]]

    return path
end
```

The time complexity of the algorithm is $O(T \times |S|^2)$. If it is known which state transitions have non-zero probability, an improved bound can be found by iterating over only those r which link to s in the inner loop. Then using amortized analysis one can show that the complexity is $O(T \times (|S| + |E|))$, where E is the number of edges in the graph, i.e. the number of non-zero entries in the transition matrix.

Example

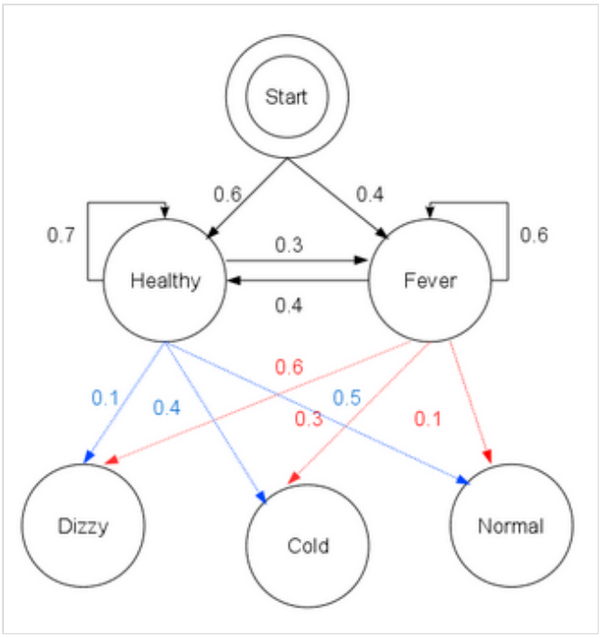
A doctor wishes to determine whether patients are healthy or have a fever. The only information the doctor can obtain is by asking patients how they feel. The patients may report that they either feel normal, dizzy, or cold.

It is believed that the health condition of the patients operates as a discrete Markov chain. There are two states, "healthy" and "fever", but the doctor cannot observe them directly; they are *hidden* from the doctor. On each day, the chance that a patient tells the doctor "I feel normal", "I feel cold", or "I feel dizzy", depends only on the patient's health condition on that day.

The *observations* (normal, cold, dizzy) along with the *hidden* states (healthy, fever) form a hidden Markov model (HMM). From past experience, the probabilities of this model have been estimated as:

```
init = {"Healthy": 0.6, "Fever": 0.4}
trans = {
    "Healthy": {"Healthy": 0.7, "Fever": 0.3},
    "Fever": {"Healthy": 0.4, "Fever": 0.6},
}
emit = {
    "Healthy": {"normal": 0.5, "cold": 0.4, "dizzy": 0.1},
    "Fever": {"normal": 0.1, "cold": 0.3, "dizzy": 0.6},
}
```

In this code, `init` represents the doctor's belief about how likely the patient is to be healthy initially. Note that the particular probability distribution used here is not the equilibrium one, which would be `{'Healthy': 0.57, 'Fever': 0.43}` according to the transition probabilities. The transition probabilities `trans` represent the change of health condition in the underlying Markov chain. In this example, a patient who is healthy today has only a 30% chance of having a fever tomorrow. The emission probabilities `emit` represent how likely each possible observation (normal, cold, or dizzy) is, given the underlying condition (healthy or fever). A patient who is healthy has a 50% chance of feeling normal; one who has a fever has a 60% chance of feeling dizzy.



Graphical representation of the given HMM

A particular patient visits three days in a row, and reports feeling normal on the first day, cold on the second day, and dizzy on the third day.

Firstly, the probabilities of being healthy or having a fever on the first day are calculated. The probability that a patient will be healthy on the first day and report feeling normal is $0.6 \times 0.5 = 0.3$. Similarly, the probability that a patient will have a fever on the first day and report feeling normal is $0.4 \times 0.1 = 0.04$.

The probabilities for each of the following days can be calculated from the previous day directly. For example, the highest chance of being healthy on the second day and reporting to be cold, following reporting being normal on the first day, is the maximum of $0.3 \times 0.7 \times 0.4 = 0.084$ and $0.04 \times 0.4 \times 0.4 = 0.0064$. This suggests it is more likely that the patient was healthy for both of those days, rather than having a fever and recovering.

The rest of the probabilities are summarised in the following table:

Day	**1**	**2**	**3**
Observation	Normal	Cold	Dizzy
Healthy	**0.3**	**0.084**	0.00588
Fever	0.04	0.027	**0.01512**

From the table, it can be seen that the patient most likely had a fever on the third day. Furthermore, there exists a sequence of states ending on "fever", of which the probability of producing the given observations is 0.01512. This sequence is precisely (healthy, healthy, fever), which can be found be tracing back which states were used when calculating the maxima (which happens to be the best guess from each day but will not always be). In other words, given the observed activities, the patient was most likely to have been healthy on the first day and also on the second day (despite feeling cold that day), and only to have contracted a fever on 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.

Extensions

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 a hidden Markov model (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 an algorithm called iterative Viterbi decoding, one can find the subsequence of an observation that matches best (on average) to a given hidden Markov model. This algorithm is proposed by Qi Wang et al. to deal with turbo code.[9] 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.[10] For many applications 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). While the original Viterbi algorithm calculates every node in the trellis of possible outcomes, the Lazy Viterbi algorithm maintains a prioritized list of nodes to evaluate in order, and the number of calculations required is typically fewer (and never more) than the ordinary Viterbi algorithm for the same result. However, it is not so easy to parallelize in hardware.

Soft output Viterbi algorithm

The **soft output Viterbi algorithm** (**SOVA**) is a variant of the classical Viterbi algorithm.

SOVA differs from the classical Viterbi algorithm in that it uses a modified path metric which takes into account the *a priori probabilities* of the input symbols, and produces a *soft* output indicating the *reliability* of the decision.

The first step in the SOVA is the selection of the survivor path, passing through one unique node at each time instant, *t*. Since each node has 2 branches converging at it (with one branch being chosen to form the *Survivor Path*, and the other being discarded), the difference in the branch metrics (or *cost*) between the chosen and discarded branches indicate the *amount of error* in the choice.

This *cost* is accumulated over the entire sliding window (usually equals *at least* five constraint lengths), to indicate the *soft output* measure of reliability of the *hard bit decision* of the Viterbi algorithm.

See also

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

References

1. Xavier Anguera et al., "Speaker Diarization: A Review of Recent Research" (http://www1.icsi.berkeley.edu/~vinyals/Files/taslp2011a.pdf) Archived (https://web.archive.org/web/20160512200056/http://www1.icsi.berkeley.edu/~vinyals/Files/taslp2011a.pdf) 2016-05-12 at the Wayback Machine, retrieved 19. August 2010, IEEE TASLP
2. 29 Apr 2005, G. David Forney Jr: The Viterbi Algorithm: A Personal History (https://arxiv.org/abs/cs/0504020v2)
3. 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* (http://www.aclweb.org/anthology/C/C04/C04-1024.pdf) (PDF). Proc. 20th Int'l Conf. on Computational Linguistics (COLING). doi:10.3115/1220355.1220379 (https://doi.org/10.3115%2F1220355.1220379).
5. Klein, Dan; Manning, Christopher D. (2003). *A* parsing: fast exact Viterbi parse selection* (http://ilpubs.stanford.edu:8090/532/1/2002-16.pdf) (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 (https://doi.org/10.3115%2F1073445.1073461).
6. Stanke, M.; Keller, O.; Gunduz, I.; Hayes, A.; Waack, S.; Morgenstern, B. (2006). "AUGUSTUS: Ab initio prediction of alternative transcripts" (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1538822). *Nucleic Acids Research*. **34** (Web Server issue): W435 – W439. doi:10.1093/nar/gkl200 (https://doi.org/10.1093%2Fnar%2Fgkl200). PMC 1538822 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1538822). PMID 16845043 (https://pubmed.ncbi.nlm.nih.gov/16845043).
7. Quach, T.; Farooq, M. (1994). "Maximum Likelihood Track Formation with the Viterbi Algorithm". *Proceedings of 33rd IEEE Conference on Decision and Control*. Vol. 1. pp. 271–276. doi:10.1109/CDC.1994.410918 (https://doi.org/10.1109%2FCDC.1994.410918).
8. Xing E, slide 11.
9. 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 (https://doi.org/10.1109%2F26.975743).
10. *A fast maximum-likelihood decoder for convolutional codes* (http://people.csail.mit.edu/jonfeld/pubs/lazyviterbi.pdf) (PDF). Vehicular Technology Conference (http://www.ieeevtc.org/). December 2002. pp. 371–375. doi:10.1109/VETECF.2002.1040367 (https://doi.org/10.1109%2FVETECF.2002.1040367).

General references

- 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 (https://doi.org/10.1109%2FTIT.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". *Proceedings IEEE 56th Vehicular Technology Conference*. Vol. 1. pp. 371–375. CiteSeerX 10.1.1.114.1314 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.114.1314). doi:10.1109/VETECF.2002.1040367 (https://doi.org/10.1109%2FVETECF.2002.1040367). ISBN 978-0-7803-7467-6. S2CID 9783963 (https://api.semanticscholar.org/CorpusID:9783963).
- Forney GD (March 1973). "The Viterbi algorithm". *Proceedings of the IEEE*. **61** (3): 268–278. doi:10.1109/PROC.1973.9030 (https://doi.org/10.1109%2FPROC.1973.9030). Subscription required.
- Press, WH; Teukolsky, SA; Vetterling, WT; Flannery, BP (2007). "Section 16.2. Viterbi Decoding" (https://web.archive.org/web/20110811154417/http://apps.nrbook.com/empanel/index.html#pg=850). *Numerical Recipes: The Art of Scientific Computing* (3rd ed.). New York: Cambridge University Press. ISBN 978-0-521-88068-8. Archived from the original (http://apps.nrbook.com/empanel/index.html#pg=850) on 2011-08-11. Retrieved 2011-08-17.
- Rabiner LR (February 1989). "A tutorial on hidden Markov models and selected applications in speech recognition". *Proceedings of the IEEE*. **77** (2): 257–286. CiteSeerX 10.1.1.381.3454 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.381.3454). doi:10.1109/5.18626 (https://doi.org/10.1109%2F5.18626). S2CID 13618539 (https://api.semanticscholar.org/CorpusID:13618539). (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.

External links

- Implementations in Java, F#, Clojure, C# on Wikibooks
- Tutorial (http://pl91.ddns.net/viterbi/tutorial.html) on convolutional coding with viterbi decoding, by Chip Fleming
- A tutorial for a Hidden Markov Model toolkit (implemented in C) that contains a description of the Viterbi algorithm (http://www.kanungo.com/software/hmmtut.pdf)
- Viterbi algorithm (http://www.scholarpedia.org/article/Viterbi_algorithm) by Dr. Andrew J. Viterbi (scholarpedia.org).

Implementations

- Mathematica (https://reference.wolfram.com/language/ref/FindHiddenMarkovStates.html) has an implementation as part of its support for stochastic processes
- Susa (http://libsusa.org/) signal processing framework provides the C++ implementation for Forward error correction codes and channel equalization here (https://github.com/libsusa/susa/blob/master/inc/susa/channel.h).
- C++ (https://github.com/xukmin/viterbi)
- C# (http://pcarvalho.com/forward_viterbi/)
- Java (http://www.cs.stonybrook.edu/~pfodor/viterbi/Viterbi.java) Archived (https://web.archive.org/web/20140504055101/http://www.cs.stonybrook.edu/~pfodor/viterbi/Viterbi.java) 2014-05-04 at the Wayback Machine
- Java 8 (https://adrianulbona.github.io/hmm/)
- Julia (HMMBase.jl) (https://juliahub.com/ui/Packages/HMMBase/8HxY5/)
- Perl (https://metacpan.org/module/Algorithm::Viterbi)
- Prolog (http://www.cs.stonybrook.edu/~pfodor/viterbi/viterbi.P) Archived (https://web.archive.org/web/20120502010115/http://www.cs.stonybrook.edu/~pfodor/viterbi/viterbi.P) 2012-05-02 at the Wayback Machine
- Haskell (https://hackage.haskell.org/package/hmm-0.2.1.1/docs/src/Data-HMM.html#viterbi)

- Go (https://github.com/nyxtom/viterbi)
- SFIHMM (http://tuvalu.santafe.edu/~simon/styled-8/) includes code for Viterbi decoding.
