Chapter 7-2:

Discrete Sequential Data

Jilles Vreeken

IRDM '15/16

26 Nov 2015

UNIVERSITÄT DES SAARLANDES

CLUSTER OF EXCELLENCE

max planck institut informatik

IRDM Chapter 7, overview

- Time Series
  1. Basic Ideas
  2. Prediction
  3. Motif Discovery
- Discrete Sequences
  4. Basic Ideas
  5. Pattern Discovery
  6. Hidden Markov Models

You'll find this covered in
Aggarwal Ch. 3.4, 14, 15

IRDM Chapter 7, today

- Time Series
  1. Basic Ideas
  2. Prediction
  3. Motif Discovery
- Discrete Sequences
  4. Basic Ideas
  5. Pattern Discovery
  6. Hidden Markov Models

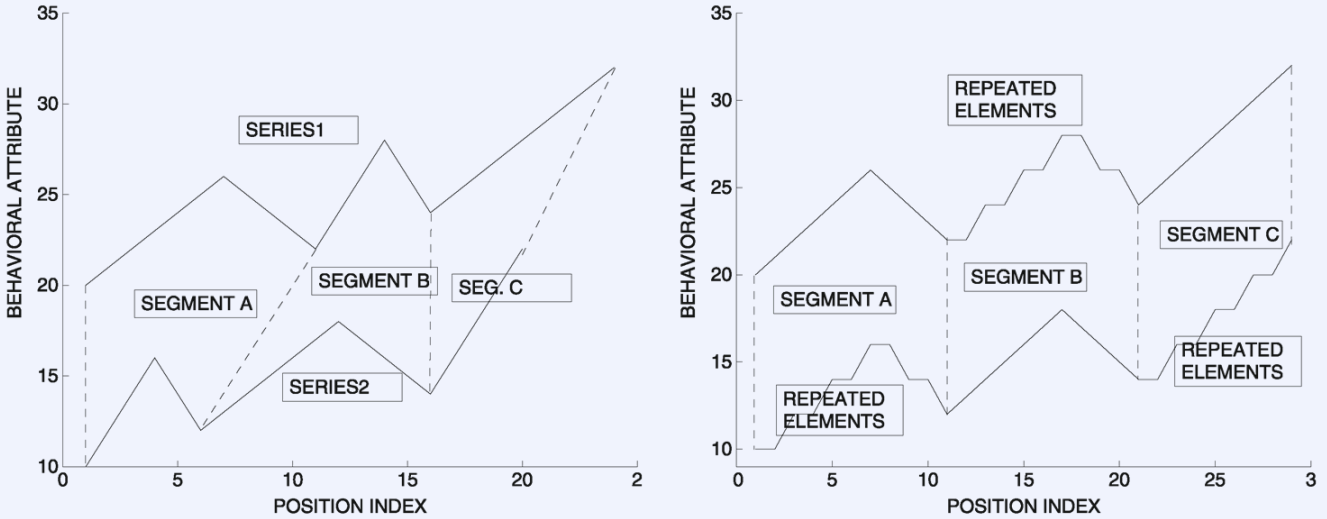
You'll find this covered in
Aggarwal Ch. 3.4, 14, 15

Chapter 7.3, ctd:
Motif Discovery

Aggarwal Ch. 14.4, 3.4

Dynamic Time Warping

DTW stretches the time axis of one series to enable better matches



(a) Original series

(b) Warped series

(Aggarwal Ch. 3.4)

DTW, formally

Let $DTW(i,j)$ be the optimal distance between the first i and first j elements of time series X of length n and Y of length m

$$DTW(i,j) = distance(X_i, Y_j) + \min \begin{cases} DTW(i,j-1) & \text{repeat } x_i \\ DTW(i-1,j) & \text{repeat } y_j \\ DTW(i-1,j-1) & \text{repeat neither} \end{cases}$$

We initialise as follows

- $DTW(0,0) = 0$
- $DTW(0,j) = \infty$ for all $j \in \{1, \dots, n\}$
- $DTW(i,0) = \infty$ for all $i \in \{1, \dots, m\}$

We can then simply iterate by increasing i and j

(Aggarwal Ch. 3.4)

Computing DTW (1)

Let $DTW(i,j)$ be the optimal distance between the first i and first j elements of time series X of length n and Y of length m

$$DTW(i,j) = distance(X_i, Y_j) + \min \begin{cases} DTW(i,j-1) & \text{repeat } x_i \\ DTW(i-1,j) & \text{repeat } y_j \\ DTW(i-1,j-1) & \text{repeat neither} \end{cases}$$

From the initialised values, can simply iterate by increasing i and j:

for $i = 1$ to m
  for $j = 1$ to n
    compute $DTW(i,j)$

We can also compute it recursively, by dynamic programming.

Both naïve strategies cost $O(nm)$, however.

(Aggarwal Ch. 3.4)

Computing DTW (2)

Let $DTW(i,j)$ be the optimal distance between the first i elements of time series X of length n and the first j elements of time series Y of length m

$$DTW(i,j) = distance(X_i, Y_j) + \min \begin{cases} DTW(i,j-1) & \text{repeat } x_i \\ DTW(i-1,j) & \text{repeat } y_j \\ DTW(i-1,j-1) & \text{repeat neither} \end{cases}$$

We can speed up computation by imposing constraints.

- e.g. a window constraint to compute $DTW(i,j)$ only when $|i-j| \leq w$
- we then only need $\max\{0, \mathrm{i} - \mathrm{w}\} - \min\{n, i+w\}$ inner loops

(Aggarwal Ch. 3.4)

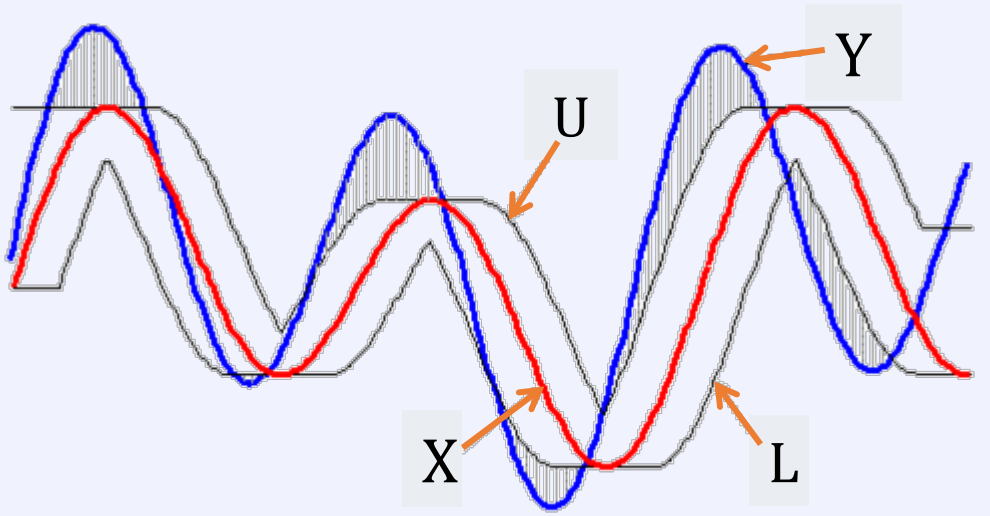
Lower bounds on DTW

Even smarter is to speed up DTW using a lower bound.

$$LB\_Keogh(X,Y) = \sum_{i=1}^{n} \begin{cases} (Y_i - U_i)^2 & \text{if } X_i > U_i \\ (Y_i - L_i)^2 & \text{if } X_i < L_i \\ 0 & \text{otherwise} \end{cases}$$

$U_i = \max\{X_{i-r} : X_{i+r}\}$
$L_i = \min\{X_{i-r} : X_{i+r}\}$

where r is the reach, the allowed range of warping



Discrete Sequences

Chapter 7.4:
Basic Ideas

Aggarwal Ch. 14.1-14.2

Trouble in Time Series Paradise

Continuous real-valued time series have their downsides

- mining results rely on either a **distance function** or **assumptions**
- indexing, pattern mining, summarisation, clustering, classification, and outlier detection results hence rely on **arbitrary choices**

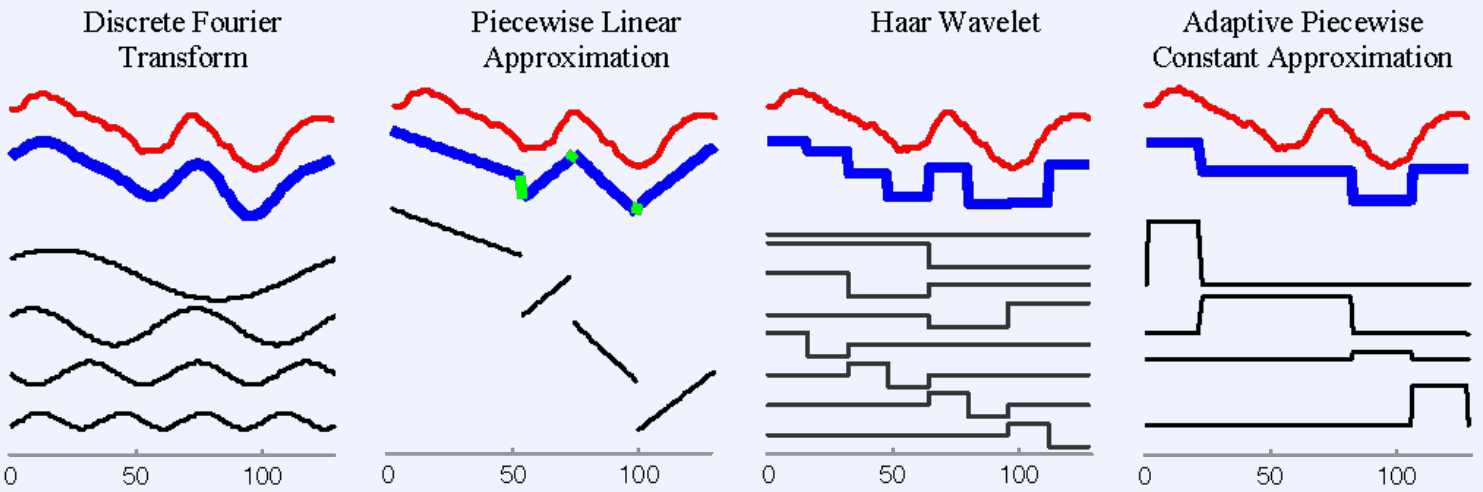
Discrete sequences are often easier to deal with

- mining results rely mostly on **counting**

How to transform a time series into an event sequence?

- discretisation

Approximating a Time Series



(Lin et al. 2002, 2007)

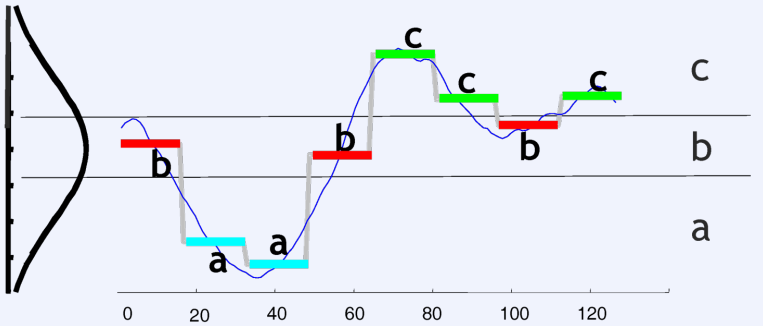
SAX

Symbolic **A**ggregate Approx**i**mation (SAX)

- most well-known approach to discretise a time series
- type of piece-wise aggregated approximation (PAA)

How to do SAX

- divide the data into w **frames**
- compute the mean per frame
- perform equal-height binning over the means, to obtain an alphabet of a characters



(Lin et al. 2002, 2007)

Definitions

A discrete **sequence** $X_1 \dots X_n$ of length n and dimensionality d, contains d discrete feature values at each of n different timestamps $t_1 \dots t_n$.

Each of the n **components** X_i contains d discrete behavioral attributes $(x_i^1 \dots x_i^d)$ collected at the ith timestamp.

The actual time stamps are usually ignored – they only induce an order on the components, or **events**.

Types of discrete sequences

In many applications, the dimensionality is 1

- e.g. strings, such as text or genomes.
- for $\mathrm{AATCGTAC}$ over an alphabet $\Sigma = \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$, each $X_i \in \Sigma$

In some applications, each X_i is not a vector, but a **set**

- e.g. a supermarket transaction, $X_i \subseteq \Sigma$
- there is no order within X_i

We will consider the set-setting, as it is most general

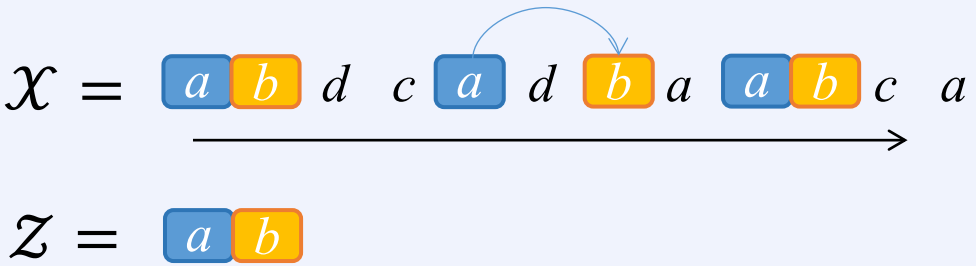
Chapter 7.5:
Frequent Patterns

Aggarwal Ch. 15.2

Sequential patterns

A **sequential pattern** is a sequence.

- to occur in the data, it has to be a subsequence of the data.



Definition: Given two sequences $\mathcal{X} = X_1 \dots X_n$ and $\mathcal{Z} = Z_1 \dots Z_k$ where all elements X_i and Z_i in the sequences are sets. Then, the sequence $\mathcal{Z}$ is a **subsequence** of $\mathcal{X}$, if k elements $X_{i_1} \dots X_{i_k}$ can be found in $\mathcal{X}$, such that $i_1 < i_2 < \cdots < i_k$ and $Z_j \subseteq X_{i_j}$ for each $j \in \{1 \dots k\}$

Support

Depending on whether we have a **database $\boldsymbol{D}$** of sequences, or a **single long sequence**, we have to define the **support** of a sequential pattern differently.

Standard, or 'per sequence' support counting

- given a database $\boldsymbol{D} = \{\mathcal{X}_1, \ldots, \mathcal{X}_N\}$, the support of a subsequence $\mathcal{Z}$ is the number of sequences in $\boldsymbol{D}$ that contain $\mathcal{Z}$.

Window-based support counting

- given a single sequence $\mathcal{X}$, the support of a subsequence $\mathcal{Z}$ is the number of **windows** over $\mathcal{X}$ that contain $\mathcal{Z}$.

(we can define frequency analogue as relative support)

Windows

A **window** $\mathcal{X}[s;e]$ is a strict subsequence of sequence $\mathcal{X}$.

$$\mathcal{X}[s;e] = \langle X_i \in \mathcal{X} \mid e \leq i \leq s \rangle$$

$\mathcal{X} =$ a b d c a d b a a b c a d a b a b c

$\mathcal{Z} =$ 

Window-based support counting

- we can choose a window length w, and sweep over the data

Windows

A window $\mathcal{X}[s;e]$ is a strict subsequence of sequence $\mathcal{X}$.

$$\mathcal{X}[s;e] = \langle X_i \in \mathcal{X} \mid e \leq i \leq s \rangle$$

$\mathcal{X} =$ a b d c a d b a a b c a d a b a b c

$\mathcal{Z} =$ a b : 1

Window-based support counting

- we can choose a window length w, and sweep over the data

Windows

A window $\mathcal{X}[s;e]$ is a strict subsequence of sequence $\mathcal{X}$.

$$\mathcal{X}[s;e] = \langle X_i \in \mathcal{X} \mid e \leq i \leq s \rangle$$

$\mathcal{X} =$ a b d c a d b a a b c a d a b a b c

$\mathcal{Z} =$ a b : 1

Window-based support counting

- we can choose a window length w, and sweep over the data

Windows

A window $\mathcal{X}[s;e]$ is a strict subsequence of sequence $\mathcal{X}$.

$$\mathcal{X}[s;e] = \langle X_i \in \mathcal{X} \mid e \leq i \leq s \rangle$$

$\mathcal{X} =$ a b d c a d b a a b c a d a b a b c

$\mathcal{Z} =$ a b : 1

Window-based support counting

- we can choose a window length w, and sweep over the data

Windows

A window $\mathcal{X}[s;e]$ is a strict subsequence of sequence $\mathcal{X}$.

$$\mathcal{X}[s;e] = \langle X_i \in \mathcal{X} \mid e \leq i \leq s \rangle$$

$\mathcal{X} =$ a b d c a d b a a b c a d a b a b c

$\mathcal{Z} =$ a b : 3

Window-based support counting

- we can choose a window length w, and sweep over the data

Windows

A window $\mathcal{X}[s;e]$ is a strict subsequence of sequence $\mathcal{X}$.

$$\mathcal{X}[s;e] = \langle X_i \in \mathcal{X} \mid e \leq i \leq s \rangle$$

$\mathcal{X} =$ a b d c a d b a a b c a d a b a b c

$\mathcal{Z} =$ a b : 4

Window-based support counting

- we can choose a window length w, and sweep over the data
- support is now dependent on w, what happens with longer w?

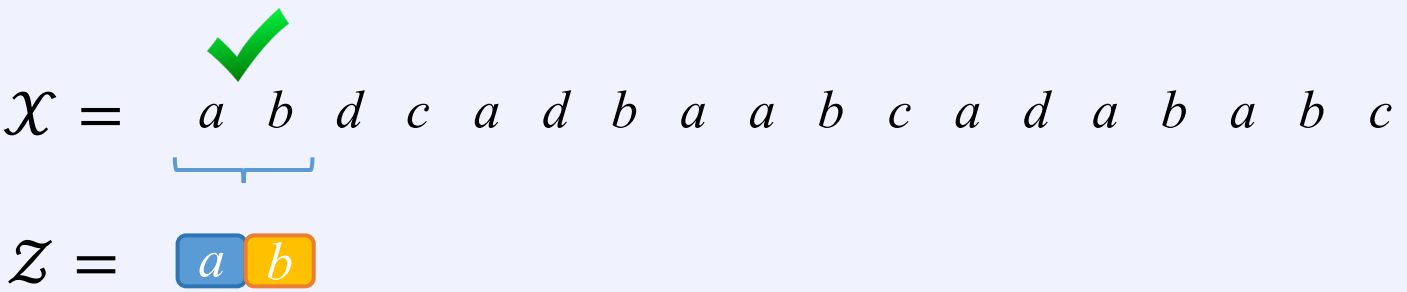
Minimal windows

Fixed window lengths lead to double counting

- if $\mathcal{X}[s;e]$ supports sequence $\mathcal{Z}$ so do $\mathcal{X}[s;e+k]$ and $\mathcal{X}[s-k;e]$

We can avoid this by counting only **minimal windows**

- $w = \mathcal{X}[s;e]$ is a minimal window of pattern $\mathcal{Z}$ if w contains $\mathcal{Z}$ but no other proper sub-windows of w contain $\mathcal{Z}$.
- for efficiency or fun, we may want to set a **maximal window** size



Minimal windows

Fixed window lengths lead to double counting

- if $\mathcal{X}[s;e]$ supports sequence $\mathcal{Z}$ so do $\mathcal{X}[s;e+k]$ and $\mathcal{X}[s-k;e]$

We can avoid this by counting only **minimal windows**

- $w = \mathcal{X}[s;e]$ is a minimal window of pattern $\mathcal{Z}$ if w contains $\mathcal{Z}$ but no other proper sub-windows of w contain $\mathcal{Z}$.
- for efficiency or fun, we may want to set a **maximal window** size

$\mathcal{X} =$ $a\ b\ d\ c\ a\ d\ b\ a\ a\ b\ c\ a\ d\ a\ b\ a\ b\ c$

$\mathcal{Z} =$ $a\ b$

Minimal windows

Fixed window lengths lead to double counting

- if $\mathcal{X}[s;e]$ supports sequence $\mathcal{Z}$ so do $\mathcal{X}[s;e+k]$ and $\mathcal{X}[s-k;e]$

We can avoid this by counting only **minimal windows**

- $w = \mathcal{X}[s;e]$ is a minimal window of pattern $\mathcal{Z}$ if w contains $\mathcal{Z}$ but no other proper sub-windows of w contain $\mathcal{Z}$.
- for efficiency or fun, we may want to set a **maximal window** size

$\mathcal{X} =$ $a\ b\ d\ c\ a\ d\ b\ a\ a\ b\ c\ a\ d\ a\ b\ a\ b\ c$

$\mathcal{Z} =$ $a\ b : 5$

Mining Frequent Sequential Patterns

Like for itemsets, the per-sequence and per-window definitions of support are also monotone

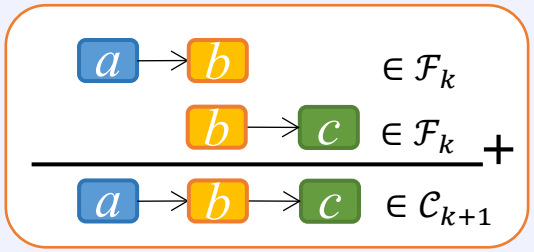
- we can employ level-wise search!

We can modify

- APRIORI to get to GSP (Agrawal & Srikant, 1995; Mannila, Toivonen, Verkamo, 1995)
- ECLAT to get SPADE (Zaki, 2000)
- FP-GROWTH to get PREFIXSPAN (Pei et al., 2001)

Generalised Sequential Pattern Mining

Algorithm GSP(sequence database $\boldsymbol{D}$, minimal support σ)
begin
  $k \leftarrow 1$;
  $\mathcal{F}_k \leftarrow \{\text{all frequent } 1-\text{item elements}\}$
  while $\mathcal{F}_k$ is not empty **do**
    Generate $\mathcal{C}_{k+1}$ by joining pairs of sequences in $\mathcal{F}_k$, such that removing an item from the first element of one sequence matches the sequence obtained by removing an item from the last element of the other
    Prune sequences from $\mathcal{C}_{k+1}$ that violate downward closure
    Determine $\mathcal{F}_{k+1}$ by support counting on $(\mathcal{C}_{k+1}, \boldsymbol{D})$ and retaining sequences from $\mathcal{C}_{k+1}$ with support at least σ
    $k \leftarrow k + 1$
  end
  return $\bigcup_{i=1}^{k} \mathcal{F}_i$
end



(Agrawal & Srikant, 1995; Mannila, Toivonen & Verkamo, 1995)

Episodes

There are many types of sequential patterns

The most well-known are

- n-grams, k-mers, or **strict subsequences**, where we **do not allow** gaps
- serial episodes, or subsequences, where we do allow gaps

$\mathcal{X} =$ a b c c a d b a d b e c d a b a b c

Episodes

There are many types of sequential patterns

The most well-known are

- n-grams, k-mers, or **strict subsequences**, where we **do not allow** gaps
- serial episodes, or subsequences, where we do allow gaps

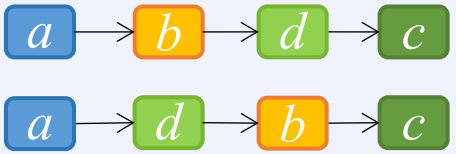
Each element can contain one or more items

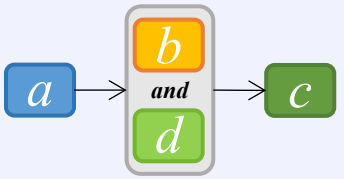
$$\mathcal{X} = \begin{array}{cccccccccccccccccc} & & b & c & & & a & & a & d & b & & c & a & & & a & \\ a & d & d & c & a & d & b & c & e & f & e & d & d & a & b & a & b & c \end{array}$$

Parallel episodes

Serial episodes are still restrictive

- not everything always happens exactly in sequential order





Parallel episodes acknowledge this

- a parallel episode defines a partial order, for a match it requires all parallel events to happen, but does not specify their exact order.
- e.g. first a, then in any order b and d, and then c

We can also combine the two into **generalised episodes**

Chapter 7.6:
Hidden Markov Models

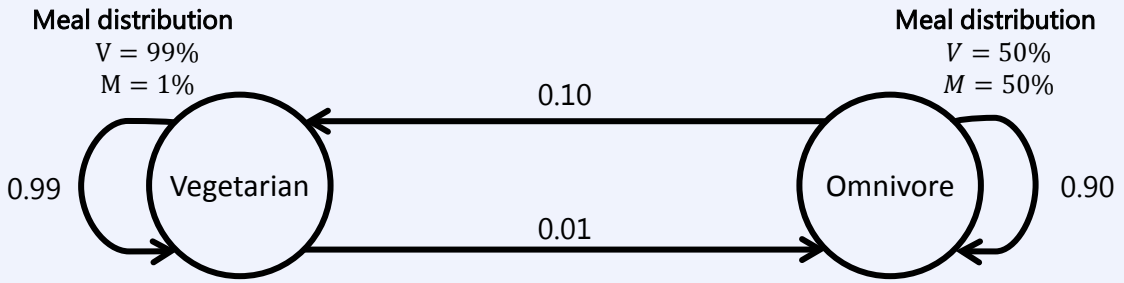
Aggarwal Ch. 15.5

Informal definition

Hidden Markov Models are probabilistic, generative models for discrete sequences. It is a graphical model in which nodes correspond to **system states**, and edges to **state changes**.

In a HMM the states of the system are **hidden**; not directly visible to the user. We only observe a sequence over symbols Σ that the system generates when it switches between states.

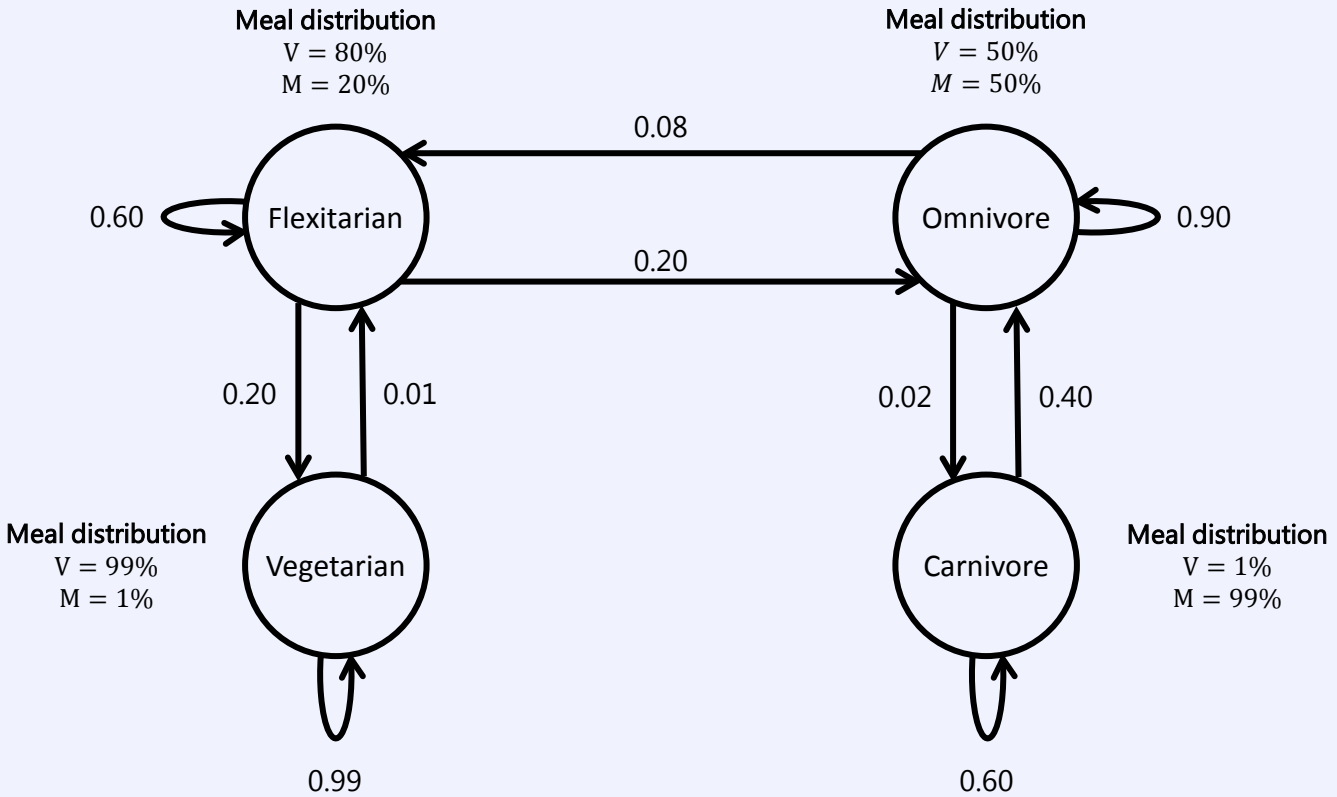
Example HMM



This HMM can generate sequences such as

- VVVVVVVMVV Veggie (common)
- MVMVVMMVM Omni (common)
- MMVMVVVVVV Omni-turned-Veggie (not very common)
- MMMMMMMM Carnivore (rare)

Example HMM (2)



Formal definition

A **Hidden Markov Model** over alphabet $\Sigma = \{\sigma_1, \ldots, \sigma_{|\Sigma|}\}$ is a directed graph $G(S, T)$ consisting of n states $S = \{s_1, \ldots, s_n\}$.

The initial state probabilities are $\pi_i, \ldots, \pi_n$. The (directed) edges correspond to state transitions. The probability of a transition from state s_i to state s_j is denoted by p_{ij}.

For every visit to a state, a symbol from Σ is generated with probability $P(\sigma_i \mid s_j)$.

What to do with an HMM

There are three main things to do with an HMM

1. **Training.**
   Given topology G and database $\boldsymbol{D}$, learn the **initial state probabilities**, **transition probabilities**, and the **symbol emission probabilities.**

2. **Explanation.**
   Given an HMM, determine the **most likely state sequence** that generated test sequence $\mathcal{Z}$.

3. **Evaluation**.
   Given an HMM, determine the **probability of test sequence** $\mathcal{Z}$.

Using an HMM for Evaluation

We want to know the **fit probability** that sequence $\mathcal{X} = X_1 \ldots X_m$ was generated by the given HMM.

Naïve approach

- compute all n^m possible paths over G
- for each, determine probability of generating $\mathcal{X}$
- sum these probabilities, this is the fit probability of $\mathcal{X}$

Recursive Evaluation

The fit probability of the first r symbols[1] can be **computed recursively** from the fit probability of the first $(r-1)$ symbols[2]

Let $\alpha_r(\mathcal{X}, s_j)$ be the probability that the first r symbols of $\mathcal{X}$ are generated by the model, and the last state is s_j.

$$\alpha_r(\mathcal{X}, s_j) = \sum_{i=1}^{n} \alpha_{r-1}(\mathcal{X}, s_i) \cdot p_{ij} \cdot P(X_r \mid s_j)$$

That is, we sum over all paths up to different final nodes.

[1] and a fixed value of the r^{th} state

[2] and a fixed value of the $(r-1)^{st}$ state

Forward Algorithm

We initialise with with $\alpha_1(\mathcal{X}, s_j) = \pi_j \cdot P(\, X_1 \mid s_j \,)$ and then iteratively compute for each $r = 1 \ldots m$.

The fit probability of $\mathcal{X}$ is the sum over all end-states,

$$F(\mathcal{X}) = \sum_{j=1}^{n} \alpha_m(\mathcal{X}, s_j)$$

The complexity of the **Forward Algorithm** is $O(n^2 m)$

But, why?

Good question to ask: why compute the fit probability?

- classification
- clustering
- anomaly detection

For the first two, we can now create group-specific HMMs, and **assign the most likely sequences** to those.

For the the third, we have an HMM for our training data, and can now report **poorly fitting sequences**.

Using an HMM for Explanation

We want to know **why a sequence $\mathcal{X}$ fits our data**. The most likely state sequence gives an intuitive explanation.

Naïve approach

- compute all n^m possible paths over the HMM
- for each, determine probability of generating $\mathcal{X}$
- report the path with **maximum probability**

Instead of naively, can re-use the recursive approach?

Viterbi Algorithm

Any subpath of an optimal state path must also be optimal for generating the corresponding subsequence.

Let $\delta_r(\mathcal{X}, s_j)$ be the probability of the best state sequence generating the first r symbols of $\mathcal{X}$ ending at state s_j, with

$$\delta_r(\mathcal{X}, s_j) = \max_{i \in [1,n]} \delta_{r-1}(\mathcal{X}, s_i) \cdot p_{ij} \cdot P(X_r \mid s_j)$$

That is, we recursively compute the maximum-probability path over all n different paths for different final nodes.

Overall, we initialise recursion with $\delta_1(\mathcal{X}, s_j) = \pi_j P(X_1 \mid s_j)$, and then iteratively compute for $r = 1 \ldots m$.

Training an HMM

So far, we assumed the given HMM was **trained**.
How do we train a HMM in practice?

Learning the parameters of an HMM is difficult

- no known algorithm is guaranteed to give the global optimum

There do exist methods for reasonably effective solutions

- e.g. the Forward-Backward (Baum-Welch) algorithm

Backward

We already know how to calculate the **forward probability** $\alpha_r(\mathcal{X}, s_j)$ for the first r symbols of a sequence $\mathcal{X}$, ending at s_j.

Now, let $\beta_r(\mathcal{X}, s_j)$ be the **backward probability** for the sequence **after and not including** the r^{th} symbol, **conditioned** that the r^{th} state is s_j. We initialise $\beta_{|\mathcal{X}|}(\mathcal{X}, s_j) = 1$, and compute $\beta_r(\mathcal{X}, s_j)$ just as $\alpha_r(\mathcal{X}, s_j)$ but from back to front.

For the Baum-Welch algorithm, we'll also need

- $\gamma_r(\mathcal{X}, s_i)$ for the probability that the r^{th} state corresponds to s_i, and
- $\psi_r(\mathcal{X}, s_i, s_j)$ for the probability of the r^{th} state s_i, and the $(r+1)^{th}$ state s_j

Baum-Welch

We initialize the model parameters **randomly**.

We will then iteratively

(E-step) Estimate $\alpha(\cdot), \beta(\cdot), \psi(\cdot)$, and $\gamma(\cdot)$ from the current model parameters

(M-step) Estimate model parameters $\pi(\cdot), P(\cdot|\cdot), p_{\cdot\cdot}$ from the current $\alpha(\cdot), \beta(\cdot), \psi(\cdot)$, and $\gamma(\cdot)$

until the parameters converge.

This is simply the EM strategy!

Estimating parameters

$\alpha(\cdot)$ Easy. We estimate these using the Forward algorithm.

$\beta(\cdot)$ Easy. We estimate these using the Backward algorithm.

Estimating parameters (2)

$\psi(\cdot)$

We can split this value into first till r^{th}, r^{th}, and $(r+1)^{th}$ till end

$$\psi_r(\mathcal{X}, s_i, s_j) = \alpha_r(\mathcal{X}, s_i) \cdot p_{ij} \cdot P(X_{r+1} \mid s_j) \cdot \beta_{r+1}(\mathcal{X}, s_j)$$

and normalize to probabilities over all pairs i, j. So, easy, after all.

$\gamma(\cdot)$

Easy. For $\gamma_r(\mathcal{X}, s_i)$ just fix s_i and sum over $\psi_r(\mathcal{X}, s_i, s_j)$ varying s_j

But, why?

Conclusions

Discrete sequences are a fun aspect of time series

- many interesting problems

Mining sequential patterns

- more expressive than itemsets, more difficult to define support

Hidden Markov Models

- can be used to predict, explain, evaluate discrete sequences

Thank you!

Discrete sequences are a fun aspect of time series

- many interesting problems

Mining sequential patterns

- more expressive than itemsets, more difficult to define support

Hidden Markov Models

- can be used to predict, explain, evaluate discrete sequences