Processamento e Recuperação de Informação

Information Extraction : Hidden Markov Models

Departamento de Engenharia Informática
Instituto Superior Técnico

1º Semestre
2018/2019

Bibliography - Articles

- Rakesh Dugad, U.B. Desai, *A Tutorial on Hidden Markov Models*, Technical Report, Department of Electrical Engineering, Indian Istitute of Technology, 1996.
- Lawrence R. Rabiner, *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*, Proceedings of the IEEE, 77(2), February, 1989.

Outline

An Example Generative Story

- Suppose a person, inside a room, has three coins (possibly biased)
- The person chooses a coin, randomly, and throws it, chooses another, throws it, and so on...
- The choice of a coin depends on the previously chosen coin
- We are outside the room, looking through a window
- We can only see the outcome of the coin (heads or tails)

- Suppose we observe the sequence:

  HHTTTHHTHTTHHTTHHT

- What probabilities can influence this outcome?

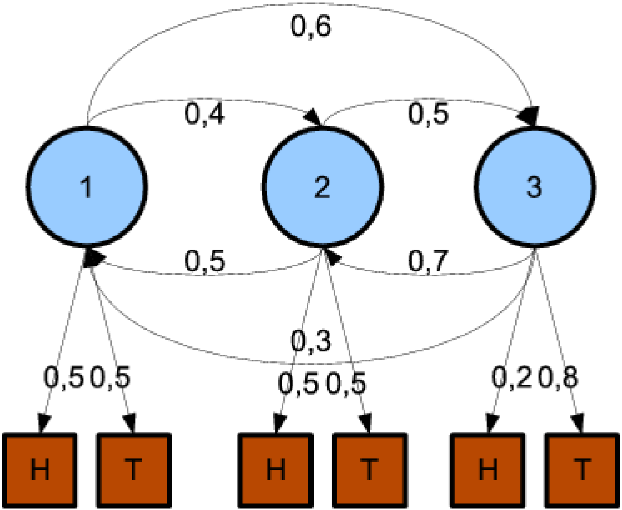
Hidden Markov Model

The outcome is influenced by three factors:

1. The probability of choosing a given coin first
2. The probability of choosing a given coin, after another
3. The probability of getting heads or tails

These three sets of probabilities characterize a Hidden Markov Model for the coin tossing experiment

Finite State Machine Representation



Definitions

We will use the following notation:

- N — the number of states in the model
- M — the number of distinct observation symbols
- T — the length of the observation sequence
- i_t — the state in which we are at time t
- $V = \{V_1, \dots, V_M\}$ — the set of observation symbols
- $\pi = \{\pi_i\}$ — the probability of being in state i at the beginning of the experiment, i.e. $\pi_i = P(i_1 = i)$
- $A = \{a_{ij}\}$ — the probability of being in state j at time $t + 1$ given that we were in state i at time t, i.e. $P(i_{t+1} = j | i_t = i)$
- $B = \{b_j(k)\}$ — the probability of observing symbol v_k given that we are in state j, i.e., $P(v_k \text{ at } t | i_t = j)$
- O_t the observation symbol observed at time t
- $\lambda = (A, B, \pi)$ — the Hidden Markov Model

An example

Consider a set of n urns, each containing marbles of m different colors. We are randomly choosing an urn and randomly picking a marble from it. How do we model this as an HMM?

- What are the states?
- What are the observation symbols?

An example (cont.)

- The model would represent the following sequence of events:

An example (cont.)

- The model would represent the following sequence of events:
  1. We choose one of the urns, according to probability distribution π

An example (cont.)

- The model would represent the following sequence of events:
  1. We choose one of the urns, according to probability distribution π
  2. We choose a marble from that urn, according to probability distribution B

An example (cont.)

- The model would represent the following sequence of events:
  1. We choose one of the urns, according to probability distribution π
  2. We choose a marble from that urn, according to probability distribution B
     - At this moment we are at time t_1, state i_1, and observed symbol O_1

An example (cont.)

- The model would represent the following sequence of events:
  1. We choose one of the urns, according to probability distribution π
  2. We choose a marble from that urn, according to probability distribution B
     - At this moment we are at time t_1, state i_1, and observed symbol O_1
     - After the next step we will be at time t_2

An example (cont.)

- The model would represent the following sequence of events:
  1. We choose one of the urns, according to probability distribution π
  2. We choose a marble from that urn, according to probability distribution B
     - At this moment we are at time t_1, state i_1, and observed symbol O_1
     - After the next step we will be at time t_2
  3. We choose another urn, according to probability distribution A

An example (cont.)

- The model would represent the following sequence of events:
  1. We choose one of the urns, according to probability distribution π
  2. We choose a marble from that urn, according to probability distribution B
     - At this moment we are at time t_1, state i_1, and observed symbol O_1
     - After the next step we will be at time t_2
  3. We choose another urn, according to probability distribution A
  4. Repeat from step 2, until we have made T observations (i.e., $t=T$)

An example (cont.)

- The model would represent the following sequence of events:
  1. We choose one of the urns, according to probability distribution π
  2. We choose a marble from that urn, according to probability distribution B
     - At this moment we are at time t_1, state i_1, and observed symbol O_1
     - After the next step we will be at time t_2
  3. We choose another urn, according to probability distribution A
  4. Repeat from step 2, until we have made T observations (i.e., t$=T$)
- The generated observation sequence will be $O_1, O_2, \ldots, O_T$.

Three Problems for HMMs

1. Given the model $\lambda = (A, B, \pi)$, compute $P(O|\lambda)$
   - I.e., compute the probability of observing a given sequence
   - Applications in language modeling, spelling correction, ...
2. Given the model $\lambda = (A, B, \pi)$, choose a state sequence $I = i_1, i_2, \ldots, i_T$ such that $P(O, I|\lambda)$ is maximized, for a given observation sequence $O = O_1, O_2, \ldots, O_T$
   - I.e., compute the most likely sequence of states to have generated an observation sequence (i.e., decoding)
   - Applications in information extraction (e.g., chunking, named entity recognition, ...)
3. Adjust the model parameters $\lambda = (A, B, \pi)$ such that $P(O|\lambda)$ or $P(O, I|\lambda)$, is maximized
   - I.e., based on a series of observations and/or state sequences, compute the HMM
   - Learning model parameters from annotated data

Outline

Computing the Probabilities

- We know that

$$P(O|\lambda) = \sum_{I} P(O|I,\lambda)P(I|\lambda)$$

- and since

$$\begin{aligned} P(O|I,\lambda) &= b_{i_1}(O_1)b_{i_2}(O_2)\cdots b_{i_T}(O_T) \\ P(I|\lambda) &= \pi_{i_1} a_{i_1 i_2} a_{i_2 i_3} \cdots a_{i_{T-1} i_T} \end{aligned}$$

- we have that

$$P(O|\lambda) = \sum_{I} \pi_{i_1} b_{i_1}(O_1) a_{i_1 i_2} b_{i_2}(O_2) \cdots a_{i_{T-1} i_T} b_{i_T}(O_T)$$

The Problem

$$P(O|\lambda) = \sum_{I} \pi_{i_1} b_{i_1}(O_1) a_{i_1 i_2} b_{i_2}(O_2) \cdots a_{i_{T-1} i_T} b_{i_T}(O_T)$$

- Computing each summand requires $2T - 1$ multiplications
- The are N^T possible state sequences
- Thus, the complexity is $O(2TN^T)$: unfeasible
- However, there is a more efficient way of computing $P(O|\lambda)$: the forward/backward procedure

The Forward Procedure

- Consider the forward variable $\alpha_t(i)$, defined as:

$$\alpha_t(i) = P(O_1, O_2, \ldots, O_t, i_t = i|\lambda)$$

  i.e., the probability of a partial observation sequence (until time t) that ends in state i

- $\alpha_t(i)$ can be computed as follows:
  1. Compute the probability of starting in state i and observing O_1: $\alpha_1(i)$
  2. For time $t+1$ compute the probability of reaching a state j and observing O_{t+1}, knowing that we already computed all probabilities for all times $\leq t$
  3. The final probability (at time T) will be the sum of all probabilities for each possible ending state i: $\alpha_T(i)$

Computing The Forward Procedure

1. Initial step:

$$\alpha_1(i) = \pi_i b_i(O_1) \ , \ 1 \leq i \leq N$$

Computing The Forward Procedure

1. Initial step:

$$\alpha_1(i) = \pi_i b_i(O_1) \ , \ 1 \leq i \leq N$$

2. For $t = 1, 2, \ldots, T-1$, $1 \leq j \leq N$

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij} \right] b_j(O_{t+1})$$

Computing The Forward Procedure

1. Initial step:

$$\alpha_1(i) = \pi_i b_i(O_1) \ , \ 1 \leq i \leq N$$

2. For $t = 1, 2, \ldots, T-1$, $1 \leq j \leq N$

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij} \right] b_j(O_{t+1})$$

3. Thus, we have that:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

Time Complexity

- Step 1 requires N multiplications
- Step 2 requires $N+1$ multiplications. This is performed for all N states and $T-1$ times, yielding $(N+1)N(T-1)$ multiplications
- Step 3 requires only to sum the computed values
- Thus, the time complexity is $O(N^2T)$

Backward Procedure

- A similar procedure can be applied *moving backwards*
- Consider the backward variable $\beta_t(i)$, defined as:

$$\beta_t(i) = P(O_{t+1}, O_{t+2}, \ldots, O_T | i_t = i, \lambda)$$

  i.e., the probability of observing a partial sequence starting at time $t+1$ and state i
- $\beta_t(i)$ can also be computed as follows:
  1. $$\beta_T(i) = 1 \;,\; 1 \leq i \leq N$$
  2. For $t = T-1, T-2, \ldots, 1$, $1 \leq i \leq N$, we have
  $$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)$$
  3. Thus, we have that:
  $$P(O|\lambda) = \sum_{i=1}^{N} \pi_i b_i(O_1) \beta_1(i)$$

Outline

The Decoding Problem

- We want to find a sequence of states $I = i_1, i_2, \ldots, i_T$ such that the probability of observing a sequence $O = O_1, O_2, \ldots, O_T$ is greater than for any other sequence
- I.e., Find I that maximizes $P(O, I|\lambda)$

$$\arg\max_{\{i_t\}_{t=1}^{T}} P(O, i_1, i_2, \ldots, i_T|\lambda)$$

- This can be computed using the Viterbi Algorithm

The Viterbi Algorithm

- We know that

$$\begin{aligned} P(O, I|\lambda) &= P(O|I, \lambda)P(I|\lambda) \\ &= \pi_{i_1} b_{i_1}(O_1) a_{i_1 i_2} b_{i_2}(O_2) \cdots a_{i_{T-1} i_T} b_{i_T}(O_T) \end{aligned}$$

- Thus, we can define

$$U(i_1, i_2, \ldots, i_T) = -\left[\ln\left(\pi_{i_1} b_{i_1}(O_1)\right) + \sum_{t=2}^{T} \ln\left(a_{i_{t-1} i_t} b_{i_t}(O_t)\right)\right]$$

- so that

$$P(O, I|\lambda) = \exp(-U(i_1, i_2, \ldots, i_T))$$

- and our problem becomes

$$\arg\min_{\{i_t\}_{t=1}^{T}} U(i_1, i_2, \ldots, i_T)$$

The Viterbi Algorithm (cont.)

- We can view the term $-\ln(a_{i_j i_k} b_{i_k}(O_t))$ as the cost of going from state i_j to state i_k at time t
- The Viterbi Algorithm is a dynamic programming approach to compute the path of least cost
- The total cost of a path is the sum of the weights on the edges we cross
  - Note that this is equivalent to multiplying the probabilities

Computing the Viterbi Algorithm (1)

- Let $\delta_t(i)$ be the accumulated weight at state i and time t
- Let $\psi_t(j)$ be the state at time $t-1$ with the lowest cost transition to state j at time t

1. Initialization, for $1 \leq i \leq N$:

$$\begin{aligned} \delta_1(i) &= -\ln(\pi_i) - \ln(b_i(O_1)) \\ \psi_1(i) &= 0 \end{aligned}$$

2. Recursive computation, for $2 \leq t \leq T$, $1 \leq j \leq N$:

$$\begin{aligned} \delta_t(j) &= \min_{1 \leq i \leq N} [\delta_{t-1}(i) - \ln(a_{ij})] - \ln(b_j(O_t)) \\ \psi_t(j) &= \arg\min_{1 \leq i \leq N} [\delta_{t-1}(i) - \ln(a_{ij})] \end{aligned}$$

Computing the Viterbi Algorithm (2)

3. Termination:

$$
\begin{aligned}
P^* &= \min_{1 \leq 1 \leq N} [\delta_T(i)] \\
q_T^* &= \arg\min_{1 \leq i \leq N} [\delta_T(i)]
\end{aligned}
$$

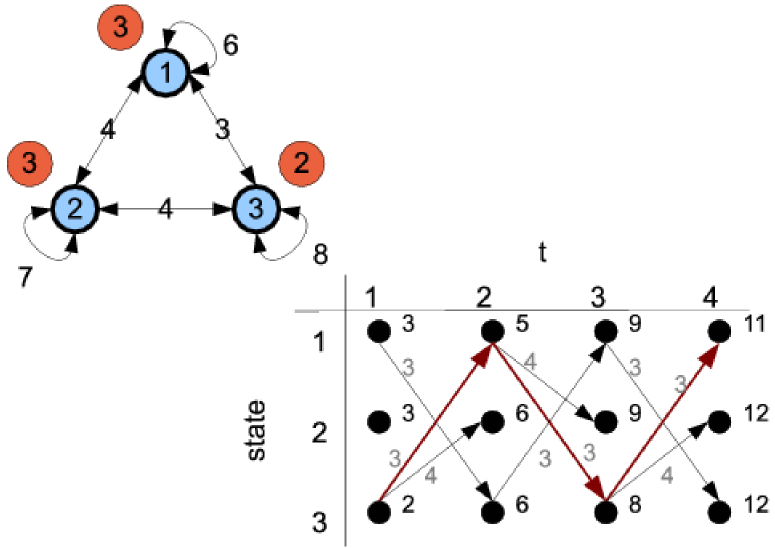
4. Trace back, for $t = T, T-1, T-2, \ldots, 1$:

$$
q_t^* = \psi_{t+1}(q_{t+1}^*)
$$

- $Q^* = \{q_1^*, q_2^*, \ldots, q_T^*\}$ is the optimal state sequence
- $\exp(-P^*)$ is the optimized probability for the state sequence

- Complexity: $O(N^2 T)$

An Example Computation



Notes

- The Viterbi algorithm can be used with HMMs, and also with other sequential classification models (e.g., structured Perceptrons, CRFs, neural network approaches, ...)
- Other *decoding* approaches are also frequently used in practice, one example being posterior decoding
  - Determine, independently for every symbol O_t, the most probable state using the forward/backward procedure
  - Often more effective when several concurring paths have similar probabilities
- Some practical implementations of Information Extraction tools, leveraging sequential classification models, rely on methods such as beam search to find an approximate solution to the problem of finding state sequences

Outline

Learning HMMs

- In a supervised setting, we use the training data to estimate the probabilities
- Transition probabilities

$$\hat{P}(i \rightarrow i') = \frac{c(i \rightarrow i')}{\sum_{s \in I} c(i \rightarrow s)}$$

- Emission probabilities

$$\hat{P}(i \uparrow o) = \frac{c(i \uparrow o)}{\sum_{\rho \in O} c(i \uparrow \rho)}$$

- $c(i \rightarrow i')$ is the number of times there is a transition from state i to state i' (in a training set)
- $c(i \uparrow o)$ counts the number of times symbol o is observed in state i (in a training set)
- The estimation of beginning probabilities is similar to that of transition probabilities, but we count the number of times there is a transition from the start (i.e., the beginning of a training sequence) to a state i

Improving Probability Estimates

- Problem: sparse training data causes poor probability estimates
  - E.g., unseen symbols have emission probabilities of zero
- Solution: use probability smoothing techniques
  - Laplace smoothing
  - Absolute discounting
  - ...

Laplace Smoothing

- Adds 1 to every count of occurrences
- Moves all estimates towards the uniform distribution
- All unseen words will have equal probability

An example:

$$\hat{P}(i \uparrow o) = \frac{c(i \uparrow o) + 1}{\sum_{\rho \in O} c(i \uparrow \rho) + |O|}$$

Absolute Discounting

- Localized (per state) smoothing
- Appropriate if zero probabilities vary from state to state
- Subtracts a fixed discount $0 < d < 1$ from all symbols with count > 0
- The total discounted value is distributed by the remaining symbols

An example:

$$\hat{P}(i \uparrow o) = \begin{cases} \frac{c(i \uparrow o) - d}{\sum_{\rho \in O} c(i \uparrow \rho)} & \text{if } c(i \uparrow o) > 0 \\ \frac{d(|O| - |Z_q|)}{|Z_q| * \sum_{\rho \in O} c(i \uparrow \rho)} & \text{if } c(i \uparrow o) = 0 \end{cases}$$

where $|Z_q|$ is the number of symbols with zero count in state i.

Unsupervised Learning of HMMs

- We want to train an HMM model with a set of example observation sequences such that, when a similar sequence is discovered later the model is able to identify it.
- Most well known method
  - Baum-Welch algorithm
- Other methods exist
  - E.g. Segmental K-means

The Baum-Welch Method

- Assume an initial model λ
  - Can be constructed in any way (e.g. randomly)
- Maximizes $P(O|\lambda)$ by adjusting λ
  - Called the maximum likelihood criterion

Probability of Visiting a State

- Let

$$\gamma_t(i) = P(i_t = t | O, \lambda)$$

i.e. the probability of being in state i at time t given the observation sequence O and the model λ

- Applying Bayes rule:

$$\gamma_t(i) = \frac{P(i_t = i, O)}{P(O|\lambda)} = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)}$$

where $\alpha_t(i)$ is computed as in the Forward procedure and $\beta_t(i)$ is computed as in the Backward procedure

Probability of Transitioning

- Let

$$\xi_t(i) = P(i_t = i, i_{t+1} = j | O, \lambda)$$

i.e. the probability of being in state i at time t and making a transition to state j at time $t+1$, given the observation sequence O and the model λ

- Applying Bayes rule:

$$\xi_t(i) = \frac{P(i_t = i, i_{t+1} = j, O | \lambda)}{P(O|\lambda)} = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{P(O|\lambda)}$$

Expected Number of Transitions

Expected number of visits to state i:

$$\sum_{t=1}^{T} \gamma_t(i)$$

Expected number of transitions from state i:

$$\sum_{t=1}^{T-1} \gamma_t(i)$$

Expected number of transitions from state i to state j:

$$\sum_{t=1}^{T-1} \xi_t(i,j)$$

Baum-Welch Re-Estimation Formulas

The new model paramters $\hat{\lambda} = (\hat{A}, \hat{B}, \hat{\pi})$ can be computed as:

$$\hat{\pi}_i = \gamma_1(i)$$

$$\hat{a}_{ij} = \sum_{t=1}^{T-1} \xi_t(i,j) \Big/ \sum_{t=1}^{T-1} \gamma_t(i)$$

$$\hat{b}_i(k) = \sum_{t=1|O_t=k}^{T} \gamma_t(i) \Big/ \sum_{t=1}^{T} \gamma_t(i)$$

Multiple Observation Sequences

For multiple observation sequences, sum $\xi_t(i,j)$ and $\gamma_t(i)$ and over all sequences:

$$\hat{\pi}_i = \sum_O \gamma_1(i)$$

$$\hat{a}_{ij} = \sum_O \sum_{t=1}^{T-1} \xi_t(i,j) \Big/ \sum_O \sum_{t=1}^{T-1} \gamma_t(i)$$

$$\hat{b}_i(k) = \sum_O \sum_{t=1|O_t=k}^{T} \gamma_t(i) \Big/ \sum_O \sum_{t=1}^{T} \gamma_t(i)$$

The final values will then have to be normalized.

Outline

Other Models

- Structured Perceptron
- Conditional Random Fields
- Recurrent or Convolutional Deep Neural Networks
- ...

Restructuring HMMs With Features (1)

- In a regular HMM, we have that:

$$
\begin{aligned}
P(O, I|\lambda) &= P(O|I, \lambda)P(I|\lambda) \\
&= \pi_{i_1} b_{i_1}(O_1) a_{i_1, i_2} b_{i_2}(O_2) \cdots a_{i_{T-1} i_T} b_{i_T}(O_T)
\end{aligned}
$$

- Considering the log likelihood:

$$
\begin{aligned}
\log(P(O, I|\lambda)) = {} & \log(\pi_{i_1}) + \log(b_{i_1}(O_1)) + \\
& \log(a_{i_1 i_2}) + \log(b_{i_2}(O_2)) + \cdots + \\
& \log(a_{i_{T-1} i_T}) + \log(b_{i_T}(O_T))
\end{aligned}
$$

- Considering scores, and assuming that $W(t, O_t) = \log(a_{i_{t-1} i_t}) + \log(b_{i_t}(O_t))$, we have:

$$
S(O, I|\lambda) = \sum_{t=1}^{T} W(t, O_t)
$$

Restructuring HMMs With Features (2)

- In the previous example, we saw how to express a model equivalent to an HMM through scoring functions $W(t, Ot)$ that leverage state transitions between adjacent positions in the sequence (i.e., from $t-1$ to t), and symbol emissions for position t of the input sequence.
- Scoring functions can also be written as a linear combination of K different features, again describing state transitions between adjacent positions in the sequence, and symbol emissions for position t of the input sequence.
- In information extraction applications, we want to find the state sequence I that satisfies:

$$\hat{I} = \arg\max_{I} S(O, I|\lambda) = \arg\max_{I} \sum_{t=1}^{T} \sum_{k=1}^{K} \lambda_k f_k(I_t, I_{t-1}, O_t)$$

Structured Perceptron (just a hint)

- Simple discriminative model that enables exploring features representing symbols (e.g., *capital letters denote nouns?*)
- Viterbi algorithm now considers feature weights for computing costs
- High dimensional feature vector represents each possible transition/emission (e.g., one feature per emission/transition in a HMM)
- Update feature weights incrementally, so as to increase/decrease score of correct/incorrect labellings

1. Create feature map and set initial feature weights w
2. For ϵ iterations, and for each labeled pair $\{O, I\}$ in the training data
   1. Compute $\hat{I}$ for the observation sequence O, using the Viterbi algorithm and the feature vector w
   2. If $I = \hat{I}$ do not update the model, else
      1. Compute the feature vector f for the pair $\{O, I\}$
      2. Compute the feature vector $\hat{f}$ for the pair $\{O, \hat{I}\}$
      3. Update the feature vector: $w = w + f - \hat{f}$

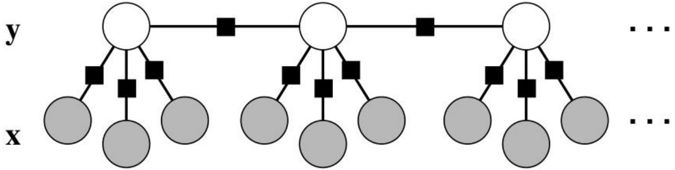
Guarantees with Perceptron Learning

Simple additive update seems intuitive, but do we have any guarantees? Collins (2002) has some proofs showing that:

- If the data is separable with some margin, then the algorithm will converge on weights which give zero error on the training data
- If the training data is not separable, but "close" to being separable, then the algorithm will make a small number of mistakes (on the training data)
- If the algorithm makes a small number of errors on the training data, it is likely to generalise well to unseen data

Reference: *Michael Collins, Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms, 2002*

Linear-Chain Conditional Random Fields (1)



$$P(I|O) = \frac{1}{Z(O)} \exp\left\{\sum_{t=1}^{T}\sum_{k=1}^{K} \lambda_k f_k(I_t, I_{t-1}, O_t)\right\}$$

- Inference with the Viterbi algorithm
- Infering the parameters by maximum likelihood learning, e.g., through generalized iterative scaling or through gradient descent algorithms

Linear-Chain Conditional Random Fields (2)

- Inference leverages the Viterbi algorithm to find the following argmax efficiently

$$\begin{aligned}
\arg\max_I P(I|O) &= \arg\max_I \exp\left\{\sum_{t=1}^{T}\sum_{k=1}^{K}\lambda_k f_k(I_t, I_{t-1}, O_t)\right\}\frac{1}{Z(O)} \\
&= \arg\max_I \left\{\sum_{t=1}^{T}\sum_{k=1}^{K}\lambda_k f_k(I_t, I_{t-1}, O_t)\right\} - \log\left(Z(O)\right) \\
&= \arg\max_I \left\{\sum_{t=1}^{T}\sum_{k=1}^{K}\lambda_k f_k(I_t, I_{t-1}, O_t)\right\}
\end{aligned}$$

- Training is a convex optimization problem
  - Generalized iterative scaling, computing the feature expectations (denominator) through forward-backward

$$\lambda_i^{k+1} = \lambda_i^k + \frac{1}{C}\log\left(\frac{\sum_{t=1}^{T} f_i(I_t, I_{t-1}, O_t)}{\sum_{I}' P(I'|O,\lambda^k)\sum_{t=1}^{T} f_i(I'_t, I'_{t-1}, O_t)}\right)$$

Questions?

Extra Credits

The Segmental K-means Algorithm

- Segmental K-means adjusts the parameters $\lambda = (A, B, \pi)$ to maximize $P(O, I|\lambda)$, where I is the optimal sequence of states for observation sequence O
- Idea: evolve from λ^k to λ^{k+1} such that $P(O, I_k^*|\lambda^k) \leq P(O, I_{k+1}^*|\lambda^{k+1})$
- I_k^* is the optimal state sequence for $O = O_1, O_2, \ldots, O_T$ and λ_k
- Function $P(O, I^*|\lambda) = \max_I P(O, I|\lambda)$ is called the state optimized likelihood function
- This optimization criterion is called maximum state optimized likelihood criterion

The Segmental K-means Algorithm (cont.)

Basic assumptions:

- We have a set of w observation sequences available (training sequences)
- Each training sequence $O = O_1, O_2, \ldots, O_T$ consists of T observation symbols
- Each observation symbol O_i is a vector of D (≥ 1) dimensions

Computing the Algorithm (1)

1. Randomly choose N observation symbols; assign each of the wT training symbols to the closest chosen symbol (e.g., using Euclidean distance)
2. Calculate the initial probabilities and transition probabilities:
   - For $1 \leq i \leq N$:

$$\hat{\pi}_i = \frac{\text{Number of occurrences of } \{O_1 \in i\}}{\text{Total number of occurrences of } O_1 \text{ (i.e., } w)}$$

   - For $1 \leq i \leq N$, $1 \leq j \leq N$:

$$\hat{a}_{ij} = \frac{\text{Number of occurrences of } \{O_t \in i \text{ and } O_{t+1} \in j\}, \forall t}{\text{Total number of occurrences of } O_t \in i, \forall t}$$

Computing the Algorithm (2)

3. Compute mean and covariance matrix of each state. For $1 \leq i \leq N$:

$$\hat{\mu}_i = \frac{1}{N_i} \sum_{O_t \in i} O_t$$

$$\hat{V}_i = \frac{1}{N_i} \sum_{O_t \in i} (O_t - \hat{\mu}_i)^T (O_t - \hat{\mu}_i)$$

4. Calculate the probability distribution of each symbol in each state:

$$\hat{b}_i(O_t) = \frac{1}{((2\pi)^{D/2} |\hat{V}_i|^{1/2}} \exp[-\frac{1}{2}(O_t - \hat{\mu}_i)\hat{V}_i^{-1}(O_t - \hat{\mu}_i)^T]$$

- We are assuming a Gaussian distribution. Others could be used.

Computing the Algorithm (3)

5. Find the optimal state sequence I^* for each training sequence, using $\hat{\lambda}_i = (\hat{A}_i, \hat{B}_i, \hat{\pi}_i)$; reassign O_t (of the k-th training sequence) to state i iff i_t^* (of the k-th training sequence) is i
   - For instance: if O_2 of the 5th sequence was in state 3, and in I^* (for the 5th training sequence) we have that i_2^* is 4, we assign O_2 of the 5th sequence to state 4.
6. If any symbol was reassigned, repeat from step 2, otherwise stop.

- It can be proved that the algorithm converges to the state-optimized likelihood function for many different observation probability distributions