COSC/DATA 405/505

Modeling and Simulation

Outline

- A Simple Hidden Markov Model
- Dynamic Programming - How to get from New York to LA in a hurry
- The Basic Idea of the Viterbi Algorithm
- Built-in Software
- An Example Using Simulated Data
- Application to Windspeed Data

```
set.seed(222696)   # use this to reproduce output
```

A Simple Hidden Markov Model

Suppose we observe data on a discrete random variable Y:
$1, 0, 0, 0, 1, 1, 0$**.**

We can model this as Bernoulli data, but we suspect there is some hidden dependence.

So we choose to model it as

$$Y_j = B(0.25 + 0.5X_j)$$

where X_j is a Bernoulli random variable which is part of an underlying (hidden) Markov chain.

Note that $P(Y_j = 1|X_j = 1) = .75$ and $P(Y_j = 1|X_j = 0) = .25$. These are called *emission probabilities*.

A Simple Hidden Markov Model

Since X_j is part of a Markov chain, we need a transition matrix. Since there are two hidden states, the transition matrix is 2×2, For example,

$$P = \begin{bmatrix} 1/3 & 2/3 \\ 2/3 & 1/3 \end{bmatrix}.$$

Together, the emission probabilities and the transition matrix make up a Hidden Markov Model (HMM).

A Simple Hidden Markov Model

Usually, we do not know the emission probabilities P_E or the transition probabilities P.

These are usually estimated by maximizing the likelihood function:

$$L(P_E, P) = P(Y_1, Y_2, \ldots, Y_n).$$

Because the Y_j's are not independent, we need to be careful how to evaluate the likelihood.

A Simple Hidden Markov Model

What we can calculate easily:

$$P(Y_1, Y_2, \ldots, Y_n | X_1, X_2, \ldots, X_n) = \Pi_{j=1}^{n} (.25 + .5X_j)^{Y_j} (.75 - .5X_j)^{1-Y_j}.$$

$$P(X_1, \ldots, X_n) = P(X_1)P(X_2|X_1) \cdots P(X_n|X_{n-1})$$

Taking the product of these, we can calculate:

$$P(X_1, \ldots, X_n, Y_1, \ldots, Y_n)$$

We then have to sum over all combinations of the Xs to get the likelihood, a big job.

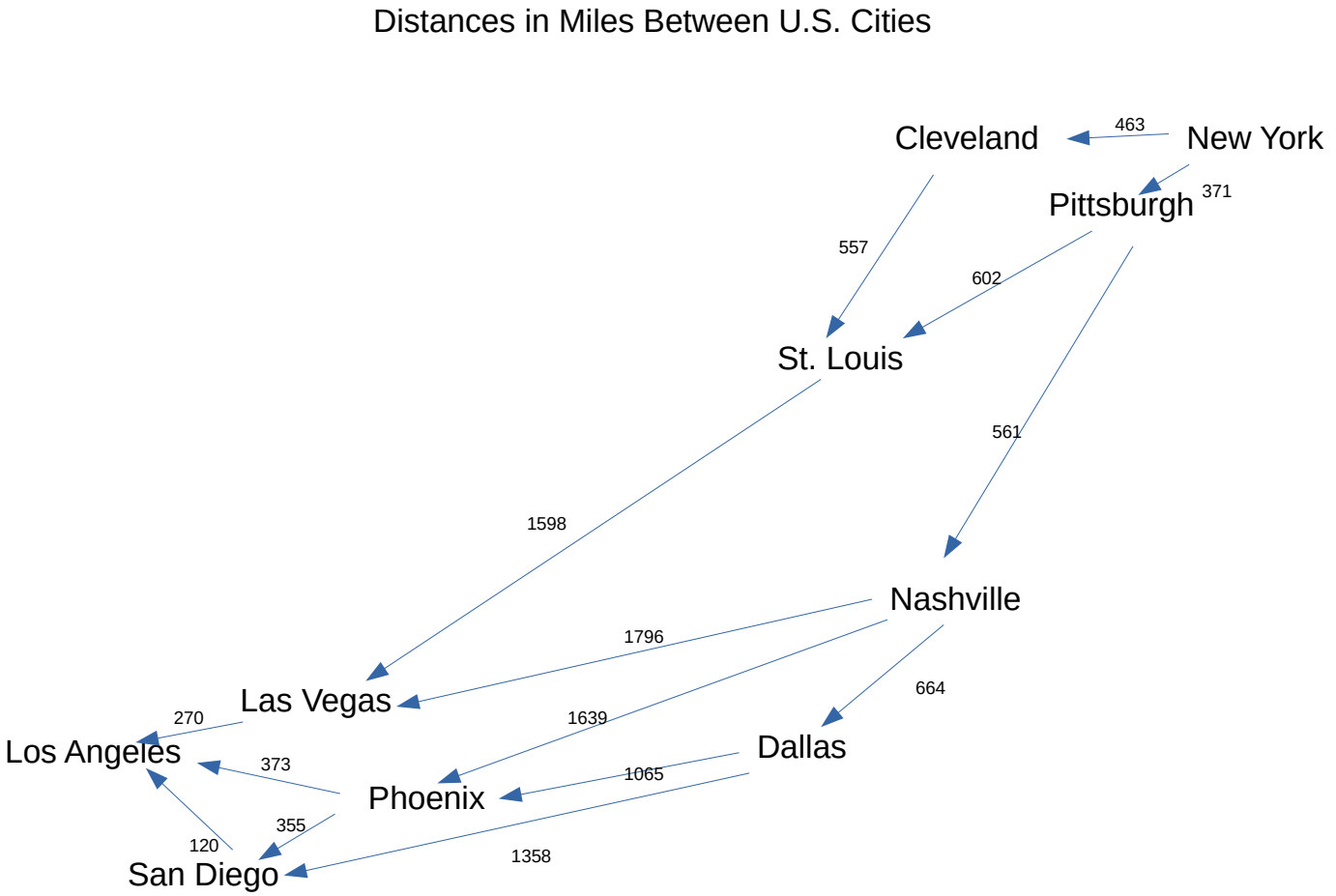
Dynamic Programming

Dynamic programming is an optimization procedure due to Richard Bellman (early 1960's) which efficiently finds the minimum distance route through a network.

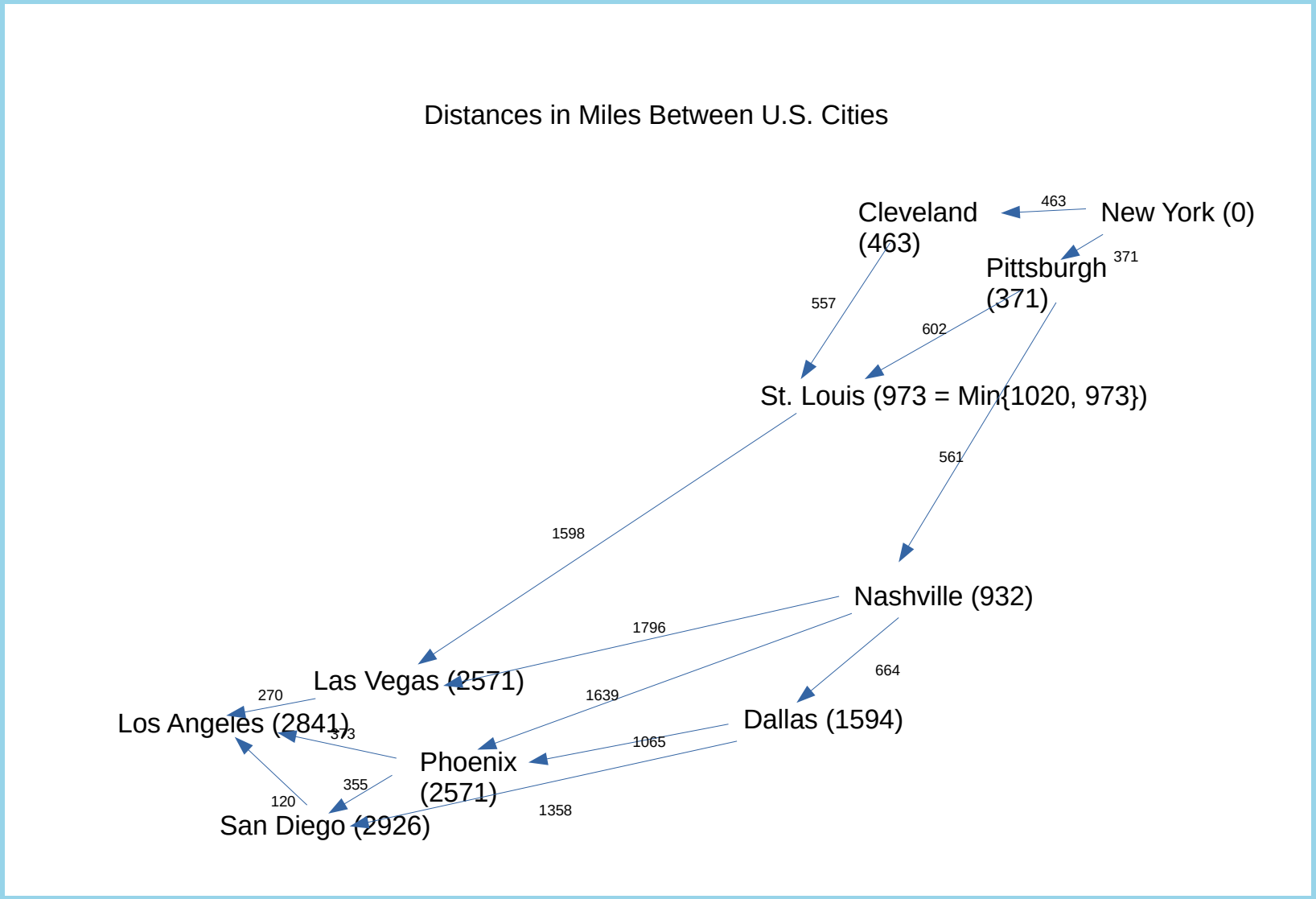
Consider the problem of driving from New York to Los Angeles. There are many possible routes.

Dynamic programming allows us to break the bigger network problem into a set of smaller network problems which can be solved recursively.

Find Shortest Route from New York to Los Angeles



Find Shortest Route from New York to Los Angeles



Find Shortest Route from New York to Los Angeles

Distances in Miles Between U.S. Cities

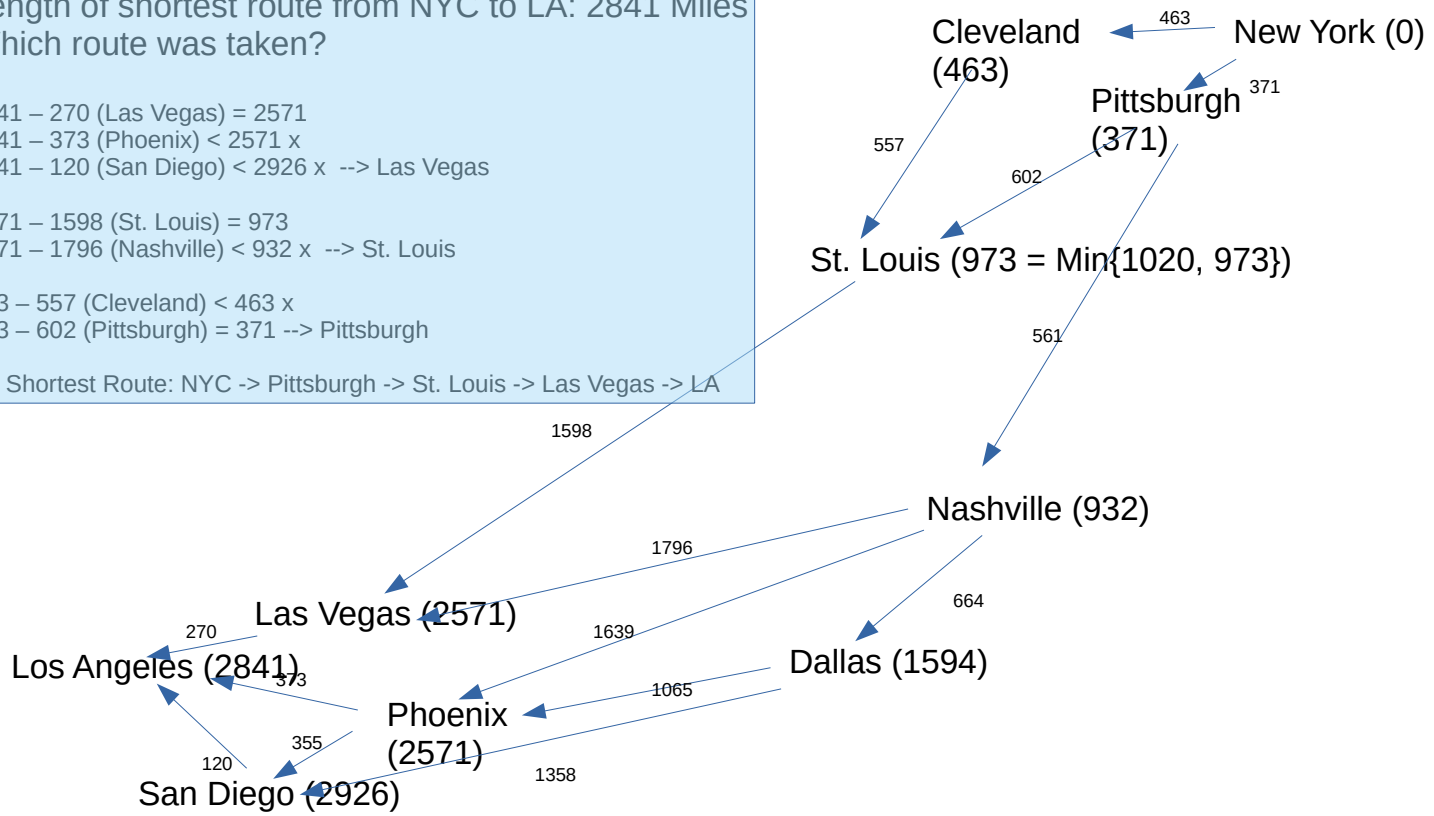
Length of shortest route from NYC to LA: 2841 Miles
Which route was taken?

2841 – 270 (Las Vegas) = 2571
2841 – 373 (Phoenix) < 2571 x
2841 – 120 (San Diego) < 2926 x --> Las Vegas

2571 – 1598 (St. Louis) = 973
2571 – 1796 (Nashville) < 932 x --> St. Louis

973 – 557 (Cleveland) < 463 x
973 – 602 (Pittsburgh) = 371 --> Pittsburgh

--> Shortest Route: NYC -> Pittsburgh -> St. Louis -> Las Vegas -> LA



The Basic Idea of the Viterbi Algorithm ... and Something Better

Dynamic programming can also be used to maximize quantities through a network.

The Viterbi algorithm employs dynamic programming to find the most likely sequence of hidden Markov chain states which could give rise to the observed data.

Using the Viterbi path, we can calculate a kind of maximum likelihood estimate for the emission probability matrix and the transition matrix, based on the empirical proportions of time we observe the Y values, given the "observed" X values, and given the proportion of time we "observe" X values, given the preceding X values.

The Baum-Welch algorithm is an implementation of the Expectation-Maximization (EM) algorithm which calculates the exact maximum likelihood estimates of the parameters, given the Y observations.

Built-in Software - the *depmixS4* package

The function `depmix()` can be used to fit general purpose Hidden Markov Models.

Its use goes beyond the scope of this module.

Instead, we focus on a simpler-to-use package, *HMM*, which can be used to fit and simulate discrete-time discrete-state Hidden Markov Models.

Built-in Software - the *HMM* package

We illustrate the use of the software for the Hidden Markov Model for which the transition matrix for the hidden 2 state Markov chain X is

$$P = \begin{bmatrix} 1/3 & 2/3 \\ 2/3 & 1/3 \end{bmatrix}$$

and the emission matrix for the probability distributions of the observed values Y, given X is

$$E = \begin{bmatrix} 3/4 & 1/4 \\ 1/4 & 3/4 \end{bmatrix}$$

The state space of the Markov chain is $S = \{0, 1\}$.

The first row of the emission corresponds to the distribution of Y, given $X = 0$ and the second row corresponds to $X = 1$.

We assume that the observed data are $Y = 1, 0, 0, 0, 1, 1, 0$.

Built-in Software - the *HMM* package

We load the package, enter the information about the transition and emission matrices as well as the observed data.

```
library(HMM)
```

```
hmm <- initHMM(c("0","1"), c("0","1"),
         transProbs=matrix(c(1, 2, 2, 1)/3,2),
         emissionProbs=matrix(c(3, 1, 1, 3)/4, 2))
observations <- as.character(c(1, 0, 0, 0, 1, 1, 0))
```

Using the `viterbi` function, we can calculate the Viterbi path, the most likely X values:

```
viterbi <- viterbi(hmm, observations)
print(viterbi)


## [1] "1" "0" "1" "0" "1" "1" "0"
```

Training (Estimating) the HMM from Y Observations

Based on the preceding Viterbi sequence, we could get estimates of the transition matrix elements by calculating the proportion of the various transitions.

In the example, the Markov chain appears to have visited state 0 two times, not including the last value.

The Markov chain entered state 1 both times, so we would estimate the P_{01} to be 1 and P_{00} as 0. Similarly, we would estimate P_{10} as $3/4$ and P_{11} as $1/4$.

We can also estimate the emission probabilities by estimating the distribution of Y for each value of X: $P(Y = 0|X = 0) = 2/3$ and $P(Y = 1|X = 0) = 1/3$. $P(Y = 0|X = 1) = 1/4$ and $P(Y = 1|X = 1) = 3/4$.

Training (Estimating) the HMM from Y Observations

1. **Begin with an initial guess as to the transition matrix and emission matrix.**

2. **Apply the Viterbi algorithm to the Y observations to obtain the most probable set of X observations, using the most recent estimates of the transition and emission matrices.**

3. **Use the Viterbi sequence of X's to estimate the transition matrix probabilities and the emission matrix probabilities.**

4. **Return to step 2, unless the transition and emission matrices have converged to within a given tolerance.**

The function `viterbiTraining` is an implementation of this algorithm.

An Example Using Simulated Data

We simulate X's from a two state Markov chain, and use these to generate Y observations which take values 0, 1 and 2, according to different distributions, depending on the corresponding value of X.

The transition matrix for the hidden Markov chain X:

$$P = \begin{bmatrix} 0.1 & 0.9 \\ 0.2 & 0.8 \end{bmatrix}$$

The emission matrix for the probability distributions of the observed values Y, given X:

$$E = \begin{bmatrix} 0.75 & 0.2 & 0.05 \\ 0.05 & 0.4 & 0.55 \end{bmatrix}$$

The first row corresponds to the distribution of Y, given $X = 0$ and the second row corresponds to $X = 1$.

An Example Using Simulated Data

```
P <- matrix(c(.1, .9, .2, .8), nrow=2, byrow=TRUE)
E <- matrix(c(.75, .2, 0.05, .05, 0.4, .55), nrow=2, byrow=TRUE)
current.state <- 1; n <- 500
X <- numeric(n); Y <- numeric(n)
for (i in 1:n) {
    current.state <- sample(0:1, size = 1, prob = P[current.state+1,])
    X[i] <- current.state
    Y[i] <- sample(0:2, size = 1, prob = E[current.state+1,])
}
```

First few simulated observations:

```
Y[1:5]

## [1] 1 0 1 2 1
```

An Example Using Simulated Data

Fitting the HMM to the data, using an arbitrary starting guess for the emission matrix and the transition matrix.

```
hmm <- initHMM(c("0","1"), c("0","1", "2"),
        transProbs=matrix(c(.5, .5, .5, .5),2),
        emissionProbs=matrix(c(.5, .4, .1, .8, .05, .15), 2))
observations <- as.character(Y)
simDataFit <- viterbiTraining(hmm, observations)
```

An Example Using Simulated Data

The output from the algorithm includes the estimated transition matrix and emission matrix:

```
print(simDataFit$hmm$transProbs)   # transition matrix estimate

##     to
## from         0         1
##    0 0.1276596 0.8723404
##    1 0.2024691 0.7975309
```

```
print(simDataFit$hmm$emissionProbs)   # emission matrix estimate

##       symbols
## states 0         1         2
##      0 1 0.0000000 0.0000000
##      1 0 0.4408867 0.5591133
```

The estimated matrices are not terribly far from the truth, but there is still considerable error. Note that the sample size is fairly large, and we are fitting a very simple model.

An Example Using Simulated Data

We can see how well our model did by comparing the original hidden Markov chain values with the most probable sequence that would be generated from the fitted Hidden Markov Model:

```
table(viterbi(simDataFit$hmm, observations), X)
```

```
##     X
##         0   1
##    0   69  25
##    1   31 375
```

Application to the Windspeed Data

Data preparation:

```
source("wind.R")
ws <- wind$h12  # Winnipeg noon hour windspeed (kmh)
wsCut <- cut(ws, c(-1e-10, 15, 22, 35, 45, 80))
levels(wsCut) # see what the cut function did

## [1] "(-1e-10,15]" "(15,22]"     "(22,35]"     "(35,45]"
## [5] "(45,80]"

levels(wsCut) <- c("N", "L", "M", "H", "E")
  # Nil, Low, Moderate, High, Extreme
WindStates <- factor(wsCut, ordered = TRUE)
```

Application to the Windspeed Data

Set up an initial guess for the HMM:

```
WindHMM <- initHMM(c("0","1","2"), levels(WindStates),
      transProbs=matrix(c(2, 1, 1, 1, 2, 2, 3, 3, 3)/6, 3),
      emissionProbs=matrix(c(4, 2, 0, 3, 2, 1, 2, 2, 2, 1,
    2, 3, 0, 2, 4)/10, nrow=3))
```

Application to the Windspeed Data

Viewing the transition matrix and emissions probabilities for the initial guess:

```
print(WindHMM$transProbs)

##      to
## from     0     1   2
##    0 0.333 0.167 0.5
##    1 0.167 0.333 0.5
##    2 0.167 0.333 0.5
```

```
print(WindHMM$emissionProbs)

##        symbols
## states   N   L   M   H   E
##      0 0.4 0.3 0.2 0.1 0.0
##      1 0.2 0.2 0.2 0.2 0.2
##      2 0.0 0.1 0.2 0.3 0.4
```

Application to the Windspeed Data

Interpretation of the initial guess emissions probability matrix P_E:

$$P_E[i, j] = P(Y = j | X = i)$$

where X is the current state of the hidden Markov chain, and Y is the current observation.

e.g. $P_E[2, 3] = 0.2$ so the probability of Moderate wind when in Markov Chain state 1 is 0.2.

Application to the Windspeed Data

Fitting the Hidden Markov Model by finding transition probabilities and emission probabilities to maximize the likelihood:

```
WindHMMFit <- viterbiTraining(WindHMM, WindStates)
```

Application to the Windspeed Data

Viewing the transition matrix and emissions probabilities for the fitted model:

```
print(WindHMMFit$hmm$transProbs)

##      to
## from      0      1      2
##    0 0.8047 0.0000 0.1953
##    1 0.0000 0.0000 1.0000
##    2 0.2020 0.2424 0.5556
```

```
print(WindHMMFit$hmm$emissionProbs)

##        symbols
## states      N      L       M      H      E
##      0 0.5560 0.3739 0.07007 0.0000 0.0000
##      1 0.5623 0.4377 0.00000 0.0000 0.0000
##      2 0.0000 0.0000 0.79976 0.1557 0.0445
```

Application to the Windspeed Data

Simulate from the fitted model:

```
WindSim <- simHMM(WindHMMFit$hmm, length(WindStates))
```

We have simulated the same number of observations as in the original data set.

Application to the Windspeed Data

Compare run lengths of the various states:

```
table(rle(as.character(WindStates)))
```

```
##         values
## lengths    E    H    L    M    N
##       1  133  409 1017 1028 1031
##       2    7   49  217  410  307
##       3    1    6   37  175  115
##       4    0    0   13   74   39
##       5    0    0    4   28   22
##       6    0    0    3   14    8
##       7    0    0    0    2    6
##       8    0    0    1    3    4
##       9    0    0    0    1    2
```

Application to the Windspeed Data

```
table(rle(WindSim$observation))
```

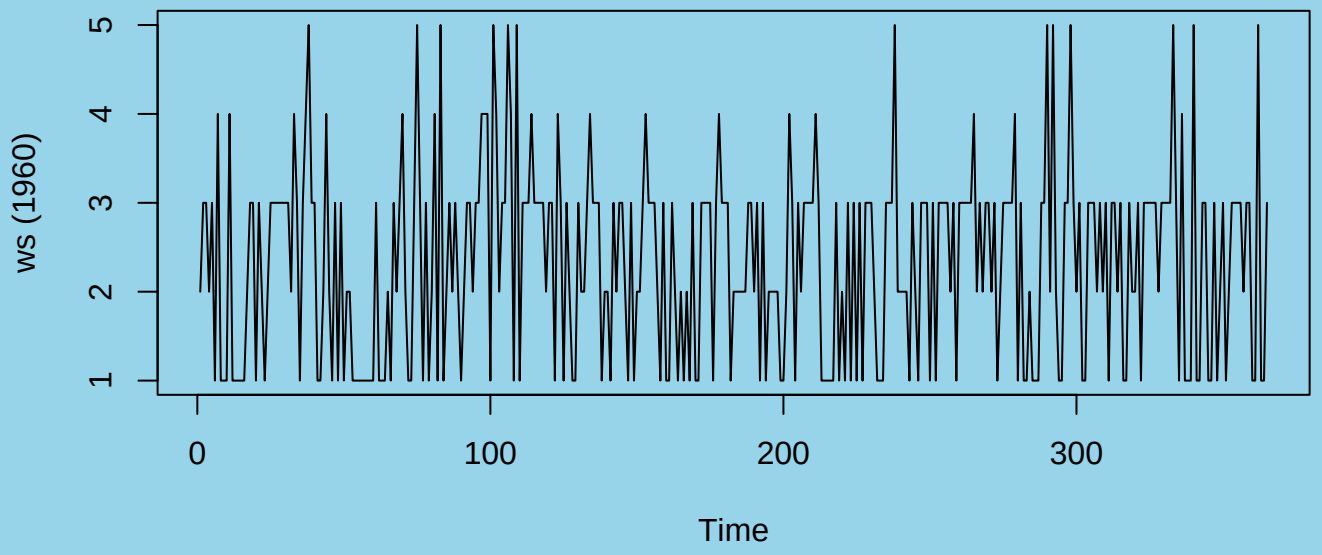
```
##         values
## lengths    E    H    L    M    N
##       1  148  437 1031  943 1107
##       2    4   51  176  404  291
##       3    0    4   58  178  107
##       4    0    0   15   87   36
##       5    0    0    2   40   19
##       6    0    0    2    9    5
##       7    0    0    0    8    3
##       8    0    0    0    3    2
##       9    0    0    0    0    2
##       12   0    0    0    1    1
```

We have use the `rle()` (Run Length Encoding) function which computes the lengths and values of runs of equal values in a vector.

Application to the Windspeed Data

One year of the original data:

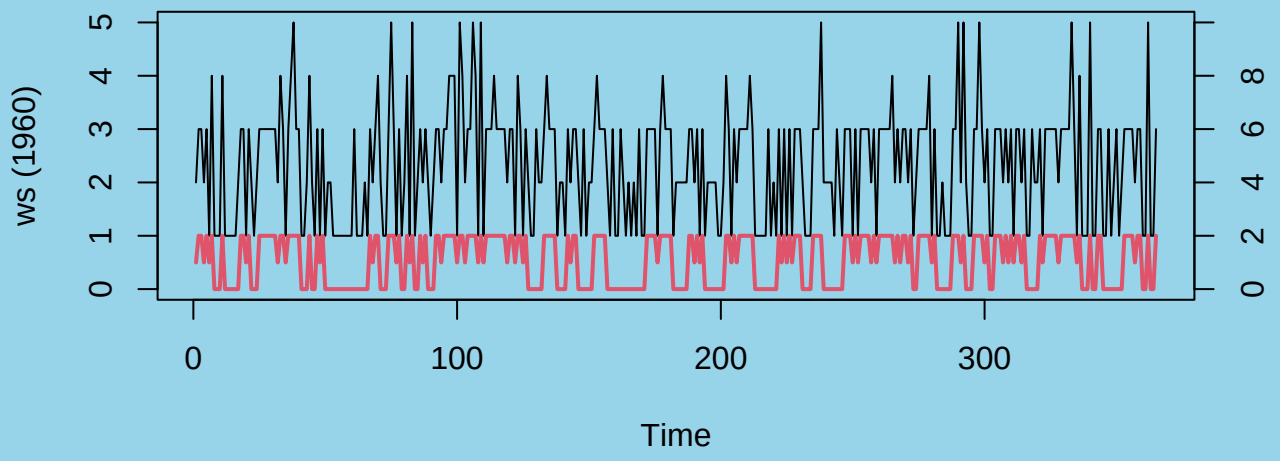
```
par(mar=c(4, 4, 1, 1))
ts.plot(WindStates[1:365], ylab="ws (1960)")
```



Application to the Windspeed Data

Including information on the most likely hidden states:

```
HiddenStates <- viterbi(WindHMMFit$hmm, WindStates[1:365])
par(mar=c(4, 4, 1, 4))
ts.plot(WindStates[1:365], ylab="ws (1960)", ylim=c(0, 5))
lines(1:365, as.numeric(HiddenStates)/2, col=2, lwd=2)
axis(side = 4, at=0:5, labels=seq(0,10,2))
```



Summary

1. **Hidden Markov Models (HMM) provide a way to model data that have complex dependencies in a structured way: a Markov chain for the hidden information, together with a matrix of probability distributions for the observed data.**

2. **The Viterbi algorithm provides a basic way to train the data, based on finding the most likely path through the hidden Markov states.**

3. **Hidden Markov Models are an important tool in diverse areas such as climate science, genomics, and speech processing.**