25 Lecture 25, Apr 22

Announcements

- Course project due Wed, 4/29 @ 11:00AM.

Last Time

- Path algorithm.
- ALM (augmented Lagrangian method) or method of multipliers.

Today

- ADMM (alternating direction method of multipliers). A generic method for solving many regularization problems.
- Dynamic programming.
- HW7 solution sketch in Julia. `http://hua-zhou.github.io/teaching/st790-2015spr/hw07sol.html`

ADMM

☞ A definite resource for learning ADMM is (Boyd et al., 2011)
`http://stanford.edu/~boyd/admm.html`

- **A**lternating **d**irection **m**ethod of **m**ultipliers (ADMM).
  - Consider optimization problem

$$\begin{aligned} \text{minimize} \quad & f(\boldsymbol{x}) + g(\boldsymbol{y}) \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{y} = \boldsymbol{c}. \end{aligned}$$

  - The augmented Lagrangian

$$\mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + g(\boldsymbol{y}) + \langle \boldsymbol{\lambda}, \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{y} - \boldsymbol{c} \rangle + \frac{\rho}{2} \|\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{y} - \boldsymbol{c}\|_2^2.$$

  - *Idea*: perform block descent on $\boldsymbol{x}$ and $\boldsymbol{y}$ and then update multiplier vector $\boldsymbol{\lambda}$

$$\begin{aligned} \boldsymbol{x}^{(t+1)} & \leftarrow \min_{\boldsymbol{x}} f(\boldsymbol{x}) + \langle \boldsymbol{\lambda}, \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{y}^{(t)} - \boldsymbol{c} \rangle + \frac{\rho}{2} \|\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{y}^{(t)} - \boldsymbol{c}\|_2^2 \\ \boldsymbol{y}^{(t+1)} & \leftarrow \min_{\boldsymbol{y}} g(\boldsymbol{y}) + \langle \boldsymbol{\lambda}, \boldsymbol{A}\boldsymbol{x}^{(t+1)} + \boldsymbol{B}\boldsymbol{y} - \boldsymbol{c} \rangle + \frac{\rho}{2} \|\boldsymbol{A}\boldsymbol{x}^{(t+1)} + \boldsymbol{B}\boldsymbol{y} - \boldsymbol{c}\|_2^2 \\ \boldsymbol{\lambda}^{(t+1)} & \leftarrow \boldsymbol{\lambda}^{(t)} + \rho(\boldsymbol{A}\boldsymbol{x}^{(t+1)} + \boldsymbol{B}\boldsymbol{y}^{(t+1)} - \boldsymbol{c}) \end{aligned}$$

☞ If we minimize $\boldsymbol{x}$ and $\boldsymbol{y}$ jointly, then it is same as ALM. We gain splitting by blockwise updates.

- ADMM converges under mild conditions: f, g convex, closed, and proper, $\mathcal{L}_0$ has a saddle point.

- Example: *Generalized lasso* problem minimizes

$$\frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \mu\|\boldsymbol{D}\boldsymbol{\beta}\|_1.$$

  - Special case $\boldsymbol{D} = \boldsymbol{I}_p$ corresponds to *lasso*. Special case

$$\boldsymbol{D} = \begin{pmatrix} 1 & -1 & & & \\ & & \cdots & & \\ & & & 1 & -1 \end{pmatrix}$$

  corresponds to *fused lasso*. Numerous applications.

  - Define $\boldsymbol{\gamma} = \boldsymbol{D}\boldsymbol{\beta}$. Then we solve

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \mu\|\boldsymbol{\gamma}\|_1 \\ \text{sujbect to} & \boldsymbol{D}\boldsymbol{\beta} = \boldsymbol{\gamma}. \end{array}$$

  - Augmented Lagrangian is

$$\mathcal{L}_\rho(\boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\lambda}) = \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \mu\|\boldsymbol{\gamma}\|_1 + \boldsymbol{\lambda}^{\mathsf{T}}(\boldsymbol{D}\boldsymbol{\beta} - \boldsymbol{\gamma}) + \frac{\rho}{2}\|\boldsymbol{D}\boldsymbol{\beta} - \boldsymbol{\gamma}\|_2^2.$$

  - ADMM algorithm:

$$\begin{aligned} \boldsymbol{\beta}^{(t+1)} &\leftarrow \min_{\boldsymbol{\beta}} \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \boldsymbol{\lambda}^{(t)T}(\boldsymbol{D}\boldsymbol{\beta} - \boldsymbol{\gamma}^{(t)}) + \frac{\rho}{2}\|\boldsymbol{D}\boldsymbol{\beta} - \boldsymbol{\gamma}^{(t)}\|_2^2 \\ \boldsymbol{\gamma}^{(t+1)} &\leftarrow \min_{\boldsymbol{\gamma}} \mu\|\boldsymbol{\gamma}\|_1 + \boldsymbol{\lambda}^{\mathsf{T}}(\boldsymbol{D}\boldsymbol{\beta}^{(t+1)} - \boldsymbol{\gamma}) + \frac{\rho}{2}\|\boldsymbol{D}\boldsymbol{\beta}^{(t+1)} - \boldsymbol{\gamma}\|_2^2 \\ \boldsymbol{\lambda}^{(t+1)} &\leftarrow \boldsymbol{\lambda}^{(t)} + \rho(\boldsymbol{D}\boldsymbol{\beta}^{(t+1)} - \boldsymbol{\gamma}^{(t+1)}) \end{aligned}$$

  ☞ Update $\boldsymbol{\beta}$ is a smooth quadratic problem. Note the Hessian keeps constant between iterations, therefore its inverse (or decomposition) can be calculated just once, cached in memory, and re-used in each iteration.

  ☞ Update $\boldsymbol{\gamma}$ is a separated lasso problem (elementwise soft-thresholding).

- Remarks on ADMM:

  - Related algorithms

    * *split Bregman iteration* (Goldstein and Osher, 2009)
    * Dykstra (1983)'s alternating projection algorithm
    * ...

  Proximal point algorithm applied to the dual.

  - Numerous applications in statistics and machine learning: lasso, generalized lasso, graphical lasso, (overlapping) group lasso, ...
  - Embraces distributed computing for big data (Boyd et al., 2011).

- Distributed computing with ADMM. Consider, for example, solving lasso with a huge training data set $(\boldsymbol{X}, \boldsymbol{y})$, which is stored on B machines. Denote the distributed data sets by $(\boldsymbol{X}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{X}_B, \boldsymbol{y}_B)$. Then the lasso criterion is

$$\frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \mu\|\boldsymbol{\beta}\|_1 = \frac{1}{2}\sum_{b=1}^{B} \|\boldsymbol{y}_b - \boldsymbol{X}_b\boldsymbol{\beta}\|_2^2 + \mu\|\boldsymbol{\beta}\|_1.$$

  The ADMM form is

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\sum_{b=1}^{B} \|\boldsymbol{y}_b - \boldsymbol{X}_b\boldsymbol{\beta}_b\|_2^2 + \mu\|\boldsymbol{\beta}\|_1 \\ \text{subject to} \quad & \boldsymbol{\beta}_b = \boldsymbol{\beta}, \quad b = 1, \ldots, B. \end{aligned}$$

  Here $\boldsymbol{\beta}_b$ are local variables and $\boldsymbol{\beta}$ is the global (or consensus) variable. The augmented Lagrangian function is

$$\mathcal{L}_\rho(\boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\lambda}) = \frac{1}{2}\sum_{b=1}^{B} \|\boldsymbol{y}_b - \boldsymbol{X}_b\boldsymbol{\beta}_b\|_2^2 + \mu\|\boldsymbol{\beta}\|_1 + \sum_{b=1}^{B} \boldsymbol{\lambda}_b^{\mathsf{T}}(\boldsymbol{\beta}_b - \boldsymbol{\beta}) + \frac{\rho}{2}\sum_{b=1}^{B} \|\boldsymbol{\beta}_b - \boldsymbol{\beta}\|_2^2.$$

  The ADMM algorithm runs as

  - Update local variables $\boldsymbol{\beta}_b$

$$\boldsymbol{\beta}_b^{(t+1)} \quad \leftarrow \quad \min \frac{1}{2}\|\boldsymbol{y}_b - \boldsymbol{X}_b\boldsymbol{\beta}_b\|_2^2 + \boldsymbol{\lambda}_b^{\mathsf{T}}(\boldsymbol{\beta}_b - \boldsymbol{\beta}^{(t)}) + \frac{\rho}{2}\|\boldsymbol{\beta}_b - \boldsymbol{\beta}^{(t)}\|_2^2, \quad b = 1, \ldots, B,$$

    in parallel on B machines.

  - Collect local variables $\boldsymbol{\beta}_b^{(t)}$, $b = 1, \ldots, B$, and update consensus variable $\boldsymbol{\beta}$

$$\boldsymbol{\beta}^{(t+1)} \quad \leftarrow \quad \min \mu\|\boldsymbol{\beta}\|_1 + \sum_{b=1}^{B} \boldsymbol{\lambda}_b^{\mathsf{T}}(\boldsymbol{\beta}_b^{(t+1)} - \boldsymbol{\beta}) + \frac{\rho}{2}\sum_{b=1}^{B} \|\boldsymbol{\beta}_b^{(t+1)} - \boldsymbol{\beta}\|_2^2$$

    by elementwise soft-thresholding.

  - Update multipliers

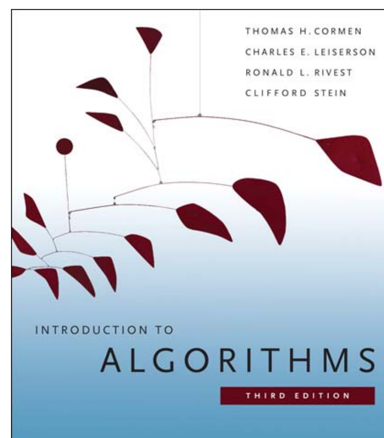
$$\boldsymbol{\lambda}_b^{(t+1)} \leftarrow \boldsymbol{\lambda}_b^{(t)} + \rho(\boldsymbol{\beta}_b^{(t+1)} - \boldsymbol{\beta}^{(t+1)}), \quad b = 1, \ldots, B.$$

  ☞ The whole procedure is carried out without ever transferring distributed data sets $(\boldsymbol{y}_b, \boldsymbol{X}_b)$ to a central location!

Dynamic programming: introduction

- *Divide-and-conquer*: break the problem into smaller *independent* subproblems
  - fast sorting,
  - FFT,
  - ...
- *Dynamic programming* (DP): subproblems are not independent, that is, subproblems share common subproblems.
- DP solves these subproblems once and store them in a table.
- Use these optimal solutions to construct an optimal solution for the original problem.
- Richard Bellman began the systematic study of DP in 50s.
- Some classical (non-statistical) DP problems:
  - Matrix-chain multiplication,
  - Longest common subsequence,
  - Optimal binary search trees,
  - ...

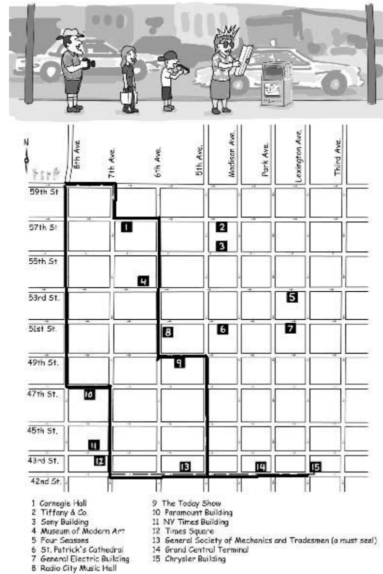
  See (Cormen et al., 2009) for a general introduction



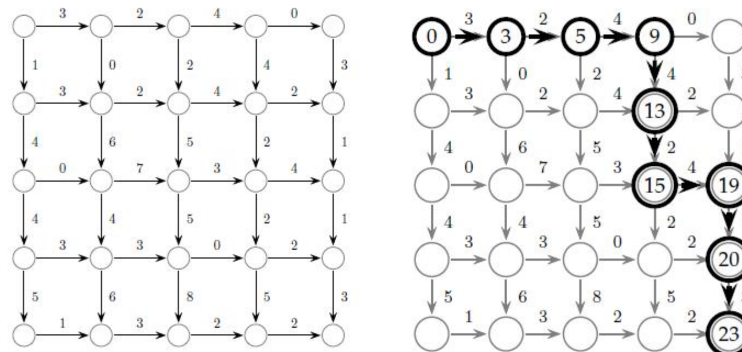
- Some classical DP problems in statistics
  - Hidden Markov model (HMM),
  - Some fused-lasso problems,

- Graphical models (Wainwright and Jordan, 2008),
- Sequence alignment, e.g., discovery of the cystic fibrosis gene in 1989,
- ...

- Let's work on the a DP algorithm for the Manhattan tourist problem (MTP), taken from Jones and Pevzner (2004, Section 6.3).



- MTP: weighted graph

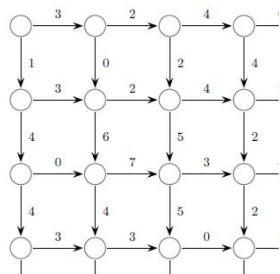


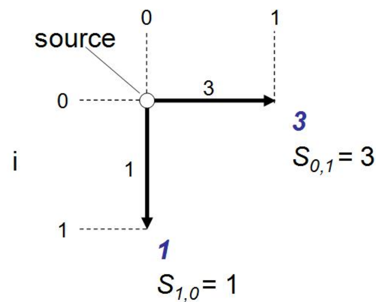
Find a longest path in a weighted grid (only eastward and southward)

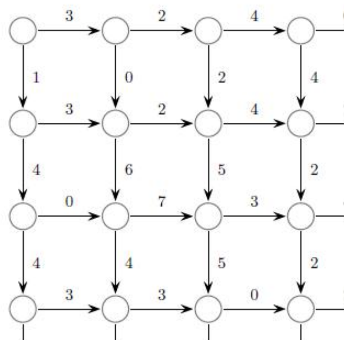
- *Input*: a weighted grid G with two distinguished vertices: a source $(0, 0)$ and a sink (n, m).
- *Output*: a longest path $MT(n, m)$ in G from source to sink.

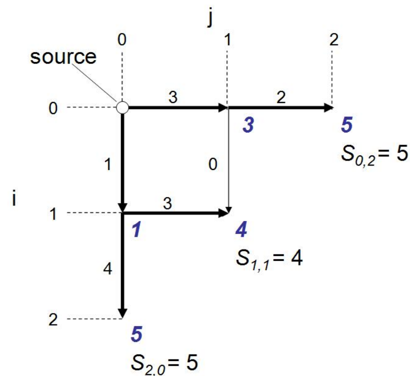
Brute force enumeration is out of the question even for a moderate sized graph.

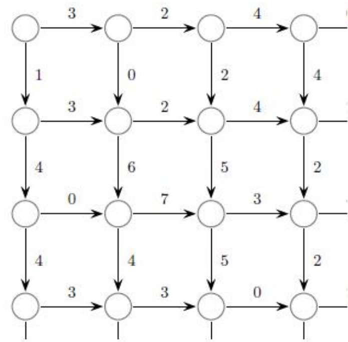
- Simple recursive program.
  $MT(n, m)$:
  - If $n = 0$ or $m = 0$, return $MT(0, 0)$
  - $x \leftarrow MT(n-1, m)+$ weight of the edge from $(n-1, m)$ to (n, m)
    $y \leftarrow MT(n, m-1)+$ weight of the edge from $(n, m-1)$ to (n, m)
  - Return $\max\{x, y\}$
- Something wrong
  - $MT(n, m-1)$ needs $MT(n-1, m-1)$, so as $MT(n-1, m)$.
  - So $MT(n-1, m-1)$ will be computed at least twice.
  - Dynamic programming: the same idea as this recursive algorithm, but keep all intermediate results in a *table* and reuse.
- MTP: dynamic programming
  - Calculate optimal path score for each vertex in the graph
  - Each vertex's score is the maximum of the previous vertices score plus the weight of the respective edge in between

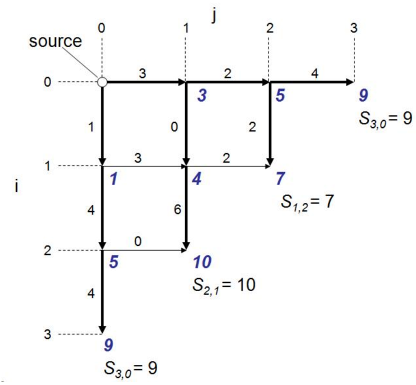


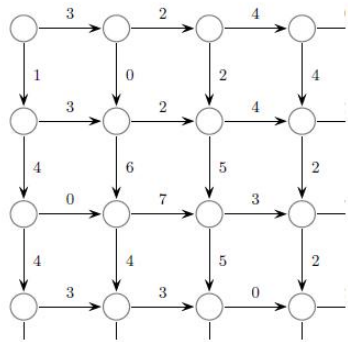


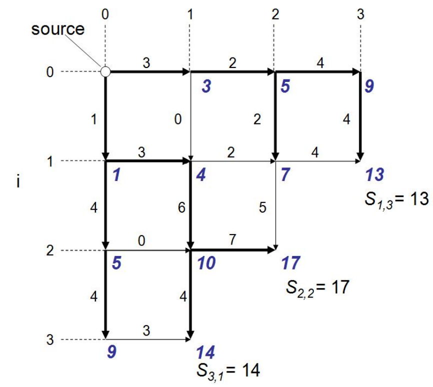


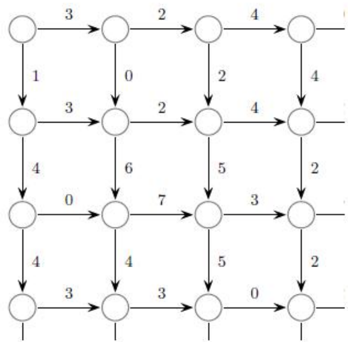


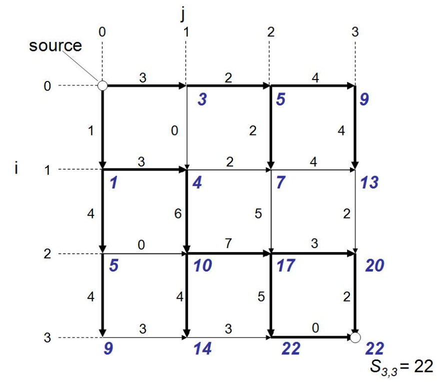




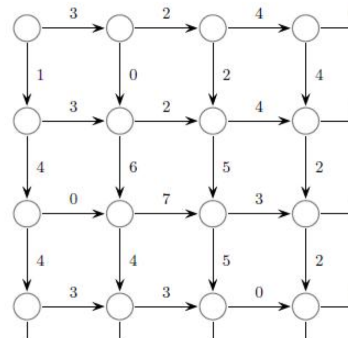


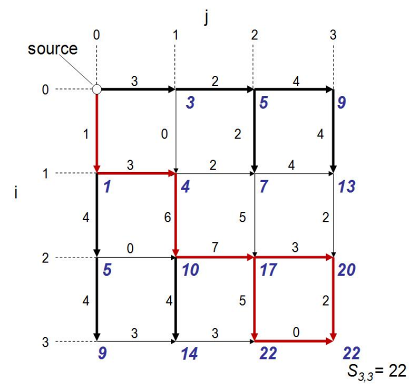






- MTP dynamic programming: *path!*





Showing all back-traces!

- MTP: recurrence
  - Computing the score for a point (i, j) by the recurrence relation:

$$s(i,j) = \max \begin{cases} s(i-1,j) + \text{weight between } (i-1,j) \text{ and } (i,j) \\ s(i,j-1) + \text{weight between } (i,j-1) \text{ and } (i,j) \end{cases}$$

  - The *run time* is mn for a n by m grid.
    (n = number of rows, m = number of columns)

- Remarks on DP:
  - Steps for developing a DP algorithm
    1. Characterize the structure of an optimal solution
    2. Recursively define the value of an optimal solution
    3. Computer the value of an optimal solution in a bottom-up fashion
    4. Construct an optimal solution from computed information
  - "*Programming*" both here and in linear programming refers to the use of a *tabular* solution method.
  - Many problems involve large tables and entries along certain directions may be filled out in parallel – fine scale *parallel computing*.

Application of dynamic programming: HMM

- Hidden Markov model (HMM) (Baum et al., 1970).
  - HMM is a Markov chain that emits symbols:
    Markov chain $(\mu, A = \{a_{kl}\})$ + emission probabilities $e_k(b)$
  - The *state sequence* $\pi = \pi_1 \cdots \pi_L$ is governed by the Markov chain

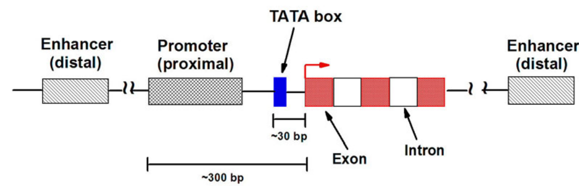
$$\mathbf{P}(\pi_1 = k) = \mu(k), \quad \mathbf{P}(\pi_i = l | \pi_{i-1} = k) = a_{kl}.$$

  - The *symbol sequence* $\mathbf{x} = x_1 \cdots x_L$ is determined by the underlying state sequence π

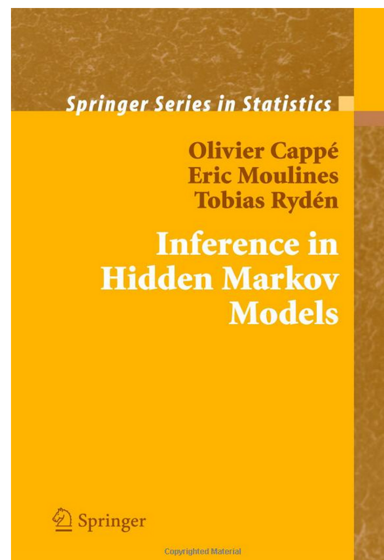
$$\mathbf{P}(\mathbf{x}, \pi) = \prod_{i=1}^{L} e_{\pi_i}(x_i) a_{\pi_{i-1}\pi_i}.$$

  - It is called *hidden* because in applications the state sequence is unobserved.

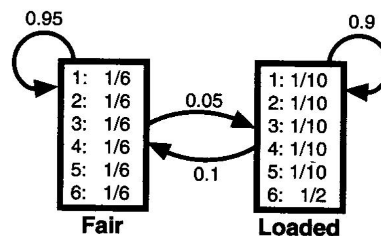
- Wide applications of HMM.
  - Wireless communication: IEEE 802.11 WLAN.
  - Mobile communication: CDMA and GSM.
  - Speech recognition (Rabiner, 1989)
    Hidden states: text, symbols: acoustic signals.
  - Haplotyping and genotype imputation
    Hidden states: haplotypes, symbols: genotypes.
  - Gene prediction (Burge, 1997)



- General reference book on HMM:



- Let's work on a simple HMM example. The Occasionally Dishonest Casino (Durbin et al., 2006)



- Fundamental questions of HMM:

```
Rolls (Observed data)  3154235314254132514636126626164...
Die (Hidden states)    FFFFFFFFFFFFFFFFFFLLLLLLLLLLLL...
```

  - How to compute the probability of the observed sequence of symbols given known parameters a_{kl} and $e_k(b)$?
    Answer: *Forward algorithm.*
  - How to compute the posterior probability of the state at a given position (posterior decoding) given a_{kl} and $e_k(b)$?
    Answer: *Backward algorithm.*
  - How to estimate the parameters a_{kl} and $e_k(b)$?
    Answer: *Baum-Welch algorithm.*
  - How to find the most likely sequence of hidden states?
    Answer: *Viterbi algorithm* (Viterbi, 1967).

- Forward algorithm:
  - Calculate the *probability of an observed sequence*

$$\mathbf{P}(\mathbf{x}) = \sum_{\pi} \mathbf{P}(\mathbf{x}, \pi).$$

  - Brute force evaluation by enumerating is *impractical*
  - Define the *forward variable*

$$f_k(i) = \mathbf{P}(x_1 \ldots x_i, \pi_i = k).$$

  - Recursion formula for forward variables

$$f_l(i+1) = \mathbf{P}(x_1 \ldots x_i x_{i+1}, \pi_{i+1} = l) = e_l(x_{i+1}) \sum_k f_k(i) a_{kl}.$$

  - Algorithm:
    * Initialization ($i = 1$): $f_k(1) = a_{0k} e_k(x_1)$.
    * Recursion ($i = 2, \ldots, L$): $f_l(i) = e_l(x_i) \sum_k f_k(i-1) a_{kl}$.
    * Termination: $\mathbf{P}(\mathbf{x}) = \sum_k f_k(L)$.

    Time complexity = (# states)^2 $\times$ length of sequence.

- Backward algorithm.

- Calculate the *posterior state probabilities at each position*

$$\mathbf{P}(\pi_i = k|\mathbf{x}) = \frac{\mathbf{P}(\mathbf{x}, \pi_i = k)}{\mathbf{P}(\mathbf{x})}.$$

- Enough to calculate the numerator

$$\begin{aligned}
\mathbf{P}(\mathbf{x}, \pi_i = k) &= \mathbf{P}(x_1 \dots x_i, \pi_i = k)\mathbf{P}(x_{i+1} \dots x_L | x_1 \dots x_i, \pi_i = k) \\
&= \mathbf{P}(x_1 \dots x_i, \pi_i = k)\mathbf{P}(x_{i+1} \dots x_L | \pi_i = k) \\
&= f_k(i) b_k(i).
\end{aligned}$$

- Recursion formula for the *backward variables*

$$b_k(i) = \mathbf{P}(x_{i+1} \dots x_L | \pi_i = k) = \sum_l a_{kl} e_l(x_{i+1}) b_l(i+1).$$

- Algorithm:
  * Initialization ($i = L$): $b_k(L) = 1$ for all k
  * Recursion ($i = L-1, \dots, 1$): $b_k(i) = \sum_l a_{kl} e_l(x_{i+1}) b_l(i+1)$
  * Termination: $\mathbf{P}(\mathbf{x}) = \sum_l a_{0l} e_l(x_1) b_l(1)$

  Time complexity = (# states)^2 $\times$ length of sequence
- The Occasionally Dishonest Casino.

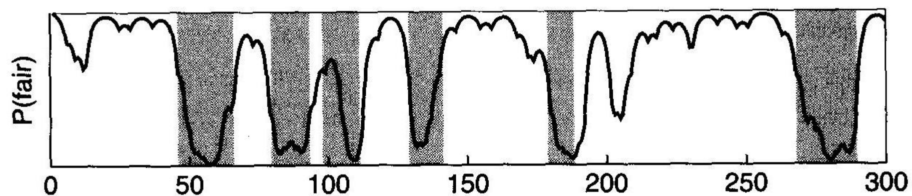


Figure 3.6 ***The posterior probability of being in the state corresponding to the fair die in the casino example. The x axis shows the number of the roll. The shaded areas show when the roll was generated by the loaded die.***

- Parameter estimation for HMM – Baum-Welch algorithm.
- Question: Given n independent training symbol sequences $\mathbf{x}^1, \dots, \mathbf{x}^n$, how to find the parameter value that maximizes the log-likelihood $\log \mathbf{P}(\mathbf{x}^1, \dots, \mathbf{x}^n|\theta) = \sum_{j=1}^n \log \mathbf{P}(\mathbf{x}^j|\theta)$?
  - When the underlying state sequences are *known*: Simple.
  - When the underlying state sequences are *unknown*: Baum-Welch algorithm.

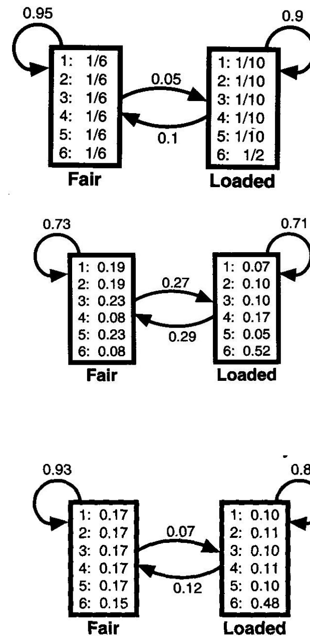
- MLE when state sequences are known.
  - Let A_{kl} = # transitions from state k to l
    $E_k(b)$ = # state k emitting symbol b
    The MLEs are
$$a_{kl} = \frac{A_{kl}}{\sum_{l'} A_{kl'}} \text{ and } e_k(b) = \frac{E_k(b)}{\sum_{b'} E_k(b')}. \tag{1}$$
  - To avoid overfitting with insufficient data, add pseudocounts
$$\begin{aligned} A_{kl} &= \# \text{ transitions } k \text{ to } l \text{ in training data } + r_{kl}; \\ E_k(b) &= \# \text{ emissions of } b \text{ from } k \text{ in training data } + r_k(b) \end{aligned}$$

- MLE when state sequences are unknown: Baum-Welch algorithm.
  - *Idea*: Replace the counts A_{kl} and $E_k(b)$ by their expectations conditional on current parameter iterate (*EM algorithm*!)
  - The probability that a_{kl} is used at position i of sequence $\mathbf{x}$:
$$\begin{aligned} & \mathbf{P}(\pi_i = k, \pi_{i+1} = l | \mathbf{x}, \theta) \\ = \; & \mathbf{P}(\mathbf{x}, \pi_i = k, \pi_{i+1} = l)/\mathbf{P}(\mathbf{x}) \\ = \; & \mathbf{P}(x_1 \ldots x_i, \pi_i = k) a_{kl} e_l(x_{i+1}) \mathbf{P}(x_{i+2} \ldots x_L | \pi_{i+1} = l)/\mathbf{P}(\mathbf{x}) \\ = \; & f_k(i) a_{kl} e_l(x_{i+1}) b_l(i+1)/\mathbf{P}(\mathbf{x}). \end{aligned}$$
  - So the expected number of times that a_{kl} is used in all training sequences is
$$A_{kl} = \sum_{j=1}^{n} \frac{1}{\mathbf{P}(\mathbf{x}^j)} \sum_i f_k^j(i) a_{kl} e_l(x_{i+1}^j) b_l^j(i+1). \tag{2}$$

- Baum-Welch Algorithm.
  - Initialization: Pick arbitrary model parameters
  - Recursion
    * Set all the A and E variables to pseudocounts rs (or to zero)
    * For each sequence $j = 1, \ldots, n$
      · calculate $f_k(i)$ for sequence j using the forward algorithm
      · calculate $b_k(i)$ for sequence j using the backward algorithm
      · add contribution of sequence j to A (2) and E (**??**)
    * Calculate the new model parameters using (1)
    * Calculate the new log-likelihood of the model

– Termination: Stop if change in log-likelihood is less than a predefined threshold or the maximum number of iteration is exceeded

- Baum-Welch – The Occasionally Dishonest Casino.



- Viterbi Algorithm:

  – Calculate *the most probable state path*

$$\boldsymbol{\pi}^* = \text{argmax}_{\boldsymbol{\pi}} P(\boldsymbol{x}, \boldsymbol{\pi}).$$

  – Define the *Viterbi variable*

$$v_l(i) = P(\text{the most probable path ending in state } k \text{ with observation } x_i).$$

  – Recursion for the Viterbi variables

$$v_l(i+1) = e_l(x_{i+1}) \max_k (v_k(i) a_{kl})$$

  – Algorithm:

    * Initialization ($i = 0$): $v_0(0) = 1$, $v_k(0) = 0$ for all $k > 0$
    * Recursion ($i = 1, \ldots, L$):

$$\begin{aligned} v_l(i) &= e_l(x_i) \max_k (v_k(i-1) a_{kl}) \\ \text{ptr}_i(l) &= \text{argmax}_k (v_k(i-1) a_{kl}) \end{aligned}$$

* Termination:

$$
\begin{aligned}
\mathbf{P}(\mathbf{x}, \boldsymbol{\pi}^*) &= \max_k (v_k(L) a_{k0} \\
\pi_L^* &= \text{argmax}_k (v_k(L) a_{k0})
\end{aligned}
$$

* Traceback ($i = L, \ldots, 1$): $\pi_{i=1}^* = \text{ptr}_i(\pi_i^*)$

Time complexity = (# states)^2 × length of sequence

– Viterbi decoding - The Occasionally Dishonest Casino.

```
Rolls   315116246446644245311321631164152133625144543631656626566666
Die     FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLLLLL
Viterbi FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLL

Rolls   651166453132651245636664631636663162326455236266666625151631
Die     LLLLLLFFFFFFFFFFFFLLLLLLLLLLLLLLLLFFFLLLLLLLLLLLLLLFFFFFFFFF
Viterbi LLLLLLFFFFFFFFFFFFLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLFFFFFFFF

Rolls   222555441666566563564324364131513465146353411126414626253356
Die     FFFFFFFFLLLLLLLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLL
Viterbi FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL

Rolls   366163666466232534413661661163252562462255265252266435353336
Die     LLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Viterbi LLLLLLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

Rolls   233121625364414432335163243633665562466662632666612355245242
Die     FFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLLLLLLLLLLLLLLFFFFFFFFFF
Viterbi FFFFFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLLLLLLLLLLFFFFFFFFFFF
```

Figure 3.5 *The numbers show 300 rolls of a die as described in the example. Below is shown which die was actually used for that roll (F for fair and L for loaded). Under that the prediction by the Viterbi algorithm is shown.*

Application of dynamic programming: fused-lasso

- Fused lasso (Tibshirani et al., 2005) minimizes

$$
-\ell(\boldsymbol{\beta}) + \lambda_1 \sum_{k=1}^{p-1} |\beta_k - \beta_{k-1}| + \lambda_2 \sum_{k=1}^{p} |\beta_k|
$$

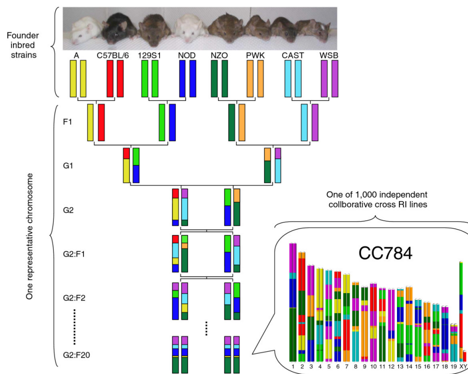
  over $\mathbf{R}^p$ for better recovery of signals that are both sparse and *smooth*

- In many applications, one needs to minimize

$$
O_n(\mathbf{u}) = -\sum_{k=1}^{n} \ell_k(u_k) + \lambda \sum_{k=1}^{n-1} p(u_k, u_{k+1})
$$

  where u_t takes values in a finite space $\mathcal{S}$ and p is a penalty function. A *discrete* (combinatorial) optimization problem.

- A genetic example:



  - Model organism study designs: inbred mice
  - *Goal*: impute the strain origin of inbred mice (Zhou et al., 2012)

- Combinatorial optimization of penalized likelihood.

  - Minimize objective function

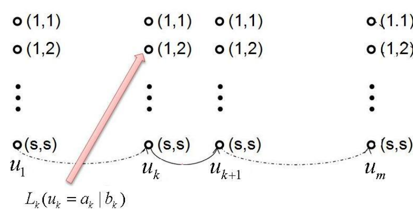
$$O(\mathbf{u}) = -\sum_{k=1}^{n} L_k(u_k) + \sum_{k=1}^{n-1} P_k(u_k, u_{k+1})$$

    by choosing the proper ordered strain origin assignment along the genome

  - $u_k = a_k | b_k$: the ordered strain origin pair
  - L_k: log-likelihood function at marker k - matching imputed genotypes with the observed ones
  - P_k: penalty function for adjacent marker k and $k+1$ - encouraging smoothness of the solution

- Loglikelihood at each marker. At marker k, $u_k = a_k | b_k$: the ordered strain origin pair; r_k / s_k: observed genotype for animal i. Log-penetrance (conditional log-likelihood) is

$$L_k(u_k) = \ln\left[\Pr(r_k / s_k \mid a_k | b_k)\right]$$

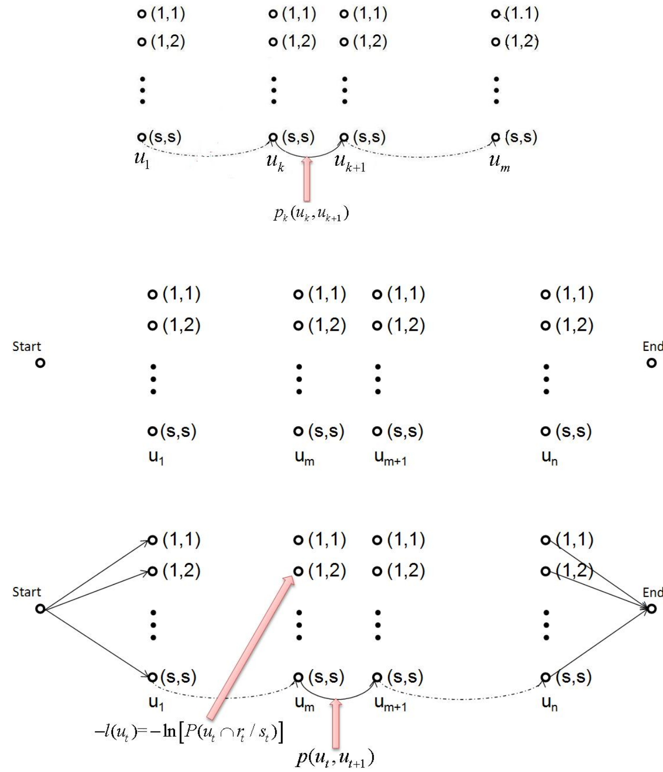


- Penalty for adjacent markers.

– Penalty $P_k(u_k, u_{k+1})$ for each pair of adjacent markers is

$$P_k(u_k, u_{k+1}) = \begin{cases} 0, & a_k = a_{k+1},\ b_k = b_{k+1} \\ -\ln \gamma_i^p(b_{k+1}) + \lambda, & a_k = a_{k+1},\ b_k \neq b_{k+1} \\ -\ln \gamma_i^m(a_{k+1}) + \lambda, & a_k \neq a_{k+1},\ b_k = b_{k+1} \\ -\ln \psi_{ii}^{mp}(a_{k+1}, b_{k+1}) + 2\lambda, & a_k \neq a_{k+1},\ b_k \neq b_{k+1}. \end{cases}$$

– Penalties suppress jumps between strains and guide jumps, when they occur, toward more likely states.



- For each $m = 1, \ldots, n$,

$$O_m(u_m) = \min_{u_1,\ldots,u_{m-1}} \Big[-\sum_{t=1}^{m} \ell_t(u_t) + \lambda \sum_{t=1}^{m-1} p(u_t, u_{t+1}) \Big]$$

beginning with $O_1(u_1) = -\ell_1(u_1)$. And to proceed

$$O_{m+1}(u_{m+1}) = \min_{u_m} \Big[O_m(u_m) - \ell_{m+1}(u_{m+1}) + p(u_m, u_{m+1}) \Big]$$

- Computational time is $O(s^4 n)$, where $n = \#$ markers and $s =$ is # founders.

- More fused-lasso examples.

- Johnson (2013) proposes the dynamic programming algorithm for maximizing the general objective function

$$\sum_{k=1}^{n} e_k(\beta_k) - \lambda \sum_{k=2}^{n} d(\beta_k, \beta_{k-1}),$$

where e is an exponential family log-likelihood and d is a penalty function, e..g, $d(\beta_k, \beta_{k-1}) = 1_{\{\beta_k \neq \beta_{k-1}\}}$

- Applications: L_0-least squares segmentation, fused lasso signal approximator (FLSA), ...

Take home message from this course

- Statistics, the science of *data analysis*, is the applied mathematics in the 21st century.
- In this course, we studied and practiced many (overwhelming?) tools for that help us deliver results faster and more accurate.
  - Operating systems: Linux and scripting basics
  - Programming languages: R (package development, Rcpp, ...), Matlab, Julia
  - Tools for collaborative and reproducible research: Git, R Markdown, sweave
  - Parallel computing: multi-core, cluster, GPU
  - Convex optimization (LP, QP, SOCP, SDP, GP, cone programming)
  - Integer and mixed integer programming
  - Algorithms for sparse regression
  - More advanced optimization methods motivated by modern statistical and machine learning problems, e.g., ALM, ADMM, online algorithms, ...
  - Dynamic programming
  - Advanced topics on EM/MM algorithms (not really ...)

  Of course there are many tools *not* covered in this course, notably Bayesian MCMC machinery. Take a Bayesian course!

- Updated benchmark results. R is upgraded to v3.2.0 and Julia to 0.3.7 since beginning of this course. I re-did the benchmark and did not see notable changes.

  Benchmark code `R-benchmark-25.R` from `http://r.research.att.com/benchmarks/R-benchmark-25.R` covers many commonly used numerical operations used in statistics. We ported to Matlab and Julia and report the run times (averaged over 5 runs) here.

Machine specs: Intel i7 @ 2.6GHz (4 physical cores, 8 threads), 16G RAM, Mac OS 10.9.5.

Test	R 3.2.0	MATLAB R2014a	JULIA 0.3.7
Matrix creation, trans, deformation (2500×2500)	0.80	**0.17**	0.16
Power of matrix (2500×2500, $A.^{1000}$)	0.22	**0.11**	0.22
Quick sort ($n = 7 \times 10^6$)	0.64	**0.24**	0.62
Cross product (2800×2800, $A^T A$)	9.89	**0.35**	0.37
LS solution ($n = p = 2000$)	1.21	**0.07**	0.09
FFT ($n = 2400000$)	0.36	**0.04**	0.14
Eigen-decomposition (600×600)	0.77	**0.31**	0.53
Determinant (2500×2500)	3.52	**0.18**	0.22
Cholesky (3000×3000)	4.08	**0.15**	0.21
Matrix inverse (1600×1600)	2.93	**0.16**	0.19
Fibonacci (vector)	0.29	**0.17**	0.65
Hilbert (matrix)	0.18	**0.07**	0.17
GCD (recursion)	0.28	**0.14**	0.20
Toeplitz matrix (loops)	0.32	**0.0014**	0.03
Escoufiers (mixed)	0.39	0.40	**0.15**

For the simple Gibbs sampler test, R v3.2.0 takes **38.32**s elapsed time. Julia v0.3.7 takes **0.35**s.

- Do not forget course evaluation: `https://classeval.ncsu.edu/secure/prod/cesurvey/`