Lecture 5: Dynamic Programming

STAT 545: Intro. to Computational Statistics

Vinayak Rao

Purdue University

September 6, 2016

Preliminaries

HW1 is up on the course webpage
(due before class Tue, Sept 13)

Write up homework using `knitr` and R markdown

Submit HWs to course email (see lecture 1)

Dynamic Programming

Solve a complex problem by breaking it into simpler problems

Recursion without recalculation:

- Relate solution of a problem to solutions of simpler problems (recursion)
- Identify and solve initial (base) problems
- Reuse existing solutions to compute more complicated solutions (memoization)

Prob 1: Who is the tallest person in class?

Setup: We can only compare heights one pair at a time.

Naïve approach: build a binary relation matrix:

$$\begin{array}{c} \\ a \\ b \\ \\ N \end{array} \begin{array}{c} \begin{array}{cccc} a & b & \ldots & N \end{array} \\ \begin{bmatrix} - & 1 & \ldots & 1 \\ 0 & - & \ldots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & - \end{bmatrix} \end{array}$$

$O(N^2)$ comparisons, but lots of redundancy.

Can we do better?

Prob 2: Order-dependent sums

Pick a set of unique integers. E.g. $\{1, 3, 4\}$.
Find the number of ways to write N as sums of these.

E.g. for $N = 5$, the answer is 6:

$$\begin{aligned} 5 &= 1+1+1+1+1 \\ &= 1+1+3 \\ &= 1+3+1 \\ &= 3+1+1 \\ &= 1+4 \\ &= 4+1 \end{aligned}$$

`http://web.stanford.edu/class/cs97si/04-dynamic-programming.pdf`

Order-dependent sums (contd.)

How do we solve for $N = 1000$?

Let D_N be the solution (e.g. $D_5 = 6$).

Define a recursion. Observe that

- any sum ends with a $1, 3$ or 4.
- if the last term is i, the remaining sum to $N - i$.
- $D_N = D_{N-1} + D_{N-3} + D_{N-4}$

Also, $D_0 = D_1 = D_2 = 1, D_3 = 2$

```
ordered_sum <- function(N) {
  D <- rep(1,N); D[c(3,4)] <- c(2, 3)
  for(i in 5:N) {
    D[i] <- D[i-1] + D[i-3] + D[i-4] }
  return(D[N]) }
```

The knapsack problem

Given:

- a bag with (integer) capacity W lbs
- n types of objects, with integer weights $(w_1, \ldots, w_n)$ lbs and positive value $(v_1, \ldots, v_n)$
- Unlimited objects of each type

Goal: Fill bag to maximize value $\mathbb{V}(W)$

What is $\mathbb{V}(0)$?

How about $\mathbb{V}(1)$ and $\mathbb{V}(2)$?

Can we express $\mathbb{V}(i)$ in terms of $\mathbb{V}(j), j < i$?

$\mathbb{V}(i) = \max_{j:w_j \leq i} \mathbb{V}(i - w_j) + v_j$

A 2-DIMENSIONAL DYNAMIC PROGRAM

Want to align nucleotides in two DNA sequences

Similarity can suggest functionality of a newly sequenced gene

Russell Doolittle and colleagues found similarities between cancer-causing gene and normal growth factor (PDGF) gene

Simple sources of misalignment:

Substitution:	A-A-C-T-G-G-A
	A-A-C-T-C-G-A
Insertion:	A-A-C-G-G-A
	A-A-C-*-G-A
Deletion:	A-A-C-T-*-G-A
	A-A-C-T-C-G-A

Sequence alignment

Given two sequences:

```
A-A-C-T-A-T-G-G-C-C-A
A-C-A-C-T-A-T-G-G-C-T
```

What is the best alignment?

Sequence alignment

Given two sequences:

```
A-A-C-T-A-T-G-G-C-C-A
A-C-A-C-T-A-T-G-G-C-T
```

Define a distance between two sequences:

- Each substitution has a cost C_S
- Each insertion/deletion has a cost C_G (gap penalty)
- In practice, these can depend on the nucleotides

```
A-A-*-C-T-A-T-G–G-C-C-A
A-C-A-C-T-A-T-G-G-*-C-T
```

This alignment has cost $2C_S + 2C_G$.

Dynamic programming recursion

Consider aligning to two strings S_1 and S_2 of length i and j:

$S_1 =$ …-G-C-C-A and $S_2 =$ …-G-G-C-T

Three possibilities:

- The last two characters are matched:

…-A …-T	$C_M(i,j) =$	$Cost(i-1,j-1)+$ Cost of matching elements $S_1(i)$ and $S_2(j)$.

- A gap in the first string:

…-* …-T	$C_I(i,j) =$	$Cost(i,j-1)+$ Cost of inserting gap after $S_2(j)$.

- A gap in the second string:

…-A …-*	$C_D(i,j) =$	$Cost(i-1,j)+$ Cost of inserting gap after $S_1(i)$.

The actual (best) cost:

$Cost(i,j) = \min\left(C_M(i,j), C_I(i,j), C_D(i,j)\right)$

Demo

[http : //baba.sourceforge.net]

Backtracking

Forward recursion only returns cost of the best alignment.
What is this alignment?

Compute via a backward trace

Recall: $Cost(i,j) = \min\left(C_M(i,j), C_I(i,j), C_D(i,j)\right)$

- If $Cost(i,j) = C_M(i,j)$ then add $S_1(i)$ and $S_2(j)$ to the heads of strings 1 and 2 respectively, and decrement i and j.

- If $Cost(i,j) = C_I(i,j)$ then add $S_1(i)$ to the head of strings 1, and decrement i.

- If $Cost(i,j) = C_D(i,j)$ then add $S_2(j)$ to the head of strings 2, and decrement j.

Demo (contd)

[http : //baba.sourceforge.net]

Overall algorithm: Needleman-Wunsch algorithm.

Cost:

- Forward pass: $O(NM)$ time (computations)
  $O(NM)$ space (memory)
- Backward pass: $O(N + M)$ time

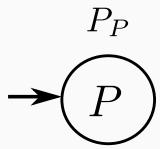
Additive loss functions and log-probability

We formulated the cost of any alignment as a sum of penalties for (mis)matches, insertions and deletions.

Can also formulate it in terms of the transition and emission probabilities of a hidden Markov model (HMM).

A 3-state Markov chain: insert (I), delete (D), and pair (P).

Additive loss functions and log-probability



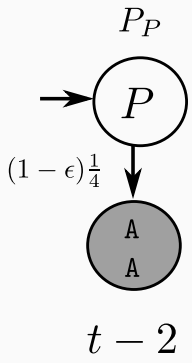
$t-2$

If $C(x)$ is the cost of config x, then the probability $P(x)$ is:

$$P(x) \propto \exp(-C(x))$$

Minimizing cost $\equiv$ Maximizing prob. (Maximum *a posteriori*)

Additive loss functions and log-probability

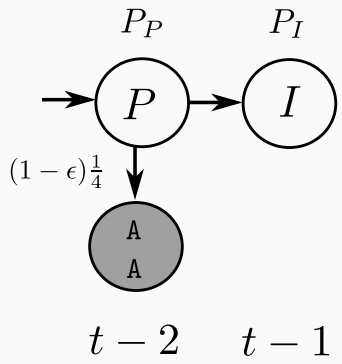


If $C(x)$ is the cost of config x, then the probability $P(x)$ is:

$$P(x) \propto \exp(-C(x))$$

Minimizing cost $\equiv$ Maximizing prob. (Maximum *a posteriori*)

Additive loss functions and log-probability

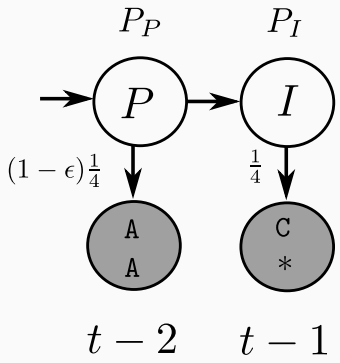


If $C(x)$ is the cost of config x, then the probability $P(x)$ is:

$$P(x) \propto \exp(-C(x))$$

Minimizing cost $\equiv$ Maximizing prob. (Maximum *a posteriori*)

Additive loss functions and log-probability

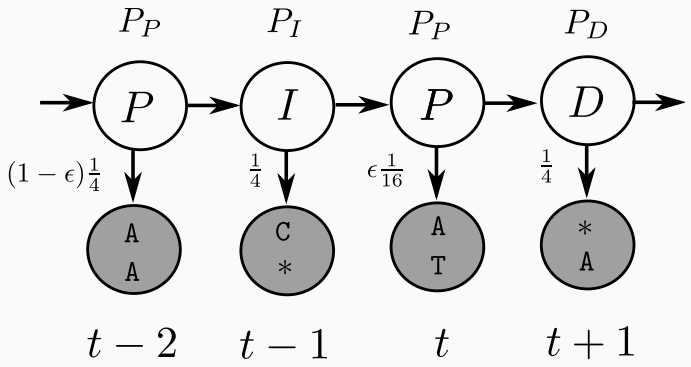


If $C(x)$ is the cost of config x, then the probability $P(x)$ is:

$$P(x) \propto \exp(-C(x))$$

Minimizing cost $\equiv$ Maximizing prob. (Maximum *a posteriori*)

Additive loss functions and log-probability

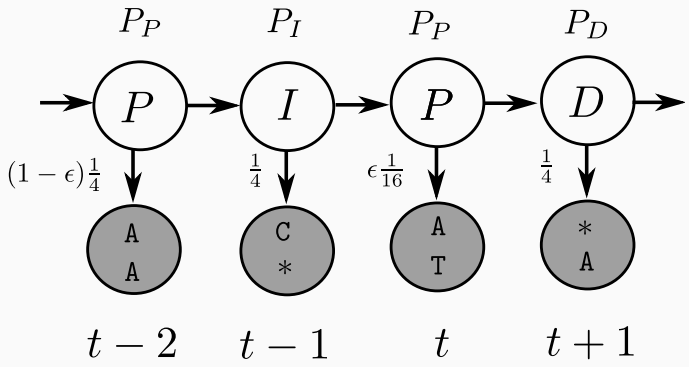


If $C(x)$ is the cost of config x, then the probability $P(x)$ is:

$$P(x) \propto \exp(-C(x))$$

Minimizing cost $\equiv$ Maximizing prob. (Maximum *a posteriori*)

Additive loss functions and log-probability



Typical HMM has a simpler observation process

- Seq. alignment discards $*$'s and emission times

The transition probability of typical HMM is more complicated

Conclusion

We looked at dynamic programming to solve complicated looking problems by recursively solving simpler subproblems.

Next class we'll focus on a special problem, viz. Kalman filtering.