Hidden Markov Models

HMM Applications

- Malware detection
- Stock prediction
- Bioinformatics
- Speech recognition (ex, Siri)
- Cryptanalysis
- Machine translation
- Handwriting recognition
- Time series analysis

and many more…

- A **hidden Markov model** (**HMM**) includes a **Markov process** that is "hidden"

  We cannot directly observe the state of the process

- But we do have access to **a series of observations** that are **probabilistically related to the underlying Markov model**

Probabilities

- The notation "**|**" denotes "*given*" information, so that **P(B|A)** is read as "**the probability of B, given A**"

  For any two events A and B, we have:

$$P(A \text{ and } B) = P(A)P(B|A)$$

  For example, suppose that we draw two cards without replacement from a standard 52-card deck

  - Let A = {1st card is ace} and B = {2nd card is ace}. Then:

$$P(A \text{ and } B) = P(A)P(B|A) = 4/52 * 3/51 = 12/2652 = 1/221$$

- In this example, P(B) depends on what happens in the first event , so we say that A and B are **dependent events**

Probabilities

On the other hand, suppose we flip a fair coin twice

- The probability that the second flip comes up heads is ½

  Regardless of the outcome of the first coin flip, so these events are **independent**

- For dependent events, the "*given*" information is relevant when determining the sample space.

  Consequently, in such cases we can view the information to the right of the "given" sign as defining the space over which probabilities will be compute

Probabilities

- Note that $P(B|A)$ would define a **First-order Markov Model**
- We can also have **Second-order Markov Model**: $P(C|A,B)$
- And **Third-order Markov Model**: $P(D|A,B,C)$

And so on…

- We rewrite:

$$P(A \text{ and } B) = P(A)\,P(B \mid A)$$

- As:

$$P(B \mid A) = \frac{P(A \text{ and } B)}{P(A)}$$

- This expression can be viewed as the definition of conditional probability

We can see it as an **intersection**

This will be useful later…

$$P(A \text{ and } B) = P(A, B) = P(A \cap B)$$

Markov Process example

We have three sentences:

```
1. “I like cats”
2. “I like you ”
3. “I love SJSU”
```

And 6 states: ‘I’, ‘like’, ‘cats’, ‘you’, ‘love’, ‘SJSU’

Then:

π(‘I’) = 1, π(w) = 0, where w = {‘like’, ‘cats’, ‘you’, ‘love’, ‘SJSU’}

so: π = [1,0,0,0,0,0]

P(‘like’ | ‘I’) = 2/3

P(‘cats’ | ‘like’) = ½

P(‘love’ | ‘I’) = 1/3

P(‘you’ | ‘like’) = 1/2

HMM Example

- Suppose we want to determine the average annual temperature at a particular location on Earth over a series of years in the distant past

  To simplify the problem, we only consider "hot" and "cold" for the average annual temperature (binary)

Modern evidence indicates that:

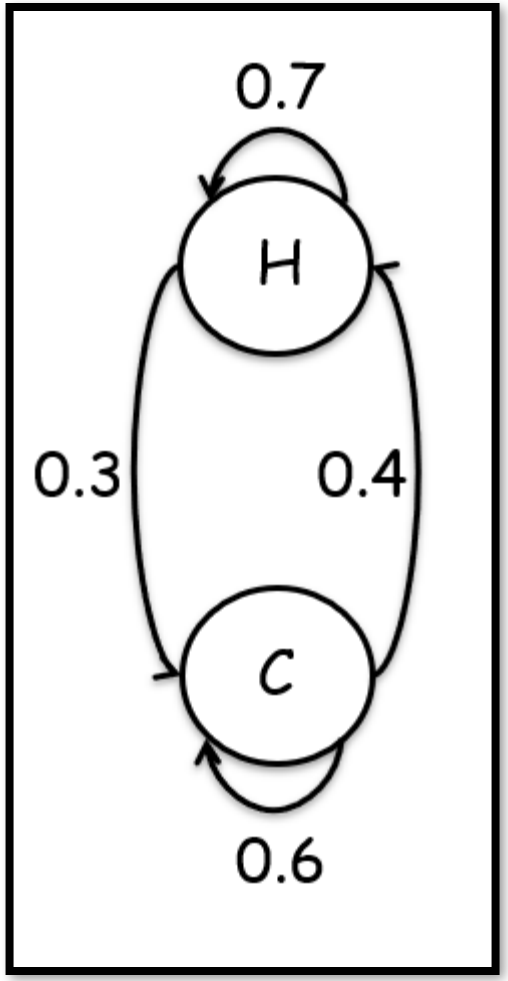
- The probability of a hot year followed by another hot year is 0.7
- The probability that a cold year is followed by another cold year is 0.6

We'll assume that these probabilities also held in the distant past

HMM Example

- This information can be summarized as:

$$\begin{array}{c c} & \begin{array}{cc} H & C \end{array} \\ \begin{array}{c} H \\ C \end{array} & \begin{pmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{pmatrix} \end{array}$$



- The transition from one state to the next is a **Markov process**

  Since the next state depends only on the current state and the fixed probabilities given in the matrix

HMM Example

- Next, suppose that current research indicates a correlation between the size of tree growth rings and temperature.

- For simplicity, we only consider three different tree ring sizes, *small*, *medium*, and *large*, denoted **S**, **M** and **L**

HMM Example

- Based on currently available evidence, the probabilistic relationship between annual temperature and tree ring sizes is given by:

$$\begin{array}{c} \\ H \\ C \end{array} \begin{array}{c} \begin{array}{ccc} S & M & L \end{array} \\ \begin{pmatrix} 0.1 & 0.4 & 0.5 \\ 0.7 & 0.2 & 0.1 \end{pmatrix} \end{array}$$

HMM Example

$$\begin{array}{c c} & \begin{array}{ccc} S & M & L \end{array} \\ \begin{array}{c} H \\ C \end{array} & \begin{pmatrix} 0.1 & 0.4 & 0.5 \\ 0.7 & 0.2 & 0.1 \end{pmatrix} \end{array}$$

➢Although we can't observe the state (temperature) in the past, we can observe the size of tree rings

- Since the underlying states are hidden, this type of system is known as a **hidden Markov model** (**HMM**)

HMM Model

- **Transition matrix**: $A = \begin{pmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{pmatrix}$

$$\begin{array}{c c} & \begin{array}{cc} H & C \end{array} \\ \begin{array}{c} H \\ C \end{array} & \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix} \end{array}$$

- **Observation matrix**: $B = \begin{pmatrix} 0.1 & 0.4 & 0.5 \\ 0.7 & 0.2 & 0.1 \end{pmatrix}$

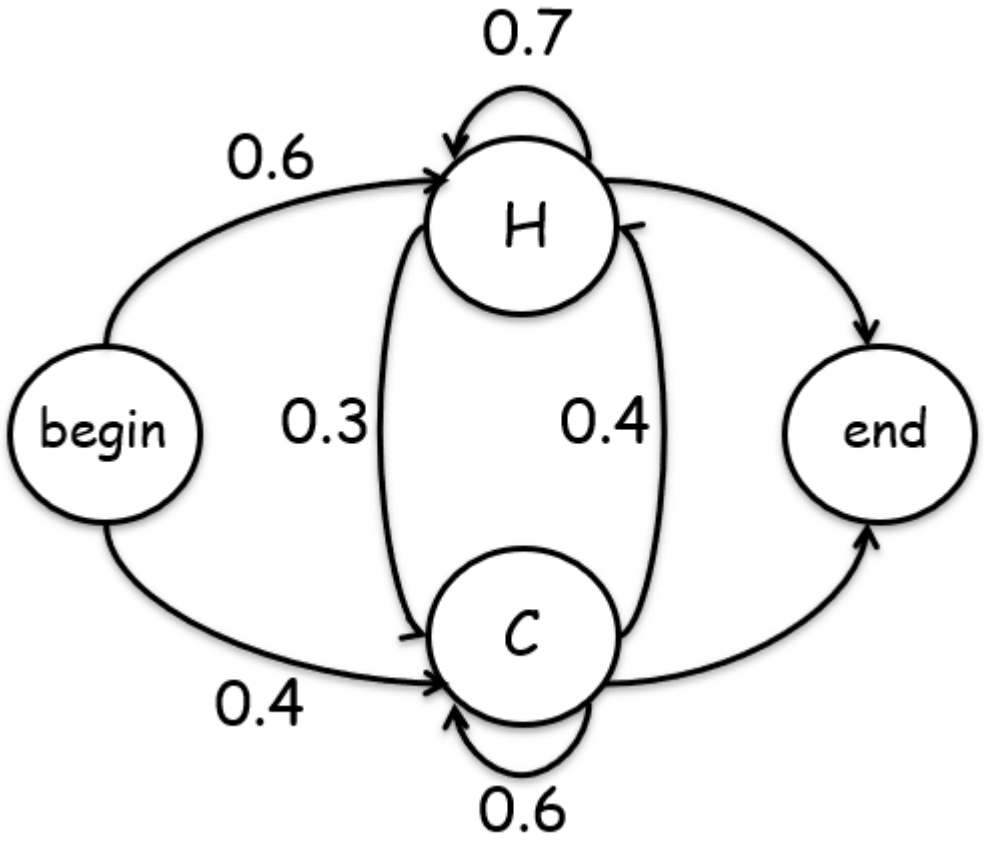
$$\begin{array}{c c} & \begin{array}{ccc} S & M & L \end{array} \\ \begin{array}{c} H \\ C \end{array} & \begin{bmatrix} 0.1 & 0.4 & 0.5 \\ 0.7 & 0.2 & 0.1 \end{bmatrix} \end{array}$$

- **Initial state distribution**: $\pi = \begin{pmatrix} 0.6 & 0.4 \end{pmatrix}$

$$\pi = \begin{array}{c} \begin{array}{cc} H & C \end{array} \\ \begin{bmatrix} 0.6 & 0.4 \end{bmatrix} \end{array}$$

The matrices are row stochastic (discrete probability

HMM Model



$$\pi = \begin{pmatrix} 0.6 & 0.4 \end{pmatrix} \qquad \pi = \begin{array}{cc} H & C \\ \end{array} \begin{bmatrix} 0.6 & 0.4 \end{bmatrix}$$

- The initial distribution matrix indicates the chance that we start in the H state is 0.6 and the chance that we start in the C state is 0.4

HMM Example

- Now, suppose that we consider a particular four-year period of interest from the distant past

- For this particular four-year period, we observe the series of tree ring sizes **S, M, S, L**. Letting:

  0 represent S

  1 represent M

  2 represent L

$$\mathcal{O} = (0, 1, 0, 2)$$

HMM Example

- We might want to determine the most likely state sequence of the Markov process given these observations

  That is, we might want to know the ***most likely*** average annual temperatures over this four-year period of interest

But what "*most likely*" means?

1. We could define "*most likely*" as <u>the state sequence with the highest probability</u> from among all possible state sequences of length 4

   Dynamic programming (DP) can be used to efficiently solve this problem

2. Or, "*most likely*" could be <u>the state sequence that maximizes the expected number of correct states</u>

➢An **HMM** can be used to find the most likely hidden state sequence in this latter sense

HMM Example

- It's important to realize that the **DP** and **HMM** solutions to this problem are not necessarily the same

- For example, the DP solution must, by definition, include valid state transitions, while this is not the case for the HMM

  And even if all state transitions are valid, the HMM solution can still differ from the DP solution, as we'll illustrate in an example

The Notation

Notation	Explanation
T	Length of the observation sequence
N	Number of states in the model
M	Number of observation symbols
Q	Distinct states of the Markov process, $q_0, q_1, \ldots, q_{N-1}$
V	Possible observations, assumed to be $0, 1, \ldots, M-1$
A	State transition probabilities
B	Observation probability matrix
π	Initial state distribution
$\mathcal{O}$	Observation sequence, $\mathcal{O}_0, \mathcal{O}_1, \ldots, \mathcal{O}_{T-1}$

HMM Notation

Note that for simplicity, observations taken from **V = {0,1,…,M−1}**

That is:

$$\mathcal{O}_i \in V \text{ for } i = 0, 1, \ldots, T-1$$

- The transition matrix **A = {a$_{ij}$}** is **N x N**, where

$$a_{ij} = P(\text{state } q_j \text{ at } t+1 \,|\, \text{state } q_i \text{ at } t)$$

- The observation matrix **B = {b$_j$(k)}** is **N x M** where

$$b_j(k) = P(\text{observation } k \text{ at } t \,|\, \text{state } q_j \text{ at } t).$$

V = possible observations

N = # of states

M = # of observation symbols

q = state in the Markov process

$\mathcal{O}$ = observation sequence

T = length of $\mathcal{O}$

Consider our temperature example…

- What are the observations?

$V = \{0,1,2\}$, corresponding to S,M,L

- What are states of Markov process?

$Q = \{H,C\}$

- What are A,B, π, and T?

A,B, π on previous slides

T is number of tree rings measured

- What are N and M?

$N = 2$ and $M = 3$

V = possible observations

M = # observation symbols

N = # of states

M = # of observation symbols

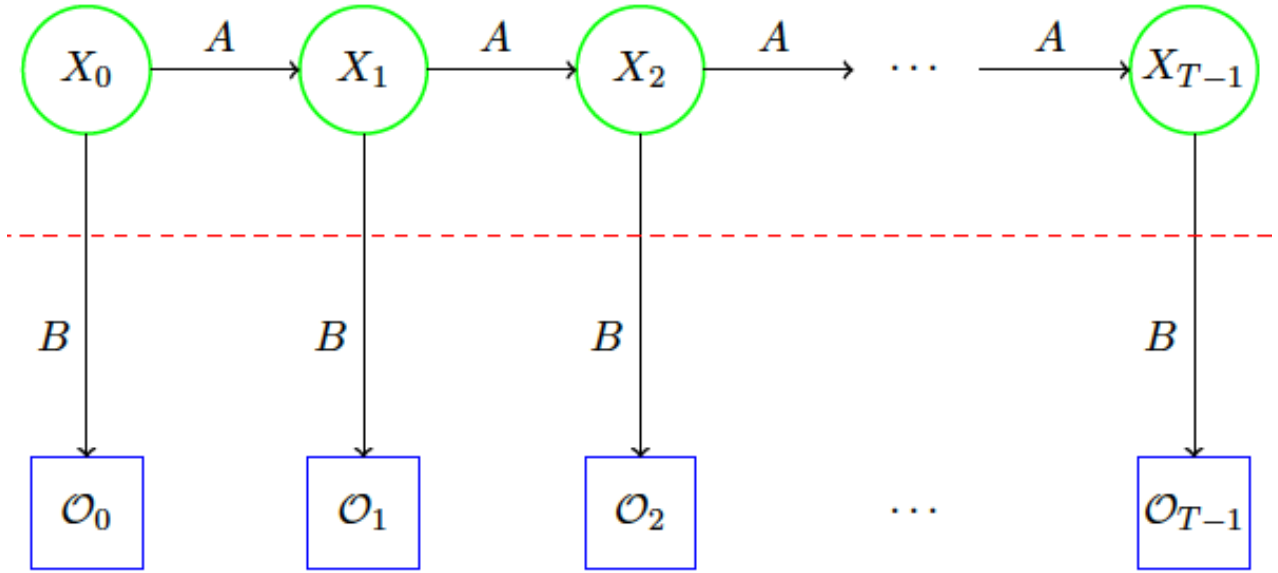
q = state in the Markov process

$\mathcal{O}$ = observation sequence

T = length of $\mathcal{O}$

Generic HMM

Generic view of HMM



HMM defined by **A**,**B**, and **π**

- We denote HMM "model" as **λ = (A,B,π)**

HMM Example

- Imagine that you have a sequence of states $X = (X_0, X_1, X_2, X_3)$
- We want to compute $P(X)$

  That is, how likely is to observe this exact sequence of states

$$P(X) = \pi_{x_0} b_{x_0}(\mathcal{O}_0) a_{x_0,x_1} b_{x_1}(\mathcal{O}_1) a_{x_1,x_2} b_{x_2}(\mathcal{O}_2) a_{x_2,x_3} b_{x_3}(\mathcal{O}_3)$$

Where:

- π_{x_0} is the probability of starting in state X_0
- $b_{x_0}(\mathcal{O}_0)$ is the probability of initial observation
- a_{x_0,x_1} is the probability of transition X_0 to X_1

HMM Example

$$P(X) = \pi_{x_0} b_{x_0}(\mathcal{O}_0) a_{x_0,x_1} b_{x_1}(\mathcal{O}_1) a_{x_1,x_2} b_{x_2}(\mathcal{O}_2) a_{x_2,x_3} b_{x_3}(\mathcal{O}_3)$$

Let's Assume that we observe: (0, 1, 0, 2) or (S, M, S, L)

- What is probability of: H H C C ?

$$P(HHCC) = 0.6(0.1)(0.7)(0.4)(0.3)(0.7)(0.6)(0.1) = 0.000212.$$

$$\pi = \begin{pmatrix} 0.6 & 0.4 \end{pmatrix} \qquad A = \begin{pmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{pmatrix} \qquad B = \begin{pmatrix} 0.1 & 0.4 & 0.5 \\ 0.7 & 0.2 & 0.1 \end{pmatrix}$$

Initial state distribution **Transition matrix** **Observation matrix**

HMM Example

$$P(X) = \pi_{x_0} b_{x_0}(\mathcal{O}_0) a_{x_0,x_1} b_{x_1}(\mathcal{O}_1) a_{x_1,x_2} b_{x_2}(\mathcal{O}_2) a_{x_2,x_3} b_{x_3}(\mathcal{O}_3)$$

Let's Assume that we observe: (0, 1, 0, 2) or (S, M, S, L)

- What is probability of: H H C C ?

$$P(HHCC) = 0.6(0.1)(0.7)(0.4)(0.3)(0.7)(0.6)(0.1) = 0.000212.$$

$$\pi = \begin{array}{cc} H & C \\ \end{array} \left[\begin{array}{cc} 0.6 & 0.4 \end{array} \right]$$

Initial state distribution

$$\begin{array}{c|cc} & H & C \\ \hline H & 0.7 & 0.3 \\ C & 0.4 & 0.6 \end{array}$$

Transition matrix A

$$\begin{array}{c|ccc} & S & M & L \\ \hline H & 0.1 & 0.4 & 0.5 \\ C & 0.7 & 0.2 & 0.1 \end{array}$$

Observation matrix B

HMM Example

- We repeat it for all the $2^4 = 16$ combinations of 4-state sequences
- We find that the winner is: CCCH
- This is the result applying **Dynamic Programming** (DP)

  Basically, just picking the maximum
- But HMM works differently…

state	probability	normalized probability
$HHHH$	.000412	.042787
$HHHC$	.000035	.003635
$HHCH$	.000706	.073320
$HHCC$	.000212	.022017
$HCHH$	.000050	.005193
$HCHC$	.000004	.000415
$HCCH$	.000302	.031364
$HCCC$	.000091	.009451
$CHHH$	.001098	.114031
$CHHC$	.000094	.009762
$CHCH$	.001882	.195451
$CHCC$	.000564	.058573
$CCHH$	.000470	.048811
$CCHC$	.000040	.004154
$CCCH$	.002822	.293073
$CCCC$	.000847	.087963

HMM Example

- To find the optimal state sequence in the **HMM** sense, we choose **the most probable symbol at each position**

- So, we sum the normalized probabilities that have an H in the first position

- The sum is equal to:

0.188182

state	probability	normalized probability
$HHHH$	.000412	.042787
$HHHC$	.000035	.003635
$HHCH$	.000706	.073320
$HHCC$	.000212	.022017
$HCHH$	.000050	.005193
$HCHC$	.000004	.000415
$HCCH$	.000302	.031364
$HCCC$	.000091	.009451
$CHHH$	.001098	.114031
$CHHC$	.000094	.009762
$CHCH$	.001882	.195451
$CHCC$	.000564	.058573
$CCHH$	.000470	.048811
$CCHC$	.000040	.004154
$CCCH$	.002822	.293073
$CCCC$	.000847	.087963

SUM

0.188181

HMM Example

- We repeat the same for C in the first position

- The normalized sum in this case is:

0.811818

state	probability	normalized probability
$HHHH$	.000412	.042787
$HHHC$	.000035	.003635
$HHCH$	.000706	.073320
$HHCC$	.000212	.022017
$HCHH$	.000050	.005193
$HCHC$	.000004	.000415
$HCCH$	.000302	.031364
$HCCC$	.000091	.009451
$CHHH$	.001098	.114031
$CHHC$	.000094	.009762
$CHCH$	.001882	.195451
$CHCC$	.000564	.058573
$CCHH$	.000470	.048811
$CCHC$	.000040	.004154
$CCCH$	.002822	.293073
$CCCC$	.000847	.087963

SUM

0.188181

SUM

0.811818

HMM Example

- We continue, now with all the cases where H is in second position:

0.519576

- And C in second position:

0.480424

And so on…

state	probability	normalized probability
$HHHH$	.000412	.042787
$HHHC$	.000035	.003635
$HHCH$	.000706	.073320
$HHCC$	.000212	.022017
$HCHH$	.000050	.005193
$HCHC$	.000004	.000415
$HCCH$	.000302	.031364
$HCCC$	.000091	.009451
$CHHH$	.001098	.114031
$CHHC$	.000094	.009762
$CHCH$	.001882	.195451
$CHCC$	.000564	.058573
$CCHH$	.000470	.048811
$CCHC$	.000040	.004154
$CCCH$	.002822	.293073
$CCCC$	.000847	.087963

SUM

0.519576

SUM

0.480424

HMM Example

- We continue with all the other positions, until we built a table like this one:

	Position in state sequence			
	0	1	2	3
$P(H)$	0.188182	0.519576	0.228788	0.804029
$P(C)$	0.811818	0.480424	0.771212	0.195971

- Finally, for each position, we select the most likely symbol. That is, our sequence (in HMM terms) would be:

CHCH

HMM Example

- **HMM** solution gives us **CHCH**
- While **DP** solution is **CCCH**

Which solution is better?

- Neither solution is better

  Just using different definitions of “best”

HMM Paradox?

- **HMM** maximizes ***expected number of correct states***

  Whereas DP chooses “best” overall path

- <u>Possible</u> for **HMM** to choose a “path” that is <u>impossible</u>

  Could be a transition probability of 0

- We cannot get impossible path with DP

  Is this a flaw with HMM?

  No, it’s a feature

The Three Problems

The Three Problems

HMMs are used to solve 3 problems:

- **Problem 1**: Given a model $\boldsymbol{\lambda} = (\mathbf{A},\mathbf{B},\boldsymbol{\pi})$ and observation sequence $\mathbf{O}$, find $\mathbf{P(O|\lambda)}$
  - We *score* an observation sequence to see how well it fits a given model

- **Problem 2**: Given $\boldsymbol{\lambda} = (\mathbf{A},\mathbf{B},\boldsymbol{\pi})$ and $\mathbf{O}$, find an optimal state sequence (in HMM sense)
  - Uncover hidden part (like previous example)

- **Problem 3**: Given $\mathbf{O}$, $\mathbf{N}$ (# states), and $\mathbf{M}$ (# of symbols), find the model $\boldsymbol{\lambda}$ that maximizes probability of $\mathbf{O}$
  - That is, ***train*** a model to fit observations

HMMs in Practice

Often, we use HMMs as follows:

1. We compute an observation sequence
   - We also assume that (hidden) Markov process exists

2. We train a model based on observations
   - That is, we solve Problem 3
   - "Best" N (# states) can be found by trial and error

3. Then given a (usually different, but '*compatible*') sequence of observations, we score it versus the model we trained
   - This is Problem 1 — high score implies similar to training data, low score says it's not similar

HMMs in Practice

Previous slide gives sense in which HMM is a "<u>machine learning</u>" technique

- To train model, we do not need to specify anything except the parameter N
- "Best" N often found by trial and error (like a "hyperparameter")

So, we don't need to think too much

- Just train HMM and then use it
- Practical, since we have efficient algorithms for training HMM and scoring

The Three Solutions

- We give detailed solutions to 3 problems

  However, we must find ***efficient*** solutions

The 3 problems:

- **Problem 1**: Score an observation sequence versus a given model
- **Problem 2**: Given a model, "uncover" *most likely* hidden part
- **Problem 3**: Given an observation sequence, train a model

Recall that we considered example for 2 and 1 in the previous slides

But direct solutions are soooooo inefficient

Solution 1

- Score observations versus a given model
  - ➢Given model $\boldsymbol{\lambda} = (\mathbf{A},\mathbf{B},\boldsymbol{\pi})$ and observation sequence $\mathbf{O}=(\mathbf{O}_0,\mathbf{O}_1,\ldots,\mathbf{O}_{T-1})$, find $\mathbf{P}(\mathbf{O}|\boldsymbol{\lambda})$

Denote hidden states as: $X = (X_0, X_1, \ldots, X_{T-1})$

And from definition of A and π: $P(X|\lambda) = \pi_{X0}\, a_{X0,X1}\, a_{X1,X2} \ldots a_{XT-2,XT-1}$

Then from definition of B: $P(O|X,\lambda) = b_{X0}(O_0)\, b_{X1}(O_1) \ldots b_{XT-1}(O_{T-1})$

$$\pi = \begin{array}{c} \begin{array}{cc} H & C \end{array} \\ \begin{bmatrix} 0.6 & 0.4 \end{bmatrix} \end{array}$$

$$A = \begin{array}{c|cc} & H & C \\ \hline H & 0.7 & 0.3 \\ C & 0.4 & 0.6 \end{array}$$

$$B = \begin{array}{c|ccc} & S & M & L \\ \hline H & 0.1 & 0.4 & 0.5 \\ C & 0.7 & 0.2 & 0.1 \end{array}$$

Remember:

$$P(A \text{ and } B) = P(A, B) = P(A \cap B)$$

$$P(B \mid A) = \frac{P(A \text{ and } B)}{P(A)}$$

- Since:

$$P(\mathcal{O}, X \mid \lambda) = \frac{P(\mathcal{O} \cap X \cap \lambda)}{P(\lambda)}$$

- And:

$$P(\mathcal{O} \mid X, \lambda) P(X \mid \lambda) = \frac{P(\mathcal{O} \cap X \cap \lambda)}{P(X \cap \lambda)} \cdot \frac{P(X \cap \lambda)}{P(\lambda)} = \frac{P(\mathcal{O} \cap X \cap \lambda)}{P(\lambda)}$$

- We have:

$$P(\mathcal{O}, X \mid \lambda) = P(\mathcal{O} \mid X, \lambda) P(X \mid \lambda)$$

The problem that we want to solve over all the possible X combinations

Definition of B times definition of A and π

Solution 1

- The probability $P(\mathcal{O} \mid \lambda)$ is given summing together all possible state sequence probabilities:

$$
\begin{aligned}
P(\mathcal{O} \mid \lambda) &= \sum_X P(\mathcal{O}, X \mid \lambda) \\
&= \sum_X P(\mathcal{O} \mid X, \lambda) P(X \mid \lambda) \\
&= \sum_X \pi_{X_0} b_{X_0}(\mathcal{O}_0) a_{X_0, X_1} b_{X_1}(\mathcal{O}_1) \cdots a_{X_{T-2}, X_{T-1}} b_{X_{T-1}}(\mathcal{O}_{T-1})
\end{aligned}
$$

Definition of B times definition of A and π

$$P(X) = \pi_{x_0} b_{x_0}(\mathcal{O}_0) a_{x_0, x_1} b_{x_1}(\mathcal{O}_1) a_{x_1, x_2} b_{x_2}(\mathcal{O}_2) a_{x_2, x_3} b_{x_3}(\mathcal{O}_3)$$

- The direct computation here is generally <u>infeasible</u>

  Since the number of multiplications is about $2TN^T$

  Where T is typically large and N ≥ 2

Forward Algorithm

- Instead, we can use ***forward algorithm***
  Also known as: "**alpha pass**"

Instead of $\mathrm{P}\left(\mathcal{O}, \mathrm{X} \middle| \lambda\right)$, that considers all the states, we now stop at the states up to time *t*

- For $\mathbf{t = 0,1,\ldots,T-1}$ and $\mathbf{i=0,1,\ldots,N-1}$, let:

$$\alpha_t(i) = P(\mathcal{O}_0, \mathcal{O}_1, \ldots, \mathcal{O}_t, X_t = q_i \mid \lambda)$$

- This is probability of partial observation sequence up to time **t**, where the underlying Markov process is in state $\mathbf{q_i}$ at time **t**
  Can be computed <u>recursively</u> and efficiently

Forward Algorithm

Given:

Model $\lambda = (A, B, \pi)$

Observations $\mathcal{O} = (O_0, \mathcal{O}_1, \ldots, \mathcal{O}_{T-1})$

for $i = 0, 1, \ldots, N-1$ **do**

$\alpha_0(i) = \pi_i b_i(\mathcal{O}_0)$

end for

for $t = 1, 2, \ldots, T-1$ **do**

for $i = 0, 1, \ldots, N-1$ **do**

$$\alpha_t(i) = \left(\sum_{j=0}^{N-1} \alpha_{t-1}(j) a_{ji} \right) b_i(\mathcal{O}_t)$$

end for

end for

$$\begin{array}{c c} & \begin{array}{cc} H & C \end{array} \\ \begin{array}{c} H \\ C \end{array} & \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix} \end{array}$$

$$\begin{array}{c c} & \begin{array}{ccc} S & M & L \end{array} \\ \begin{array}{c} H \\ C \end{array} & \begin{bmatrix} 0.1 & 0.4 & 0.5 \\ 0.7 & 0.2 & 0.1 \end{bmatrix} \end{array}$$

$$\pi = \begin{array}{c} \begin{array}{cc} H & C \end{array} \\ \begin{bmatrix} 0.6 & 0.4 \end{bmatrix} \end{array}$$

This requires only N^2T multiplications

Forward Algorithm

Given:

Model $\lambda = (A, B, \pi)$

Observations $\mathcal{O} = (O_0, \mathcal{O}_1, \ldots, \mathcal{O}_{T-1})$

This is also called:
Initialization Step

for $i = 0, 1, \ldots, N-1$ **do**

$\alpha_0(i) = \pi_i b_i(\mathcal{O}_0)$

end for

This instead is called:
Induction Step

for $t = 1, 2, \ldots, T-1$ **do**

for $i = 0, 1, \ldots, N-1$ **do**

$$\alpha_t(i) = \left(\sum_{j=0}^{N-1} \alpha_{t-1}(j) a_{ji} \right) b_i(\mathcal{O}_t)$$

end for

end for

$$\begin{array}{c|cc} & H & C \\ \hline H & 0.7 & 0.3 \\ C & 0.4 & 0.6 \end{array}$$

$$\begin{array}{c|ccc} & S & M & L \\ \hline H & 0.1 & 0.4 & 0.5 \\ C & 0.7 & 0.2 & 0.1 \end{array}$$

$$\pi = \begin{array}{cc} H & C \\ \hline 0.6 & 0.4 \end{array}$$

This requires only N^2T multiplications

Forward Algorithm

- From the formula:

$$\alpha_t(i) = P(\mathcal{O}_0, \mathcal{O}_1, \ldots, \mathcal{O}_t, X_t = q_i \mid \lambda)$$

- It follows that:

$$\boxed{P(\mathcal{O} \mid \lambda) = \sum_{i=0}^{N-1} \alpha_{T-1}(i)}$$

Hence, the forward algorithm gives us an efficient way to compute a score for a given sequence O, relative to a given model λ

Solution 2

- Given a model, find hidden states

  Given $\boldsymbol{\lambda} = \mathbf{(A,B,\pi)}$ and $\mathbf{O}$, find an optimal state sequence

  - Recall that optimal means "maximize expected number of correct states"
  - In contrast, DP finds best scoring path

- For temp/tree ring example, we solved this

  But hopelessly inefficient approach

- A better way: ***backward algorithm***

  Also known as: "**beta pass**"

Backward Algorithm

- For $\mathbf{t = 0,1,\ldots,T-1}$ and $\mathbf{i = 0,1,\ldots,N-1}$, let:

$$\beta_t(i) = P(\mathcal{O}_{t+1}, \mathcal{O}_{t+2}, \ldots, \mathcal{O}_{T-1} \mid X_t = q_i, \lambda)$$

- This is probability of partial observation sequence <u>starting from</u> time t, where the underlying Markov process is in state $\mathbf{q_i}$ at time t

- Analogous to the forward algorithm

  As with forward algorithm, this can be computed <u>recursively</u> and efficiently

Backward Algorithm

Given:

Model $\lambda = (A, B, \pi)$

Observations $\mathcal{O} = (O_0, \mathcal{O}_1, \ldots, \mathcal{O}_{T-1})$

for $i = 0, 1, \ldots, N-1$ **do**

$\beta_{T-1}(i) = 1$

end for

for $t = T-2, T-3, \ldots, 0$ **do**

for $i = 0, 1, \ldots, N-1$ **do**

$$\beta_t(i) = \sum_{j=0}^{N-1} a_{ij} b_j(\mathcal{O}_{t+1}) \beta_{t+1}(j)$$

end for

end for

$$\begin{array}{c c} & \begin{array}{cc} H & C \end{array} \\ \begin{array}{c} H \\ C \end{array} & \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix} \end{array}$$

$$\begin{array}{c c} & \begin{array}{ccc} S & M & L \end{array} \\ \begin{array}{c} H \\ C \end{array} & \begin{bmatrix} 0.1 & 0.4 & 0.5 \\ 0.7 & 0.2 & 0.1 \end{bmatrix} \end{array}$$

$$\pi = \begin{array}{c} \begin{array}{cc} H & C \end{array} \\ \begin{bmatrix} 0.6 & 0.4 \end{bmatrix} \end{array}$$

Note: it goes backward now

Solution 2

- Now, for **t = 0,1,…,T−1** and **i=0,1,…,N−1** define:

$$\gamma_t(i) = P(X_t = q_i \mid \mathcal{O}, \lambda)$$

Most likely state at **t** is $\mathbf{q_i}$ that maximizes $\boldsymbol{\gamma_t(\mathbf{i})}$

➢Since $\alpha_t(i)$ measures the relevant probability up to time t and $\beta_t(i)$ measures the relevant probability after time t, we have:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(\mathcal{O} \mid \lambda)}$$

And recall: $$P(\mathcal{O} \mid \lambda) = \sum_{i=0}^{N-1} \alpha_{T-1}(i)$$

Solution 2

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(\mathcal{O} \mid \lambda)}$$

Why is it necessary to normalize gamma by dividing by P(O|λ)?

- Because these probabilities are computed assuming the observation sequence is known (given O), as opposed to being computed relative to the larger probability space

Solution 2

- From the definition of $\gamma_t(i)$ it follows that the most likely state at time **t** is the state for which $\boldsymbol{\gamma_t(i)}$ is maximum, where the maximum is taken over the index i

  Then the most likely state at time t is given by:

$$\widetilde{X}_t = \max_i \gamma_t(i)$$

- The bottom line?
  - Forward algorithm solves Problem 1
  - Forward/backward algorithms solve Problem 2

Solution 3

- Train a model: Given **O**, **N**, and **M**, find **λ** that maximizes probability of **O**

- We'll <u>iteratively adjust</u> $\boldsymbol{\lambda} = (\mathbf{A},\mathbf{B},\boldsymbol{\pi})$ to better fit the given observations **O**
  - The size of matrices are fixed (N and M)
  - But elements of matrices can change

- It is nice that this works…

  …and amazing that it's efficient!

Solution 3

- For **t=0,1,…,T−2** and **i,j in {0,1,…,N−1}**, define "**di-gammas**" as:

$$\gamma_t(i,j) = P(X_t = q_i, X_{t+1} = q_j \mid \mathcal{O}, \lambda)$$

➢Note: $\boldsymbol{\gamma t(i,j)}$ is probability of being in state $\mathbf{q_i}$ at time **t** and transiting to state $\mathbf{q_j}$ at **t+1**

Then we can rewr

$$\gamma_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(\mathcal{O}_{t+1}) \beta_{t+1}(j)}{P(\mathcal{O} \mid \lambda)}$$

And:

$$\gamma_t(i) = \sum_{j=0}^{N-1} \gamma_t(i,j)$$

Solution 3

$$\pi = \begin{array}{c} \begin{array}{cc} H & C \end{array} \\ \begin{bmatrix} \quad & \quad \end{bmatrix} \end{array}$$

$$\begin{array}{c|cc} & H & C \\ \hline H & \multicolumn{2}{c}{\multirow{2}{*}{$\begin{bmatrix} \quad & \quad \\ \quad & \quad \end{bmatrix}$}} \\ C & & \end{array}$$

$$\begin{array}{c|ccc} & S & M & L \\ \hline H & \multicolumn{3}{c}{\multirow{2}{*}{$\begin{bmatrix} \quad & \quad & \quad \\ \quad & \quad & \quad \end{bmatrix}$}} \\ C & & & \end{array}$$

Given:

$\gamma_t(i)$, for $t = 0, 1, \ldots, T-1$ and $i = 0, 1, \ldots, N-1$

$\gamma_t(i,j)$, for $t = 0, 1, \ldots, T-2$ and $i, j \in \{0, 1, \ldots, N-1\}$

for $i = 0, 1, \ldots, N-1$ **do**

  $\pi_i = \gamma_0(i)$

end for

for $i = 0, 1, \ldots, N-1$ **do**

  for $j = 0, 1, \ldots, N-1$ **do**

    $$a_{ij} = \sum_{t=0}^{T-2} \gamma_t(i,j) \Big/ \sum_{t=0}^{T-2} \gamma_t(i)$$

  end for

end for

for $j = 0, 1, \ldots, N-1$ **do**

  for $k = 0, 1, \ldots, M-1$ **do**

    $$b_j(k) = \sum_{\substack{t \in \{0,1,\ldots,T-1\} \\ \mathcal{O}_t = k}} \gamma_t(j) \Big/ \sum_{t=0}^{T-1} \gamma_t(j)$$

  end for

end for

These two sums are very similar, but in the nominator we consider only **t** for which $\mathbf{O_t} = \mathbf{k}$ and only these values are counted in the numerator

That is, per each individual symbol [0, M-1] that appears in O

Solution 3

Given:

$\gamma_t(i)$, for $t = 0, 1, \ldots, T-1$ and $i = 0, 1, \ldots, N-1$

$\gamma_t(i,j)$, for $t = 0, 1, \ldots, T-2$ and $i, j \in \{0, 1, \ldots, N-1\}$

for $i = 0, 1, \ldots, N-1$ **do**

  $\pi_i = \gamma_0(i)$

end for

for $i = 0, 1, \ldots, N-1$ **do**

  for $j = 0, 1, \ldots, N-1$ **do**

    $$a_{ij} = \sum_{t=0}^{T-2} \gamma_t(i,j) \Big/ \sum_{t=0}^{T-2} \gamma_t(i)$$

  end for

end for

for $j = 0, 1, \ldots, N-1$ **do**

  for $k = 0, 1, \ldots, M-1$ **do**

    $$b_j(k) = \boxed{\sum_{\substack{t \in \{0,1,\ldots,T-1\} \\ \mathcal{O}_t = k}} \gamma_t(j)} \Big/ \boxed{\sum_{t=0}^{T-1} \gamma_t(j)}$$

  end for

end for

- The <u>numerator</u> of the re-estimated $\boldsymbol{b_j(k)}$ is the expected number of times the model is in state q_j with observation k
- While the <u>denominator</u> is the expected number of times the model is in state q_j
- Therefore, the <u>ratio</u> is the probability of observing symbol k, given that the model is in state q_j, and this is the desired value for $\boldsymbol{b_j(k)}$

Model Re-estimation

Re-estimation is an iterative process

- First, we **<u>initialize</u>** $\lambda = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi})$ with a reasonable guess, or, if no reasonable guess is available, we choose random values such that
  - $\boldsymbol{\pi} \approx \frac{\mathbf{1}}{\boldsymbol{N}}$
  - $\mathbf{a_{ij}} \approx \frac{\mathbf{1}}{\boldsymbol{N}}$
  - $\mathbf{b_j(k)} \approx \frac{\mathbf{1}}{\boldsymbol{M}}$

It's critical that A, B and π are <u>randomized</u>

- Since exactly uniform values will result in a **local maximum** from which the model cannot climb

Solution 3

To summarize:

1. Initialize $\lambda = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi})$
2. Compute $\boldsymbol{\alpha t}(\boldsymbol{i}),\ \boldsymbol{\beta t}(\boldsymbol{i}),\ \boldsymbol{\gamma t}(\boldsymbol{i}, \boldsymbol{j}),\ \boldsymbol{\gamma t}(\boldsymbol{i})$
3. Re-estimate the model $\lambda = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi})$
4. If $\boldsymbol{P}(\mathcal{O} \mid \lambda)$ increases by more than **ε** (where **ε** is small), goto 2

Solution 3

<u>Model initialization</u>

- If we have a good guess for $\boldsymbol{\lambda} = \mathbf{(A,B,\pi)}$ then we can use it for initialization
- If not, let: $\boldsymbol{\pi i} \approx \dfrac{\mathbf{1}}{\boldsymbol{N}}$ $\quad \boldsymbol{aij} \approx \dfrac{\mathbf{1}}{\boldsymbol{N}}$ $\quad \boldsymbol{bj}(\boldsymbol{k}) \approx \dfrac{\mathbf{1}}{\boldsymbol{M}}$
- Subject to row stochastic conditions
- But do ***not*** initialize to exactly uniform values

<u>Stopping conditions</u>

- Stop after some number of iterations and/or…
- Stop if increase in $\boldsymbol{P}(\mathcal{O} \mid \lambda)$ is too small (less than ϵ)

HMM as Discrete Hill Climb

- Algorithm on previous slides shows that **HMM** is a "**discrete hill climb**"

- HMM consists of discrete states $\mathbf{X_t}$

  "Climb" on the elements of the matrices

- And re-estimation process improves model by modifying parameters

  So, "climbs" toward improved model

  This happens in a high-dimensional space

Dynamic Programming

Brief detour to show the close relationship between dynamic programming (DP) and HMMs

- For $\boldsymbol{\lambda = (A,B,\pi)}$ it's easy to define a dynamic program

  DP is like the forward algorithm (alpha-pass) but with "sum" replaced by "<u>max</u>"

Dynamic Programming

- Here we see the **dynamic programming algorithm**, which is also known as the Viterbi algorithm

Given:
Model $\lambda = (A, B, \pi)$
Observations $\mathcal{O} = (O_0, \mathcal{O}_1, \ldots, \mathcal{O}_{T-1})$
for $i = 0, 1, \ldots, N-1$ **do**
$\delta_0(i) = \pi_i b_i(\mathcal{O}_0)$
end for
for $t = 1, 2, \ldots, T-1$ **do**
for $i = 0, 1, \ldots, N-1$ **do**
$\delta_t(i) = \max_{j \in \{0,1,\ldots,N-1\}} \left(\delta_{t-1}(j) a_{ji} b_i(\mathcal{O}_t)\right)$
end for
end for

- Note that at each t, the DP computes best path for each state, up to that point

  So, probability of best path is:

$$\max_{j \in \{0,1,\ldots,N-1\}} \delta_{T-1}(j)$$

- This max gives the highest probability, but not the best path

Dynamic Programming

$$\max_{j \in \{0,1,\ldots,N-1\}} \delta_{T-1}(j)$$

- It is important to realize that this formula only gives the optimal probability, not the corresponding path

- To determine optimal path:
  1. While computing deltas, keep track of pointers to previous state
  2. When finished, construct optimal path by tracing back points

Dynamic Programming

- For example, let's consider again the temperature example

  Recall that we observe (0,1,0,2)

$$\begin{array}{c c} & \begin{array}{cc} H & C \end{array} \\ \begin{array}{c} H \\ C \end{array} & \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix} \end{array}$$

$$\begin{array}{c c} & \begin{array}{ccc} S & M & L \end{array} \\ \begin{array}{c} H \\ C \end{array} & \begin{bmatrix} 0.1 & 0.4 & 0.5 \\ 0.7 & 0.2 & 0.1 \end{bmatrix} \end{array}$$

$$\pi = \begin{array}{c} \begin{array}{cc} H & C \end{array} \\ \begin{bmatrix} 0.6 & 0.4 \end{bmatrix} \end{array}$$

- The initial probabilities (path of length 1) are:

$$P(H) = \pi_0 b_0(0) = 0.6(0.1) = 0.06$$

$$P(C) = \pi_1 b_1(0) = 0.4(0.7) = 0.28$$

- The probabilities of the paths of length two are given by:

$$P(HH) = 0.06(0.7)(0.4) = 0.0168$$

Dynamic Programming

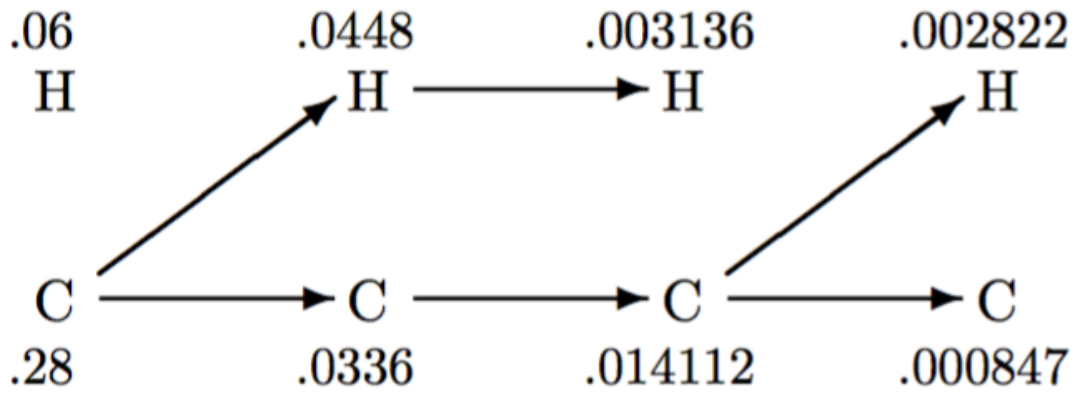
- Probabilities for each path of length 2

$$
\begin{aligned}
P(HH) &= 0.06(0.7)(0.4) = 0.0168 \\
P(HC) &= 0.06(0.3)(0.2) = 0.0036 \\
P(CH) &= 0.28(0.4)(0.4) = 0.0448 \\
P(CC) &= 0.28(0.6)(0.2) = 0.0336
\end{aligned}
$$

  - Best path of length 2 ending with H is CH
  - Best path of length 2 ending with C is CC

Dynamic Program

1. Continuing, we compute best path ending at H and C at each step
2. And then we save pointers
   - Note that at each stage, the dynamic programming algorithm only needs to maintain the highest-scoring path ending at each state—not a list of all possible paths (this is the key to the efficiency of the algorithm)



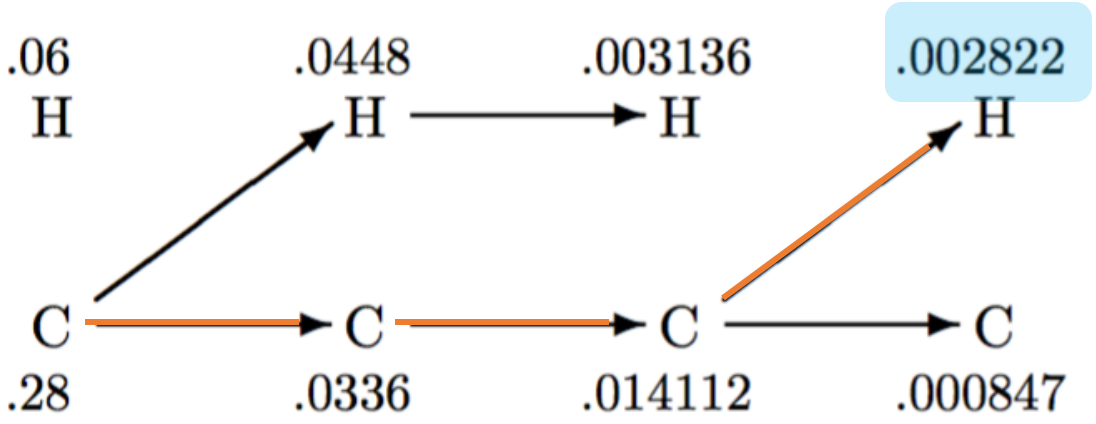
Dynamic Program

- Best final score is .002822

  And thanks to pointers, best path is **CCCH**

- But what about underflow?

  A serious problem in bigger cases



state	probability
$HHHH$	.000412
$HHHC$	.000035
$HHCH$	.000706
$HHCC$	.000212
$HCHH$	.000050
$HCHC$	.000004
$HCCH$	.000302
$HCCC$	.000091
$CHHH$	.001098
$CHHC$	.000094
$CHCH$	.001882
$CHCC$	.000564
$CCHH$	.000470
$CCHC$	.000040
$CCCH$	.002822
$CCCC$	.000847

Underflow Resistant DP

- Common trick to prevent underflow:

  Instead of multiplying probabilities…

  - We **add** logarithms of probabilities

- Why does this work?

  Because $\log(xy) = \log x + \log y$

  ➢Adding logs does not tend to 0

- Note that these logs are negative… and we must avoid 0 probabilities

Underflow Resistant DP

Given:

Model $\lambda = (A, B, \pi)$

Observations $\mathcal{O} = (O_0, \mathcal{O}_1, \ldots, \mathcal{O}_{T-1})$

for $i = 0, 1, \ldots, N-1$ **do**

$\quad \widehat{\delta}_0(i) = \log\big(\pi_i b_i(\mathcal{O}_0)\big)$

end for

for $t = 1, 2, \ldots, T-1$ **do**

$\quad$ **for** $i = 0, 1, \ldots, N-1$ **do**

$$\widehat{\delta}_t(i) = \max_{j \in \{0,1,\ldots,N-1\}} \Big(\widehat{\delta}_{t-1}(j) + \log(a_{ji}) + \log\big(b_i(\mathcal{O}_t)\big)\Big)$$

$\quad$ **end for**

end for

- Again, the optimal score is given by: $\max_{j \in \{0,1,\ldots,N-1\}} \widehat{\delta}_{T-1}(j)$

HMM Scaling

- The three HMM solutions all involve products of probabilities

  It's very easy to see, for example, that $\alpha_t(i)$ tends to 0 exponentially as T increases

$$
\begin{array}{l}
\textbf{for } t = 1, 2, \ldots, T-1 \textbf{ do} \\
\quad \textbf{for } i = 0, 1, \ldots, N-1 \textbf{ do} \\
\qquad \alpha_t(i) = \left(\sum_{j=0}^{N-1} \alpha_{t-1}(j) a_{ji} \right) b_i(\mathcal{O}_t) \\
\quad \textbf{end for} \\
\textbf{end for}
\end{array}
$$

- Therefore, any attempt to implement the HMM algorithms will inevitably result in **underflow**

HMM Scaling

- The <u>solution</u> to this underflow problem is to scale the numbers

  However, care must be taken to ensure that the algorithms remain valid

First, consider the computation of $\alpha_t(i)$

- The basic recurrence is:

$$\alpha_t(i) = \sum_{j=0}^{N-1} \alpha_{t-1}(j) a_{ji} b_i(\mathcal{O}_t)$$

- It seems sensible to normalize each $\alpha_t(i)$ by dividing by: $\sum_{j=0}^{N-1} \alpha_t(j)$

HMM Scaling

- Following this approach, we compute **scaling factors** $\boldsymbol{c_t}$ and the scaled $\alpha_t(i)$, which we denote as $\hat{\boldsymbol{\alpha}}_{\boldsymbol{t}}(\boldsymbol{i})$

- To verify, we first note that $\hat{\alpha}_0(i) = c_0\alpha_0(i)$

Next slide…

- Now suppose that for some **t** we have:

$$\boxed{\widehat{\alpha}_t(i) = c_0 c_1 \cdots c_t \alpha_t(i)}$$

- Then:

$$\begin{aligned}
\widehat{\alpha}_{t+1}(i) &= c_{t+1} \boxed{\widetilde{\alpha}_{t+1}(i)} \\
&= c_{t+1} \boxed{\sum_{j=0}^{N-1} \boxed{\widehat{\alpha}_t(j)} a_{ji} b_i(\mathcal{O}_{t+1})} \\
&= \boxed{c_0 c_1 \cdots c_t} c_{t+1} \boxed{\sum_{j=0}^{N-1} \boxed{\alpha_t(j)} a_{ji} b_i(\mathcal{O}_{t+1})} \\
&= c_0 c_1 \cdots c_{t+1} \boxed{\alpha_{t+1}(i)}
\end{aligned}$$

- And hence $\boxed{\widehat{\alpha}_t(i) = c_0 c_1 \cdots c_t \alpha_t(i)}$ holds, **by induction**, for all t

Given:

$\alpha_t(i)$, for $t = 0, 1, \ldots, T-1$ and $i = 0, 1, \ldots, N-1$

for $i = 0, 1, \ldots, N-1$ **do**

$\widetilde{\alpha}_0(i) = \alpha_0(i)$

end for

$$c_0 = 1 / \sum_{j=0}^{N-1} \widetilde{\alpha}_0(j)$$

for $i = 0, 1, \ldots, N-1$ **do**

$\widehat{\alpha}_0(i) = c_0 \widetilde{\alpha}_0(i)$

end for

for $t = 1, 2, \ldots, T-1$ **do**

for $i = 0, 1, \ldots, N-1$ **do**

$$\widetilde{\alpha}_t(i) = \sum_{j=0}^{N-1} \widehat{\alpha}_{t-1}(j) a_{ji} b_i(\mathcal{O}_t)$$

end for

$$c_t = 1 \Big/ \sum_{j=0}^{N-1} \widetilde{\alpha}_t(j)$$

for $i = 0, 1, \ldots, N-1$ **do**

$\widehat{\alpha}_t(i) = c_t \widetilde{\alpha}_t(i)$

end for

end for

Given:

$\alpha_t(i)$, for $t = 0, 1, \ldots, T-1$ and $i = 0, 1, \ldots, N-1$

for $i = 0, 1, \ldots, N-1$ **do**

$\quad \widetilde{\alpha}_0(i) = \alpha_0(i)$

end for

$c_0 = 1 / \sum_{j=0}^{N-1} \widetilde{\alpha}_0(j)$

for $i = 0, 1, \ldots, N-1$ **do**

$\quad \widehat{\alpha}_0(i) = c_0 \widetilde{\alpha}_0(i)$

end for

for $t = 1, 2, \ldots, T-1$ **do**

$\quad$ **for** $i = 0, 1, \ldots, N-1$ **do**

$\quad\quad \widetilde{\alpha}_t(i) = \sum_{j=0}^{N-1} \widehat{\alpha}_{t-1}(j) a_{ji} b_i(\mathcal{O}_t)$

$\quad$ **end for**

$\quad c_t = 1 \Big/ \sum_{j=0}^{N-1} \widetilde{\alpha}_t(j)$

$\quad$ **for** $i = 0, 1, \ldots, N-1$ **do**

$\quad\quad \widehat{\alpha}_t(i) = c_t \widetilde{\alpha}_t(i)$

$\quad$ **end for**

end for

Note that (obviously):

$$\sum_{j=0}^{N-1} \widehat{\alpha}_{T-1}(j) = 1$$

Also, remembering that:

$$\widehat{\alpha}_t(i) = c_0 c_1 \cdots c_t \alpha_t(i)$$

We have:

$$\sum_{j=0}^{N-1} \widehat{\alpha}_{T-1}(j) = c_0 c_1 \cdots c_{T-1} \sum_{j=0}^{N-1} \alpha_{T-1}(j)$$

$$= c_0 c_1 \cdots c_{T-1} P(\mathcal{O} \,|\, \lambda)$$

$$\sum_{j=0}^{N-1} \widehat{\alpha}_{T-1}(j) = 1$$

$$\sum_{j=0}^{N-1} \widehat{\alpha}_{T-1}(j) = c_0 c_1 \cdots c_{T-1} \sum_{j=0}^{N-1} \alpha_{T-1}(j)$$

$$= c_0 c_1 \cdots c_{T-1} P(\mathcal{O} \mid \lambda)$$

Combining these results, we obtain:

$$P(\mathcal{O} \mid \lambda) = 1 \Big/ \prod_{j=0}^{T-1} c_j$$

It follows that we can compute the log of $P\left(\mathcal{O} \middle| \lambda\right)$ directly from the scaling factors c_t as:

Just "logging" the two sides of the equation

$$\log(P(\mathcal{O} \mid \lambda)) = -\sum_{j=0}^{T-1} \log c_j$$

$$\boxed{\sum_{j=0}^{N-1} \hat{\alpha}_{T-1}(j) = 1}$$

$$\boxed{\sum_{j=0}^{N-1} \hat{\alpha}_{T-1}(j)} = c_0 c_1 \cdots c_{T-1} \sum_{j=0}^{N-1} \alpha_{T-1}(j)$$

$$= \boxed{c_0 c_1 \cdots c_{T-1}} P(\mathcal{O} \mid \lambda)$$

Combining these results, we obtain:

$$P(\mathcal{O} \mid \lambda) = \boxed{1} \Big/ \boxed{\prod_{j=0}^{T-1} c_j}$$

It follows that we can compute the log of $P(\mathcal{O} \mid \lambda)$ directly from the scaling factors c_t as:

Just "logging" the two sides of the equation

$$\log(P(\mathcal{O} \mid \lambda)) = -\sum_{j=0}^{T-1} \log c_j$$

HMM Scaling

- Similarly, scale betas as $ct\beta t(i)$
- For re-estimation:

  Compute $\gamma t(i, j)$ and $\gamma t(i)$ using original formulas, but with scaled alphas, betas

- This gives us new values for $\lambda = (A,B,\pi)$

  "Easy exercise" to show re-estimate is exact when scaled alphas and betas used

- Also, $P(O|\lambda)$ cancels from formula

  Use $\log P(O|\lambda) = -\Sigma \log(c_j)$ to decide if iterate improves

All Together Now

Here we see the complete pseudo code for Solution 3

Given: $(O_0, O_1, \ldots, O_{T-1})$ and N and M

Initialize: $\lambda = (A, B, \pi)$

- A is NxN, B is NxM and π is 1xN
- $\pi_i \approx 1/N$, $a_{ij} \approx 1/N$, $b_j(k) \approx 1/M$, each matrix row stochastic, but not uniform

Initialize:

- maxIters = max number of re-estimation steps
- iters = 0
- oldLogProb = $-\infty$

Solution to problem 1
Solution to problem 2
Solution to problem 3

Forward Algorithm

- Forward algorithm

  With scaling

```
// compute α_0(i)
c_0 = 0
for i = 0 to N − 1
    α_0(i) = π(i) b_i(O_0)
    c_0 = c_0 + α_0(i)
next i

// scale the α_0(i)
c_0 = 1/c_0
for i = 0 to N − 1
    α_0(i) = c_0 α_0(i)
next i

// compute α_t(i)
for t = 1 to T − 1
    c_t = 0
    for i = 0 to N − 1
        α_t(i) = 0
        for j = 0 to N − 1
            α_t(i) = α_t(i) + α_{t−1}(j) a_{ji}
        next j
        α_t(i) = α_t(i) b_i(O_t)
        c_t = c_t + α_t(i)
    next i

    // scale α_t(i)
    c_t = 1/c_t
    for i = 0 to N − 1
        α_t(i) = c_t α_t(i)
    next i
next t
```

Solution to problem 1

Solution to problem 2

Solution to problem 3

Backward Algorithm

- Backward algorithm or "beta pass"

  With scaling

- Note: same scaling factor as alphas

```
// Let β_{T-1}(i) = 1, scaled by c_{T-1}
for i = 0 to N - 1
    β_{T-1}(i) = c_{T-1}
next i

// β-pass
for t = T - 2 to 0 by -1
    for i = 0 to N - 1
        β_t(i) = 0
        for j = 0 to N - 1
            β_t(i) = β_t(i) + a_{ij} b_j(O_{t+1}) β_{t+1}(j)
        next j
        // scale β_t(i) with same scale factor as α_t(i)
        β_t(i) = c_t β_t(i)
    next i
next t
```

Solution to problem 1
Solution to problem 2
Solution to problem 3

NOTE: Solution to problem 2 is NOT shown completely in this pseudo-code

Gammas

- Using scaled alphas and betas

```
for t = 0 to T − 2
    denom = 0
    for i = 0 to N − 1
        for j = 0 to N − 1
            denom = denom + α_t(i) a_ij b_j(O_{t+1}) β_{t+1}(j)
        next j
    next i
    for i = 0 to N − 1
        γ_t(i) = 0
        for j = 0 to N − 1
            γ_t(i, j) = (α_t(i) a_ij b_j(O_{t+1}) β_{t+1}(j))/denom
            γ_t(i) = γ_t(i) + γ_t(i, j)
        next j
    next i
next t

// Special case for γ_{T−1}(i)
denom = 0
for i = 0 to N − 1
    denom = denom + α_{T−1}(i)
next i
for i = 0 to N − 1
    γ_{T−1}(i) = α_{T−1}(i)/denom
next i
```

Solution to problem 1
Solution to problem 2
Solution to problem 3

Re-Estimation

- Again, using scaled gammas
- So formulas unchanged

```
// re-estimate π
for i = 0 to N − 1
    π_i = γ_0(i)
next i

// re-estimate A
for i = 0 to N − 1
    denom = 0
    for t = 0 to T − 2
        denom = denom + γ_t(i)
    next t
    for j = 0 to N − 1
        numer = 0
        for t = 0 to T − 2
            numer = numer + γ_t(i, j)
        next t
        a_ij = numer/denom
    next j
next i

// re-estimate B
for i = 0 to N − 1
    denom = 0
    for t = 0 to T − 1
        denom = denom + γ_t(i)
    next t
    for j = 0 to M − 1
        numer = 0
        for t = 0 to T − 1
            if(O_t == j) then
                numer = numer + γ_t(i)
            end if
        next t
        b_i(j) = numer/denom
    next j
next i
```

Solution to problem 1
Solution to problem 2
Solution to problem 3

Stopping Criteria

- Check that probability increases

  In practice, we want:

  logProb > oldLogProb + ε

- And don't exceed max iterations

```
// Compute log[P(O | λ)]

logProb = 0
for i = 0 to T − 1
    logProb = logProb + log(c_i)
next i
logProb = −logProb

// To iterate or not to iterate, that is the question...

iters = iters + 1
if (iters < maxIters and logProb > oldLogProb) then
    oldLogProb = logProb
    goto α-pass
else
    output λ = (π, A, B)
end if
```

Solution to problem 1
Solution to problem 2
Solution to problem 3

Recap solutions

- To problem 1:

$$\log\big(P(\mathcal{O}\,|\,\lambda)\big) = -\sum_{j=0}^{T-1} \log c_j$$

- To problem 2:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(\mathcal{O}\,|\,\lambda)} \qquad \widetilde{X}_t = \max_i \gamma_t(i)$$

- To problem 3:

  The full code seen before…

HMM Applications

- Applications ***not*** chosen because they are the most ingenious or clever, but because they:
  - Illustrate the basics
  - Show off the strengths of technique
  - Easy to understand and appreciate

- Academic publications usually favor novel, different, clever, (weird?), …

HMM for English Text Analysis

- Marvin the Martian arrives on Earth
  - Marvin sees written English text
  - Wants to learn something about it
  - Martians know HMMs, but not English

- So, strip out all non-letters, make all letters lower-case
  - 27 symbols (26 letters, word-space)
  - Train HMM on long sequence of “symbols”

Training

- For first training case, initialize
  - $N = 2$ and $M = 27$
  - Elements of A and π are all ~1/2
  - Elements of B are each ~1/27
- We'll use 50,000 symbols for training
- After 1st iteration: $\log P(O|\lambda) \approx -165097$
- After 100th iteration: $\log P(O|\lambda) \approx -137305$

The A and π Matrices

- Matrices A and π converge to:

$$\pi = \begin{bmatrix} 0.00000 & 1.00000 \end{bmatrix} \qquad A = \begin{bmatrix} 0.25596 & 0.74404 \\ 0.71571 & 0.28429 \end{bmatrix}$$

- What does this tell us?
  - Started in hidden state 1 (not state 0)
  - And we know transition probabilities between hidden states
- Nothing too interesting here
  - We don't (yet) know about hidden states

The B Matrix (Transpose)

- What does B matrix tell us?

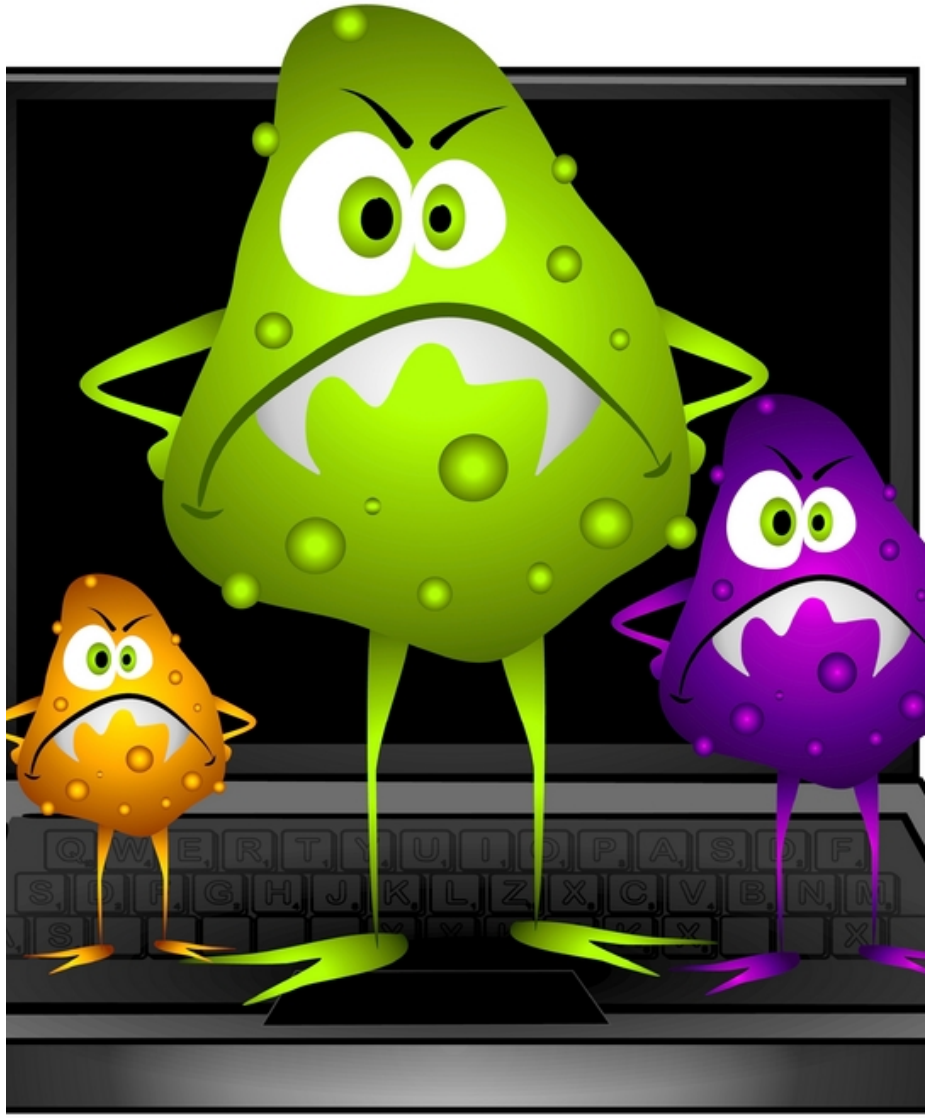
- This is **very interesting**!

  Why???

	Initial		Final	
a	0.03735	0.03909	0.13845	0.00075
b	0.03408	0.03537	0.00000	0.02311
c	0.03455	0.03537	0.00062	0.05614
d	0.03828	0.03909	0.00000	0.06937
e	0.03782	0.03583	0.21404	0.00000
f	0.03922	0.03630	0.00000	0.03559
g	0.03688	0.04048	0.00081	0.02724
h	0.03408	0.03537	0.00066	0.07278
i	0.03875	0.03816	0.12275	0.00000
j	0.04062	0.03909	0.00000	0.00365
k	0.03735	0.03490	0.00182	0.00703
l	0.03968	0.03723	0.00049	0.07231
m	0.03548	0.03537	0.00000	0.03889
n	0.03735	0.03909	0.00000	0.11461
o	0.04062	0.03397	0.13156	0.00000
p	0.03595	0.03397	0.00040	0.03674
q	0.03641	0.03816	0.00000	0.00153
r	0.03408	0.03676	0.00000	0.10225
s	0.04062	0.04048	0.00000	0.11042
t	0.03548	0.03443	0.01102	0.14392
u	0.03922	0.03537	0.04508	0.00000
v	0.04062	0.03955	0.00000	0.01621
w	0.03455	0.03816	0.00000	0.02303
x	0.03595	0.03723	0.00000	0.00447
y	0.03408	0.03769	0.00019	0.02587
z	0.03408	0.03955	0.00000	0.00110
space	0.03688	0.03397	0.33211	0.01298

Conclusion

- The B matrix tells Marvin about <u>consonants</u> and <u>vowels</u>
- He made no assumption *a priori*
- The training itself “learned” this info
- This is the essence of machine learning!
  - We don’t have to think too much
  - Machine does all the hard work
  - Machine “learns” (and so do we)



Detecting "Undetectable" Malware

- Malware is malicious software
  Designed to do bad things
- Most common form of detection is based on <u>signatures</u>
  What is a signature?
- Malware writers try to avoid detection
  - How to avoid signature detection?
  One option is to morph/modify code

Metamorphic Malware

- Malware that morphs with each new infection is called "metamorphic"

- How to write **metamorphic** malware?

  Standalone generator vs malware that "carries its own morphing engine"

- How to detect metamorphic malware?

  Also not easy, it's a research topic

  ➢Machine learning

Real-World Metamorphism

Examples of metamorphic malware

	Evol	Zmist	Zperm	Regswap	MetaPHOR
Instruction substitution	—	—	—	✓	—
Instruction permutation	✓	✓	—	—	✓
Garbage code	✓	✓	—	—	✓
Variable substitution	✓	✓	—	✓	✓
Alter control flow	—	✓	✓	—	✓

- Why “garbage code” (or “dead code”)?
- Why is substitution not used more?

Metamorphic Example

From malware known as <u>NGVCK</u>

3 morphed versions

- <u>All do the same thing</u>
- Each has different opcode sequence

What can you say about signature(s)?

Base Version	
	call delta
delta:	pop ebp
	sub ebp, offset delta
Morphed Version 1	
	call delta
delta:	sub dword ptr[esp], offset delta
	pop eax
	mov ebp, eax
Morphed Version 2	
	add ecx, 0031751B ; junk
	call delta
delta:	sub dword ptr[esp], offset delta
	sub ebx,00000909 ; junk
	mov edx, [esp]
	xchg ecx, eax ; junk
	add esp, 00000004
	and ecx, 00005E44 ; junk
	xchg edx, ebp

Experiments

- Seed the generator with NGVCK virus

- Generated 200 morphed copies
  1. Assemble each morphed *asm* into *exe*
  2. Verify that seed virus detected by AV…
  3. …and morphed copies ***not*** detected by AV

- Disassemble *exes*, extract opcodes
  1. Train HMMs, using 5-fold cross validation
  2. Score each model vs 40 benign samples

Scatterplot of Results

- Ideal separation!

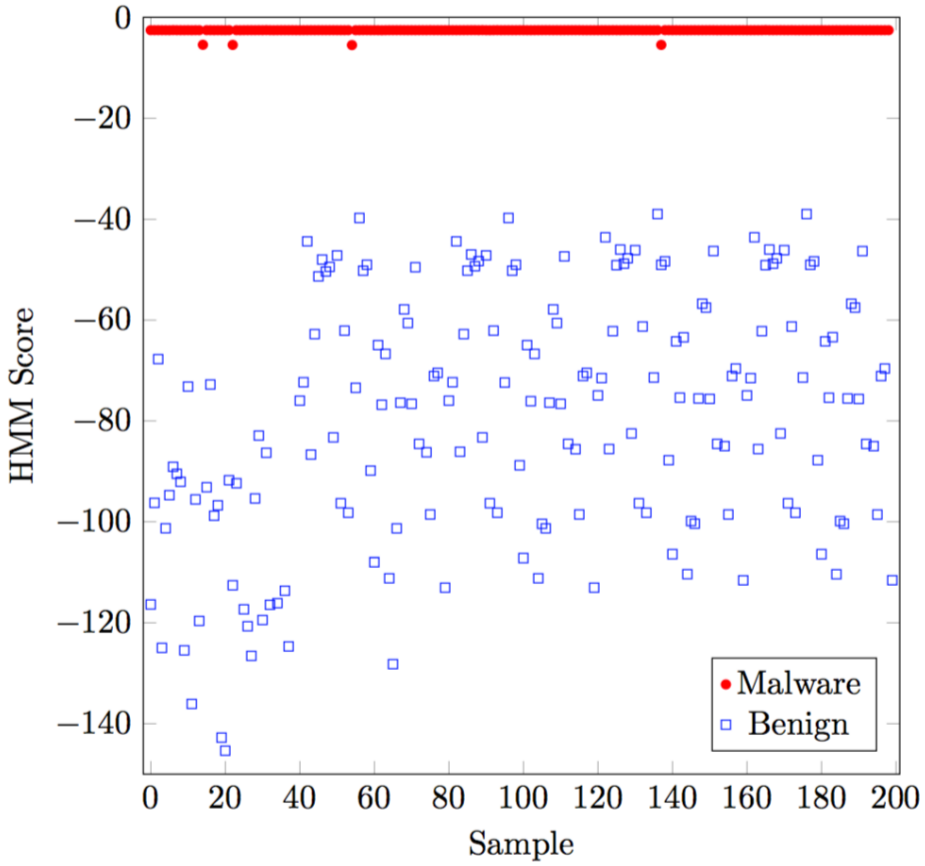
  Best-case scenario

- Using HMM…

  Easy to separate benign from virus

  Easy to detect!

- Undetectable using signatures…



What's Going On Here?

- Why can we detect this morphed malware using HMMs

  But not using signatures?

- Signatures are easily disrupted by simple transposition strategy

- HMM not affected by transposition

  ➢HMM "sees" differences between viruses and benign, in spite of this morphing

Statistical properties are not significantly affected by morphing

Conclusion

- Transposition is highly effective anti-signature strategy
- But ineffective for machine learning (or statistical-based) analysis
- Can virus writer defeat both signatures and machine learning?

  Yes, but something more is required.

  There is much more to say about this…