Biostatistics 615/815 Lecture 10: Hidden Markov Models

Hyun Min Kang

October 4th, 2012

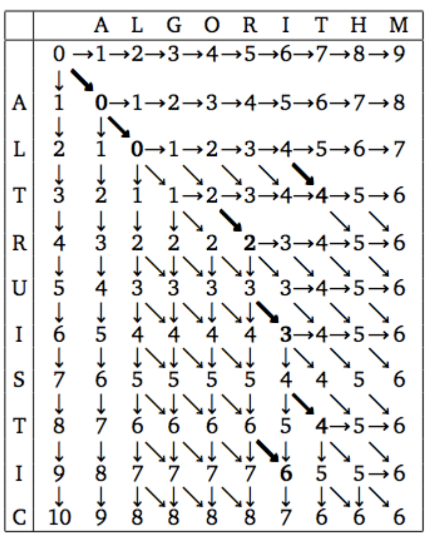
Manhattan Tourist Problem

- Let $C(r, c)$ be the optimal cost from $(0, 0)$ to (r, c)
- Let $h(r, c)$ be the weight from (r, c) to $(r, c + 1)$
- Let $v(r, c)$ be the weight from (r, c) to $(r + 1, c)$
- We can recursively define the optimal cost as

$$C(r,c) = \begin{cases} \min \begin{cases} C(r-1,c) + v(r-1,c) \\ C(r,c-1) + h(r,c-1) \end{cases} & r > 0, c > 0 \\ C(r,c-1) + h(r,c-1) & r = 0, c > 0 \\ C(r-1,c) + v(r-1,c) & r > 0, c = 0 \\ 0 & r = 0, c = 0 \end{cases}$$

- Once $C(r, c)$ is evaluated, it must be stored to avoid redundant computation.

Edit Distance Problem



Dynamic Programming for Edit Distance Problem

- Input strings are $x[1, \cdots, m]$ and $y[1, \cdots, n]$.
- Let $x_i = x[1, \cdots, i]$ and $y_j = y[1, \cdots, j]$ be substrings of x and y.
- Edit distance $d(x, y)$ can be recursively defined as follows

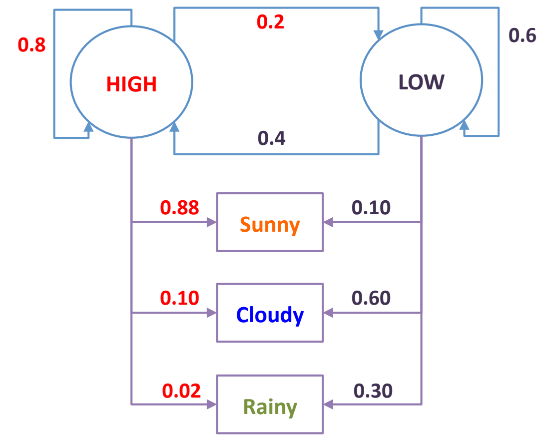
$$d(x_i, y_j) = \begin{cases} i & j = 0 \\ j & i = 0 \\ \min \left\{ \begin{array}{l} d(x_{i-1}, y_j) + 1 \\ d(x_i, y_{j-1}) + 1 \\ d(x_{i-1}, y_{j-1}) + I(x[i] \neq y[j]) \end{array} \right\} & otherwise \end{cases}$$

- Similar to the Manhattan tourist problem, but with 3-way choice.
- Time complexity is $\Theta(mn)$.

Hidden Markov Models (HMMs)

- A Markov model where actual state is unobserved
  - Transition between states are probabilistically modeled just like the Markov process
- Typically there are observable outputs associated with hidden states
  - The probability distribution of observable outputs given an hidden states can be obtained.

An example of HMM



- Direct Observation : (SUNNY, CLOUDY, RAINY)
- Hidden States : (HIGH, LOW)

Mathematical representation of the HMM example

States $S = \{S_1, S_2\}$ = (HIGH, LOW)

Outcomes $O = \{O_1, O_2, O_3\}$ = (SUNNY, CLOUDY, RAINY)

Initial States $\pi_i = \Pr(q_1 = S_i)$, $\pi = \{0.7, 0.3\}$

Transition $A_{ij} = \Pr(q_{t+1} = S_j | q_t = S_i)$

$$A = \begin{pmatrix} 0.8 & 0.2 \\ 0.4 & 0.6 \end{pmatrix}$$

Emission $B_{ij} = b_{q_t}(o_t) = b_{S_i}(O_j) = \Pr(o_t = O_j | q_t = S_i)$

$$B = \begin{pmatrix} 0.88 & 0.10 & 0.02 \\ 0.10 & 0.60 & 0.30 \end{pmatrix}$$

Unconditional marginal probabilities

What is the chance of rain in the day 4?

$$\mathbf{f}(\mathbf{q}_4) = \begin{pmatrix} \Pr(q_4 = S_1) \\ \Pr(q_4 = S_2) \end{pmatrix} = (A^T)^3 \pi = \begin{pmatrix} 0.669 \\ 0.331 \end{pmatrix}$$

$$\mathbf{g}(o_4) = \begin{pmatrix} \Pr(o_4 = O_1) \\ \Pr(o_4 = O_2) \\ \Pr(o_4 = O_3) \end{pmatrix} = B^T \mathbf{f}(\mathbf{q}_4) = \begin{pmatrix} 0.621 \\ 0.266 \\ 0.233 \end{pmatrix}$$

The chance of rain in day 4 is 23.3%

Marginal likelihood of data in HMM

- Let $\lambda = (A, B, \pi)$
- For a sequence of observation $\mathbf{o} = \{o_1, \cdots, o_t\}$,

$$
\begin{aligned}
\Pr(\mathbf{o}|\lambda) &= \sum_{\mathbf{q}} \Pr(\mathbf{o}|\mathbf{q}, \lambda) \Pr(\mathbf{q}|\lambda) \\
\Pr(\mathbf{o}|\mathbf{q}, \lambda) &= \prod_{i=1}^{t} \Pr(o_i|q_i, \lambda) = \prod_{i=1}^{t} b_{q_i}(o_i) \\
\Pr(\mathbf{q}|\lambda) &= \pi_{q_1} \prod_{i=2}^{t} a_{q_{i-1}q_i} \\
\Pr(\mathbf{o}|\lambda) &= \sum_{\mathbf{q}} \pi_{q_1} b_{q_1}(o_1) \prod_{i=2}^{t} a_{q_{i-1}q_i} b_{q_i}(o_i)
\end{aligned}
$$

Naive computation of the likelihood

$$
\Pr(\mathbf{o}|\lambda) = \sum_{\mathbf{q}} \pi_{q_1} b_{q_1}(o_1) \prod_{i=2}^{t} a_{q_{i-1}q_i} b_{q_i}(o_i)
$$

- Number of possible $q = 2^t$ are exponentially growing with the number of observations
- Computational would be infeasible for large number of observations
- Algorithmic solution required for efficient computation.

More Markov Chain Question

- If the observation was (SUNNY,SUNNY,CLOUDY,RAINY,RAINY) from day 1 through day 5, what is the distribution of hidden states for each day?
- Need to know $\Pr(q_t|\mathbf{o}, \lambda)$

Forward and backward probabilities

$$
\begin{aligned}
\mathbf{q}_t^- &= (q_1, \cdots, q_{t-1}), \qquad \mathbf{q}_t^+ = (q_{t+1}, \cdots, q_T) \\
\mathbf{o}_t^- &= (o_1, \cdots, o_{t-1}), \qquad \mathbf{o}_t^+ = (o_{t+1}, \cdots, o_T) \\
\Pr(q_t = i|\mathbf{o}, \lambda) &= \frac{\Pr(q_t = i, \mathbf{o}|\lambda)}{\Pr(\mathbf{o}|\lambda)} = \frac{\Pr(q_t = i, \mathbf{o}|\lambda)}{\sum_{j=1}^{n} \Pr(q_t = j, \mathbf{o}|\lambda)} \\
\Pr(q_t, \mathbf{o}|\lambda) &= \Pr(q_t, \mathbf{o}_t^-, o_t, \mathbf{o}_t^+|\lambda) \\
&= \Pr(\mathbf{o}_t^+|q_t, \lambda) \Pr(\mathbf{o}_t^-|q_t, \lambda) \Pr(o_t|q_t, \lambda) \Pr(q_t|\lambda) \\
&= \Pr(\mathbf{o}_t^+|q_t, \lambda) \Pr(\mathbf{o}_t^-, o_t, q_t|\lambda) \\
&= \beta_t(q_t)\alpha_t(q_t)
\end{aligned}
$$

If $\alpha_t(q_t)$ and $\beta_t(q_t)$ is known, $\Pr(q_t|\mathbf{o}, \lambda)$ can be computed in a linear time.

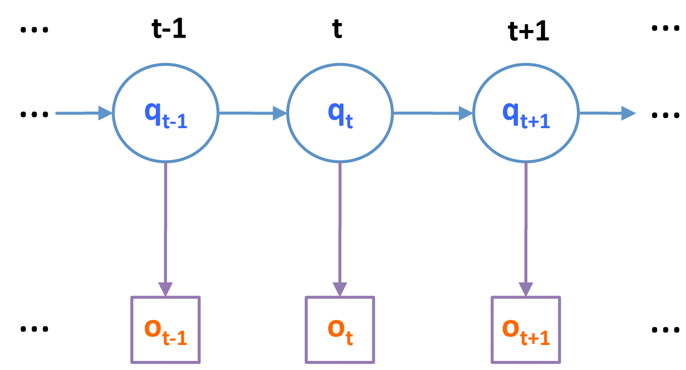
DP algorithm for calculating forward probability

- Key idea is to use $(q_t, o_t) \perp \mathbf{o}_t^- | \mathbf{q}_{t-1}$.
- Each of q_{t-1}, q_t, and q_{t+1} is a Markov blanket.

$$
\begin{aligned}
\alpha_t(i) &= \Pr(o_1, \cdots, o_t, q_t = i|\lambda) \\
&= \sum_{j=1}^{n} \Pr(\mathbf{o}_t^-, o_t, q_{t-1} = j, q_t = i|\lambda) \\
&= \sum_{j=1}^{n} \Pr(\mathbf{o}_t^-, q_{t-1} = j|\lambda) \Pr(q_t = i|q_{t-1} = j, \lambda) \Pr(o_t|q_t = i, \lambda) \\
&= \sum_{j=1}^{n} \alpha_{t-1}(j) a_{ji} b_i(o_t) \\
\alpha_1(i) &= \pi_i b_i(o_1)
\end{aligned}
$$

Conditional dependency in forward-backward algorithms

- Forward : $(q_t, o_t) \perp \mathbf{o}_t^- | \mathbf{q}_{t-1}$.
- Backward : $o_{t+1} \perp \mathbf{o}_{t+1}^+ | \mathbf{q}_{t+1}$.



DP algorithm for calculating backward probability

- Key idea is to use $o_{t+1} \perp \mathbf{o}_{t+1}^+ | \mathbf{q}_{t+1}$.

$$
\begin{aligned}
\beta_t(i) &= \Pr(o_{t+1}, \cdots, o_T|q_t = i, \lambda) \\
&= \sum_{j=1}^{n} \Pr(o_{t+1}, \mathbf{o}_{t+1}^+, q_{t+1} = j|q_t = i, \lambda) \\
&= \sum_{j=1}^{n} \Pr(o_{t+1}|q_{t+1}, \lambda) \Pr(\mathbf{o}_{t+1}^+|q_{t+1} = j, \lambda) \Pr(q_{t+1} = j|q_t = i, \lambda) \\
&= \sum_{j=1}^{n} \beta_{t+1}(j) a_{ij} b_j(o_{t+1}) \\
\beta_T(i) &= 1
\end{aligned}
$$

Putting forward and backward probabilities together

- Conditional probability of states given data

$$
\begin{aligned}
\Pr(q_t = i|\mathbf{o}, \lambda) &= \frac{\Pr(\mathbf{o}, q_t = S_i|\lambda)}{\sum_{j=1}^{n} \Pr(\mathbf{o}, q_t = S_j|\lambda)} \\
&= \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{n} \alpha_t(j)\beta_t(j)}
\end{aligned}
$$

- Time complexity is $\Theta(n^2 T)$.

Finding the most likely trajectory of hidden states

- Given a series of observations, we want to compute

$$\arg\max_{\mathbf{q}} \Pr(\mathbf{q}|\mathbf{o},\lambda)$$

- Define $\delta_t(i)$ as

$$\delta_t(i) = \max_{\mathbf{q}} \Pr(\mathbf{q},\mathbf{o}|\lambda)$$

- Use dynamic programming algorithm to find the 'most likely' path

The Viterbi algorithm

Initialization $\delta_1(i) = \pi b_i(o_1)$ for $1 \leq i \leq n$.

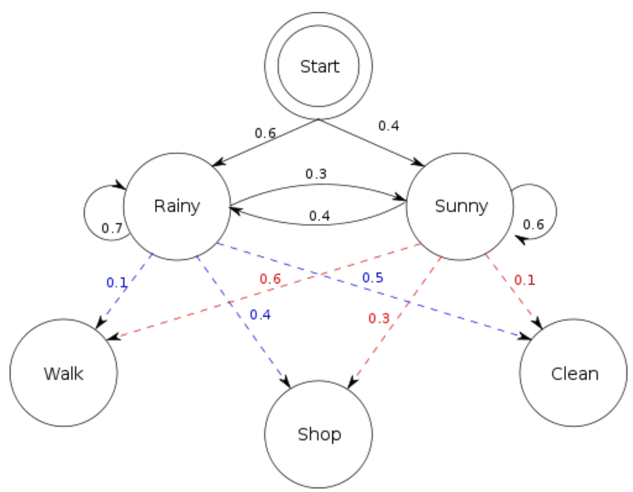
Maintenance $\delta_t(i) = \max_j \delta_{t-1}(j) a_{ji} b_i(o_t)$

$\phi_t(i) = \arg\max_j \delta_{t-1}(j) a_{ji}$

Termination Max likelihood is $\max_i \delta_T(i)$

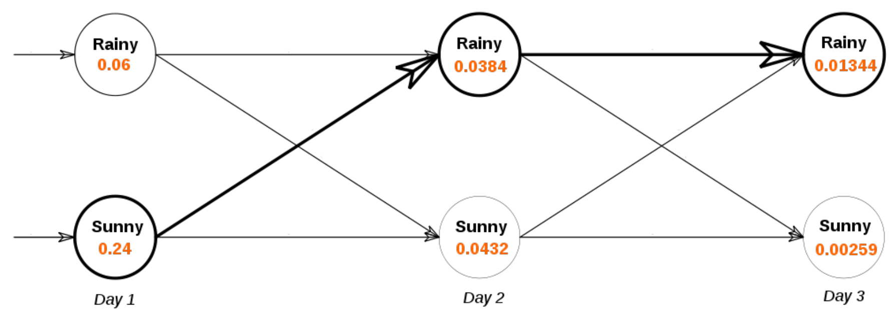
Optimal path can be backtracked using $\phi_t(i)$

An HMM example



An example Viterbi path

- When observations were (walk, shop, clean)
- Similar to Manhattan tourist problem.



A working example : Occasionally biased coin

A generative HMM

- Observations : $O = \{1(Head), 2(Tail)\}$
- Hidden states : $S = \{1(Fair), 2(Biased)\}$
- Initial states : $\pi = \{0.5, 0.5\}$
- Transition probability : $A(i,j) = a_{ij} = \begin{pmatrix} 0.95 & 0.05 \\ 0.2 & 0.8 \end{pmatrix}$
- Emission probability : $B(i,j) = b_i(j) = \begin{pmatrix} 0.5 & 0.5 \\ 0.9 & 0.1 \end{pmatrix}$

Questions

- Given coin toss observations, estimate the probability of each state
- Given coin toss observations, what is the most likely series of states?

Implementing HMM - Matrix615.h

```
#ifndef __MATRIX_615_H  // to avoid multiple inclusion of same headers
#define __MATRIX_615_H
#include <vector>

template <class T>
class Matrix615 {
public:
  std::vector< std::vector<T> > data;
  Matrix615(int nrow, int ncol, T val = 0) {
    data.resize(nrow); // make n rows
    for(int i=0; i < nrow; ++i) {
      data[i].resize(ncol,val); // make n cols with default value val
    }
  }
  int rowNums() { return (int)data.size(); }
  int colNums() { return ( data.size() == 0 ) ? 0 : (int)data[0].size(); }
};

#endif // __MATRIX_615_H
```


HMM Implementations - HMM615.h

```
#ifndef __HMM_615_H
#define __HMM_615_H
#include "Matrix615.h"
class HMM615 {
 public:
  // parameters
  int nStates;  // n : number of possible states
  int nObs;     // m : number of possible output values
  int nTimes;   // t : number of time slots with observations
  std::vector<double> pis; // initial states
  std::vector<int> outs;   // observed outcomes
  Matrix615<double> trans;   // trans[i][j] corresponds to A_{ij}
  Matrix615<double> emis;

  // storages for dynamic programming
  Matrix615<double> alphas, betas, gammas, deltas;
  Matrix615<int> phis;
  std::vector<int> path;
```


HMM Implementations - HMM615.h

```
  HMM615(int states, int obs, int times) : nStates(states), nObs(obs),
    nTimes(times), trans(states, states, 0), emis(states, obs, 0),
    alphas(times, states, 0), betas(times, states, 0),
    gammas(times, states, 0), deltas(times, states, 0),
    phis(times, states, 0)
  {
    pis.resize(nStates);
    path.resize(nTimes);
  }

  void forward();   // given below
  void backward();  //
  void forwardBackward(); // given below
  void viterbi();   //
};
#endif // __HMM_615_H
```

HMM Implementations - HMM615::forward()

```
void HMM615::forward() {
  for(int i=0; i < nStates; ++i) {
    alphas.data[0][i] = pis[i] * emis.data[i][outs[0]];
  }
  for(int t=1; t < nTimes; ++t) {
    for(int i=0; i < nStates; ++i) {
      alphas.data[t][i] = 0;
      for(int j=0; j < nStates; ++j) {
        alphas.data[t][i] += (alphas.data[t-1][j] * trans.data[j][i]
          * emis.data[i][outs[t]]);
      }
    }
  }
}
```


HMM Implementations - HMM615::backward()

```
void HMM615::backward() {
  for(int i=0; i < nStates; ++i) {
    betas.data[nTimes-1][i] = 1;
  }
  for(int t=nTimes-2; t >=0; --t) {
    for(int i=0; i < nStates; ++i) {
      betas.data[t][i] = 0;
      for(int j=0; j < nStates; ++j) {
        betas.data[t][i] += (betas.data[t+1][j] * trans.data[i][j]
                            * emis.data[j][outs[t+1]]);
      }
    }
  }
}
```


HMM Implementations - HMM615::forwardBackward()

```
void HMM615::forwardBackward() {
  forward();
  backward();

  for(int t=0; t < nTimes; ++t) {
    double sum = 0;
    for(int i=0; i < nStates; ++i) {
      sum += (alphas.data[t][i] * betas.data[t][i]);
    }
    for(int i=0; i < nStates; ++i) {
      gammas.data[t][i] = (alphas.data[t][i] * betas.data[t][i])/sum;
    }
  }
}
```


HMM Implementations - HMM615::viterbi()

```
void HMM615::viterbi() {
  for(int i=0; i < nStates; ++i) {
    deltas.data[0][i] = pis[i] * emis.data[i][ outs[0] ];
   }
  for(int t=1; t < nTimes; ++t) {
    for(int i=0; i < nStates; ++i) {
      int maxIdx = 0;
      double maxVal = deltas.data[t-1][0] * trans.data[0][i]
                          * emis.data[i][ outs[t] ];
      for(int j=1; j < nStates; ++j) {
        double val = deltas.data[t-1][j] * trans.data[j][i]
                          * emis.data[i][ outs[t] ];
        if ( val > maxVal ) { maxIdx = j; maxVal = val; }
      }
      deltas.data[t][i] = maxVal;
      phis.data[t][i] = maxIdx;
    }
  }
```

HMM Implementations - HMM615::viterbi() (cont'd)

```
    // backtrack viterbi path
    double maxDelta = deltas.data[nTimes-1][0];
    path[nTimes-1] = 0;
    for(int i=1; i < nStates; ++i) {
      if ( maxDelta < deltas.data[nTimes-1][i] ) {
        maxDelta = deltas.data[nTimes-1][i];
        path[nTimes-i] = i;
      }
    }
    for(int t=nTimes-2; t >= 0; --t) {
      path[t] = phis.data[t+1][ path[t+1] ];
    }
  }
```


HMM Implementations - biasedCoin.cpp

```
#include <iostream>
#include <iomanip>
int main(int argc, char** argv) {
  std::vector<int> toss;
  std::string tok;
  while( std::cin >> tok ) {
    if ( tok == "H" ) toss.push_back(0);
    else if ( tok == "T" ) toss.push_back(1);
    else {
      std::cerr << "Cannot recognize input " << tok << std::endl;
      return -1;
    }
  }

  int T = toss.size();
  HMM615 hmm(2, 2, T);

  hmm.trans.data[0][0] = 0.95; hmm.trans.data[0][1] = 0.05;
  hmm.trans.data[1][0] = 0.2;  hmm.trans.data[1][1] = 0.8;
```


HMM Implementations - biasedCoin.cpp

```
  hmm.emis.data[0][0] = 0.5; hmm.emis.data[0][1] = 0.5;
  hmm.emis.data[1][0] = 0.9; hmm.emis.data[1][1] = 0.1;

  hmm.pis[0] = 0.5; hmm.pis[1] = 0.5;

  hmm.outs = toss;

  hmm.forwardBackward();
  hmm.viterbi();

  std::cout << "TIME\tTOSS\tP(FAIR)\tP(BIAS)\tMLSTATE" << std::endl;
  std::cout << std::setiosflags(std::ios::fixed) << std::setprecision(4);
  for(int t=0; t < T; ++t) {
    std::cout << t+1 << "\t" << (toss[t] == 0 ? "H" : "T") << "\t"
        << hmm.gammas.data[t][0] << "\t" << hmm.gammas.data[t][1] << "\t"
        << (hmm.path[t] == 0 ? "FAIR" : "BIASED" ) << std::endl;
  }
  return 0;
}
```


Example runs

```
$ cat ~hmkang/Public/615/data/toss.20.txt | ~hmkang/Public/615/bin/biasedCoin
TIME    TOSS    P(FAIR) P(BIAS) MLSTATE
1       H       0.5950  0.4050  FAIR
2       T       0.8118  0.1882  FAIR
3       H       0.8071  0.1929  FAIR
4       T       0.8584  0.1416  FAIR
5       H       0.7613  0.2387  FAIR
6       H       0.7276  0.2724  FAIR
7       T       0.7495  0.2505  FAIR
8       H       0.5413  0.4587  BIASED
9       H       0.4187  0.5813  BIASED
10      H       0.3533  0.6467  BIASED
11      H       0.3301  0.6699  BIASED
12      H       0.3436  0.6564  BIASED
13      H       0.3971  0.6029  BIASED
14      T       0.5028  0.4972  BIASED
15      H       0.3725  0.6275  BIASED
16      H       0.2985  0.7015  BIASED
17      H       0.2635  0.7365  BIASED
18      H       0.2596  0.7404  BIASED
19      H       0.2858  0.7142  BIASED
20      H       0.3482  0.6518  BIASED
```

Summary

Today - Hidden Markov Models

- Forward-backward algorithm
- Viterbi algorithm
- Biased Coin Example

Next Week

- Basic Usage of STL containers
- Using Eigen Library
- Interfacing between R and C++