ICSI431/ICSI531 Data Mining
Lecture 8-B
Markov Model, Hidden Markov Model

Feng Chen

fchen5@albany.edu

http://www.cs.albany.edu/~fchen/course/2016-ICSI-431-531

Slides adapted from Eric Conklin, Lindsay Kulzer, Ian Zink, Dan Pallas, Amy Sha, Patrick Cillen, Kyuseo Park, Kisuk Yoo

Content

- Applications
- Vacation Time Example
- Markov Model
- Back on Vacation
  - With a spending spree!
- Hidden Markov Model
- Python

Applications

- The games “Snakes and Ladders” and “Hi Ho! Cherry-O” can both be represented exactly as Markov Chains
- Google’s PageRank algorithm behaves like a Markov Chain over the graph of the Web
- Markov Chains are used in queuing theory for analyzing and optimizing the performance of communications networks

Vacation Time

- Using techniques learned up to this point, how would you create a system to recommend a vacation trip?
- It's probably no surprise…
- This can be accomplished easily with a Markov Model!

Vacation Time

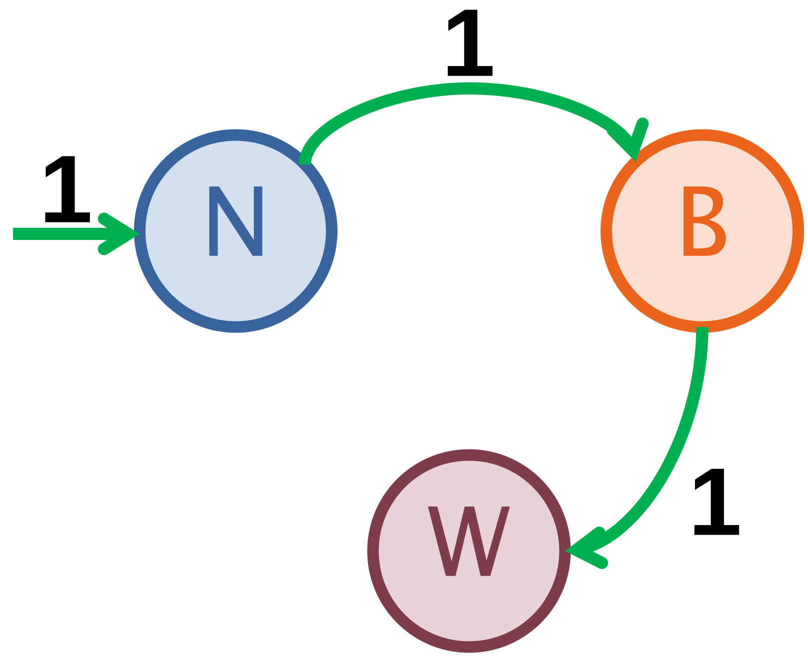
Person A	New York City →	Boston →	Washington D.C.
Person B	Washington D.C. →	New York City →	Boston
Person C	New York City →	Washington D.C. →	Boston
Person D	Boston →	Washington D.C. →	New York City

- How can a Markov Model describe these sequences of city visitations?

Person A’s Vacation

Person A

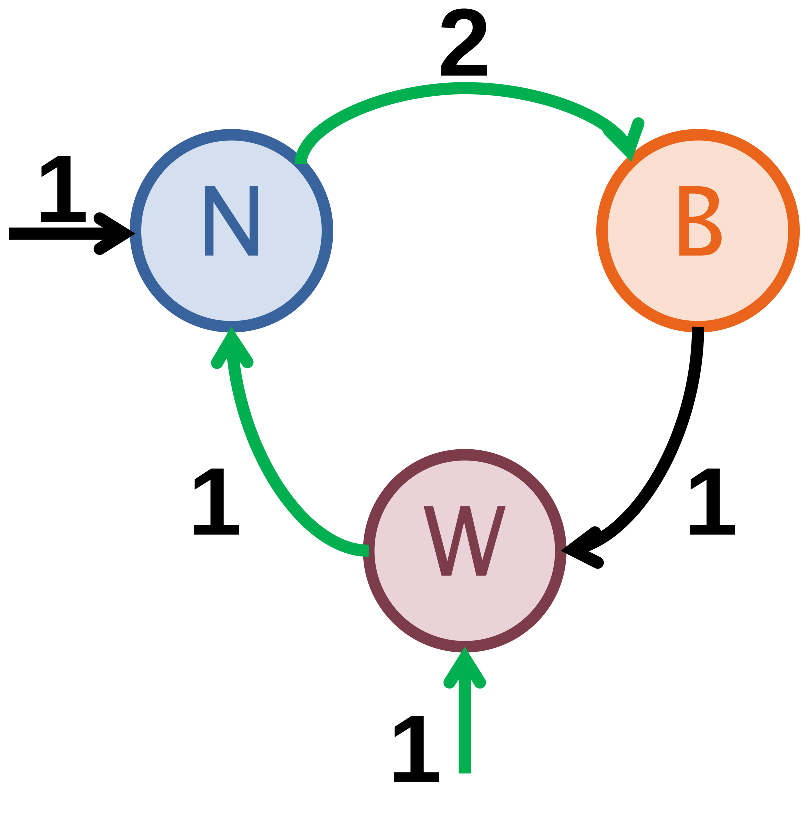
New York City → Boston → Washington D.C.



Person B's Vacation

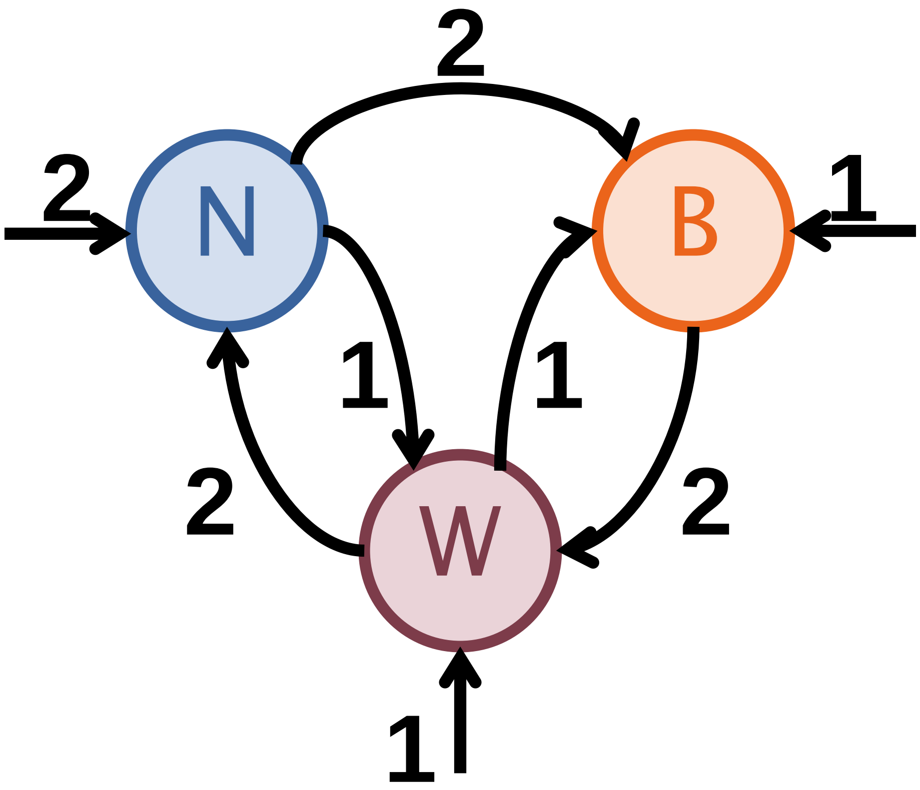
Person B

Washington D.C. → New York City → Boston



Everybody's Vacation

- Completed diagram

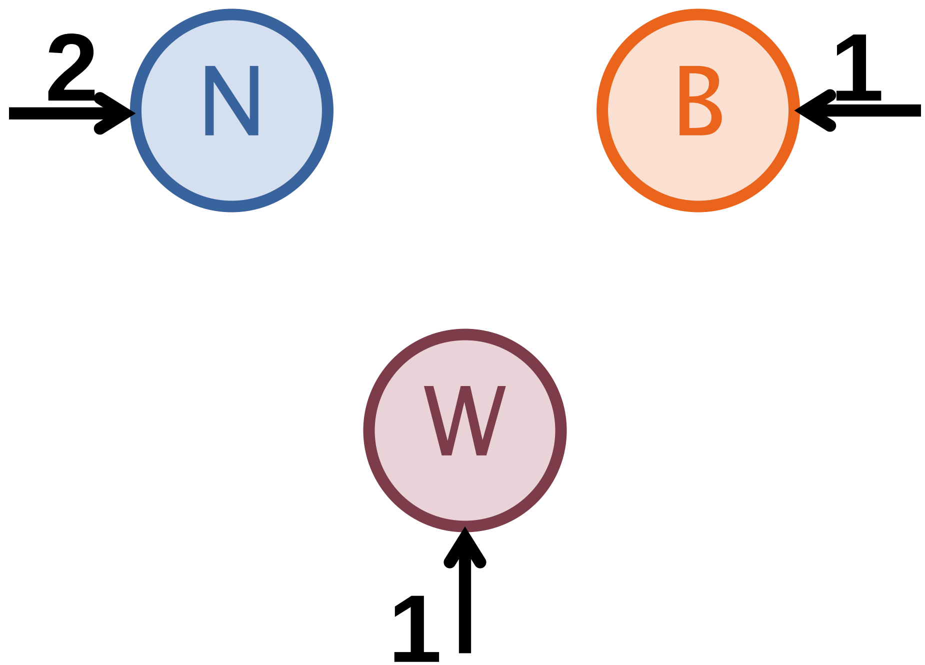


- Now what?

Initial Cities

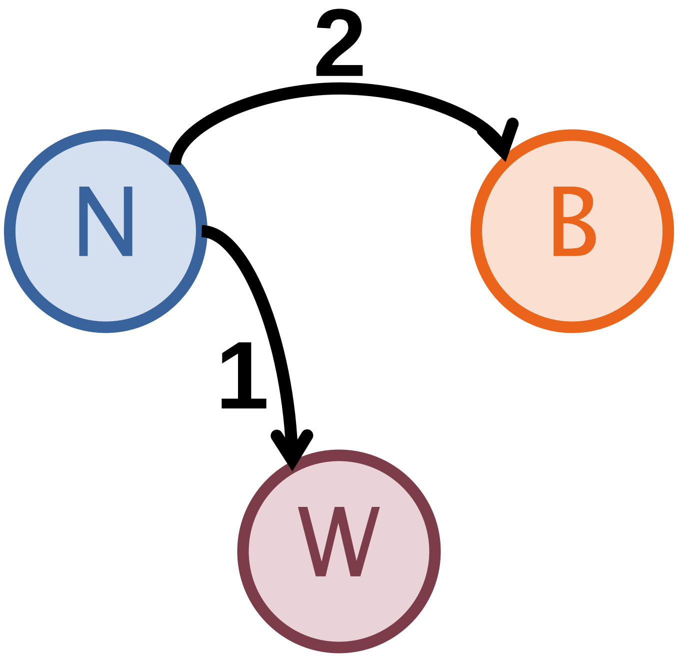
- Starting city popularity

$$\Pi = [1/2,\ 1/4,\ 1/4]$$



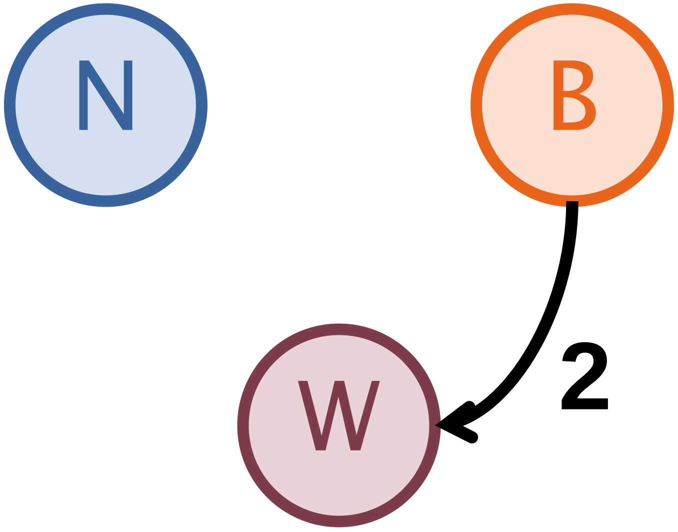
New York City Transitions

- N→W=1/3
- N→B=2/3



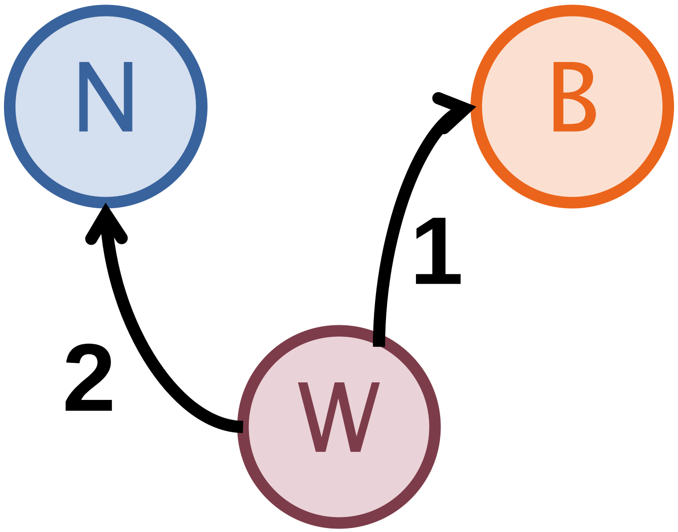
Boston Transitions

- B→W=1



Washington DC Transitions

- W→N=2/3
- W→B=1/3



Transition Matrix

$$A = \begin{array}{c c} & \begin{array}{ccc} \mathbf{N} & \mathbf{B} & \mathbf{W} \end{array} \\ \begin{array}{c} \mathbf{N} \\ \mathbf{B} \\ \mathbf{W} \end{array} & \left[\begin{array}{ccc} \mathbf{0} & \mathbf{2/3} & \mathbf{1/3} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \\ \mathbf{2/3} & \mathbf{1/3} & \mathbf{0} \end{array} \right] \end{array}$$

- Trans-whose-a-what???

Trans-whose-a-what?

- New York City…

$$A = \begin{array}{c} \\ N \\ B \\ W \end{array} \begin{array}{ccc} N & B & W \\ \left[\begin{array}{ccc} 0 & 2/3 & 1/3 \\ 0 & 0 & 1 \\ 2/3 & 1/3 & 0 \end{array}\right] \end{array}$$

- To Washington DC = 1/3

Trans-whose-a-what?

- Washington DC…

$$A = \begin{array}{c} \\ N \\ B \\ W \end{array} \begin{array}{c} \begin{array}{ccc} N & B & W \end{array} \\ \begin{bmatrix} 0 & 2/3 & 1/3 \\ 0 & 0 & 1 \\ 2/3 & 1/3 & 0 \end{bmatrix} \end{array}$$

- To Boston = 1/3

Trans-whose-a-what?

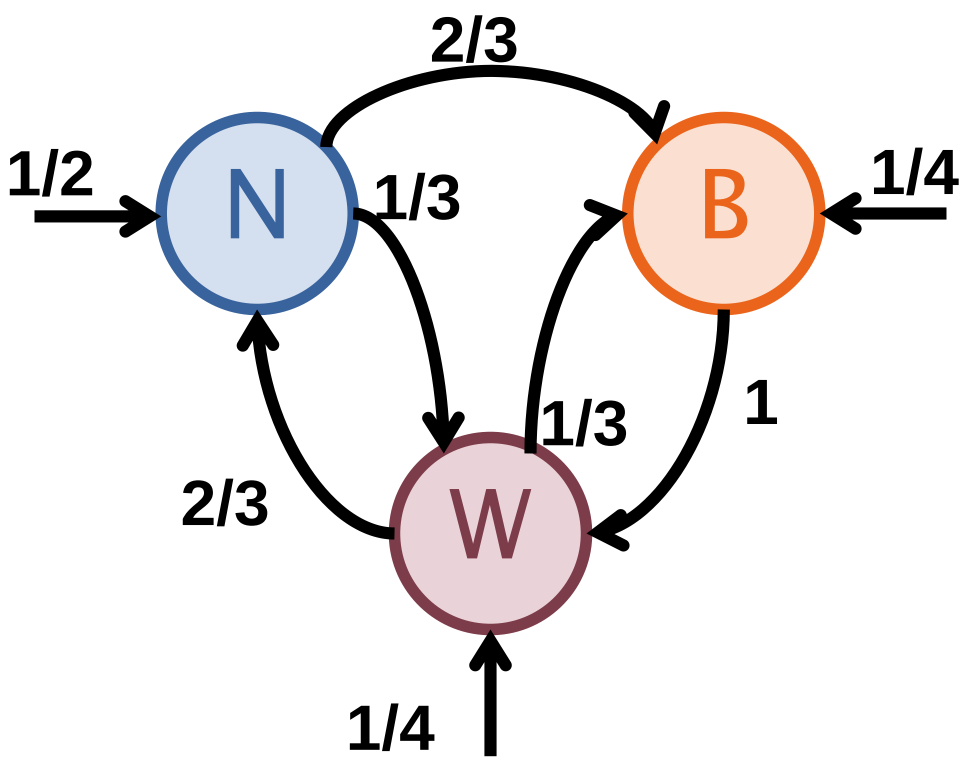
- Boston…

$$A = \begin{array}{c} \\ N \\ B \\ W \end{array} \begin{array}{c} \begin{array}{ccc} N & B & W \end{array} \\ \begin{bmatrix} 0 & 2/3 & 1/3 \\ 0 & 0 & 1 \\ 2/3 & 1/3 & 0 \end{bmatrix} \end{array}$$

- To Boston = 0

Markov Model

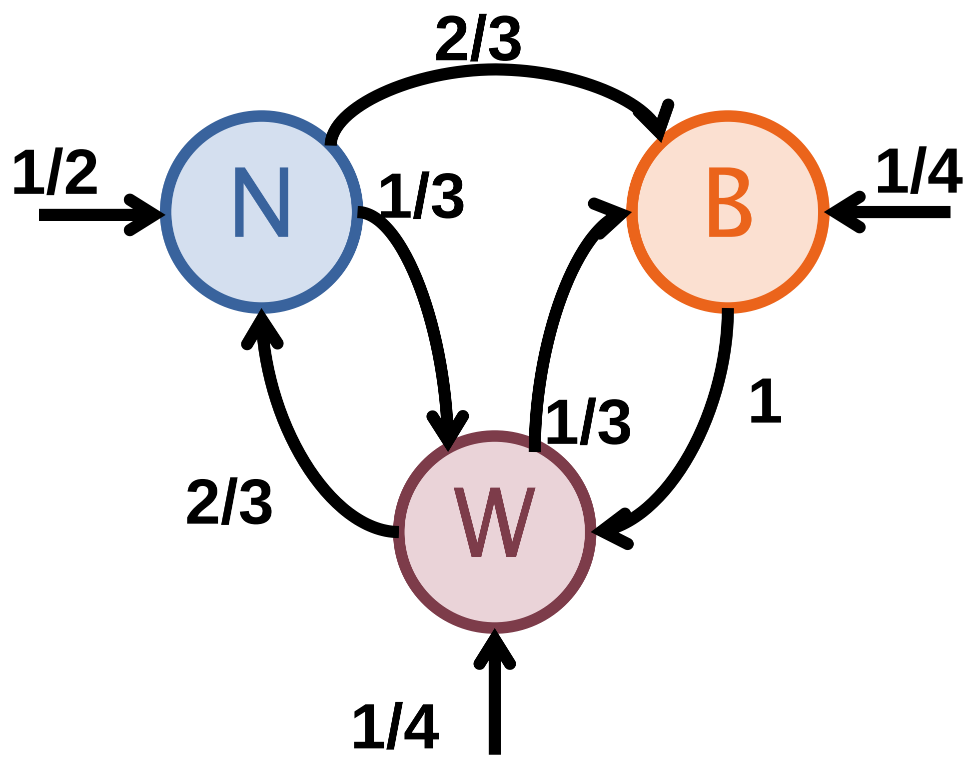
- The completed Markov Model…



- Now what?

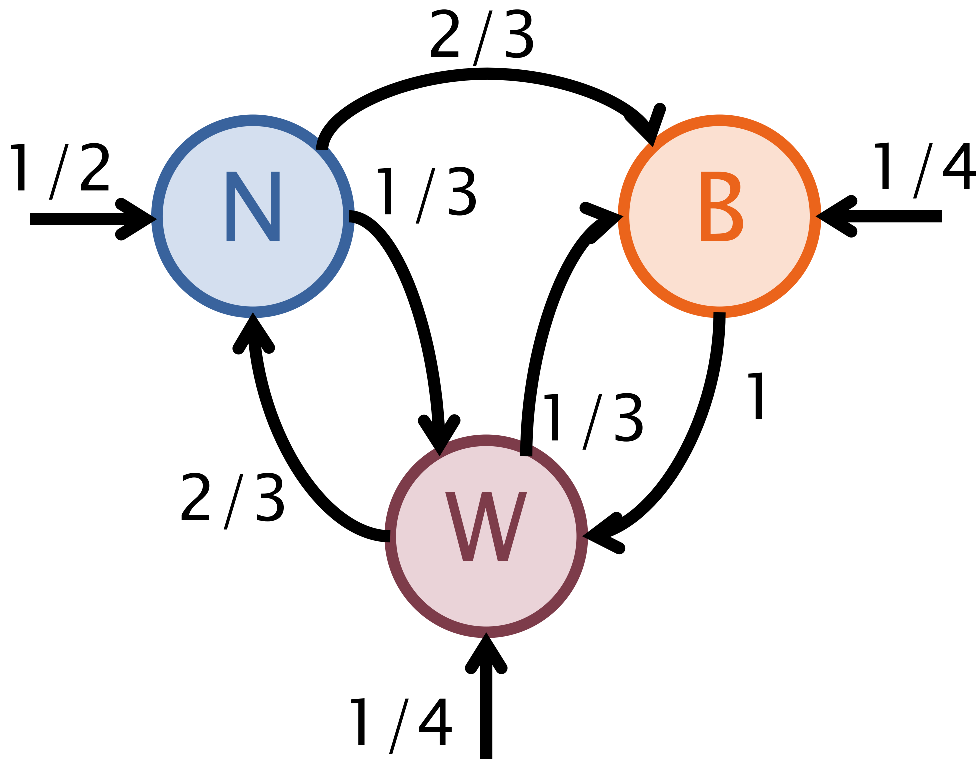
Markov Model

- If somebody is planning a trip…
- Recommend the most popular trip



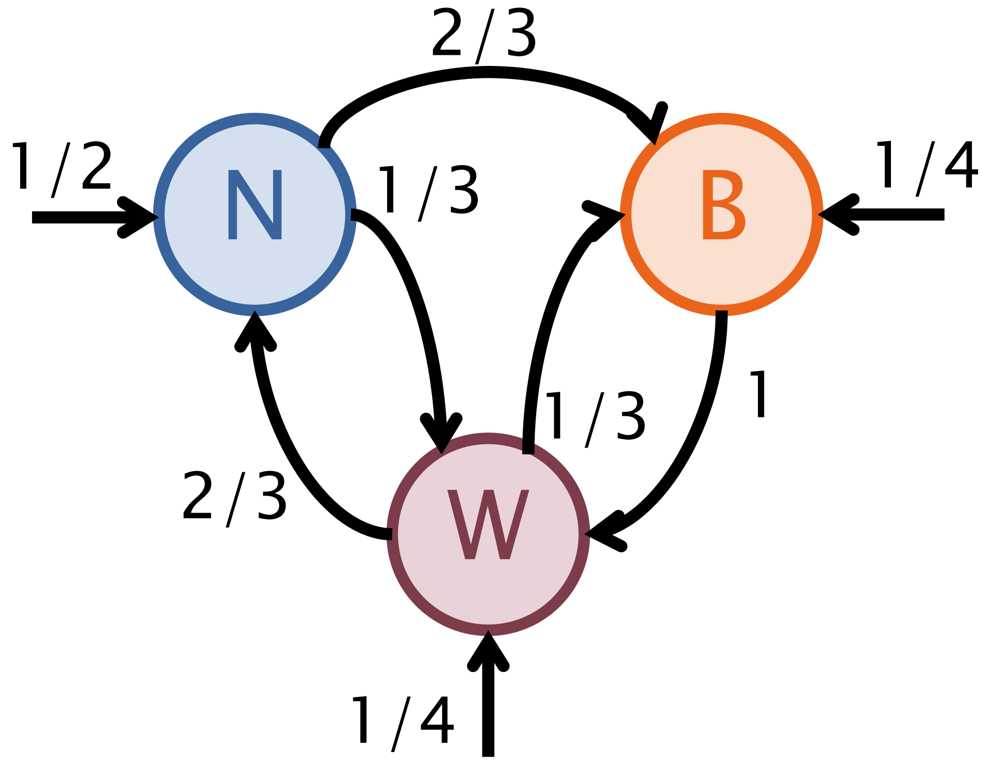
What Makes it Popular?

P([W→N→B]) = 1/4*2/3*2/3=1/9



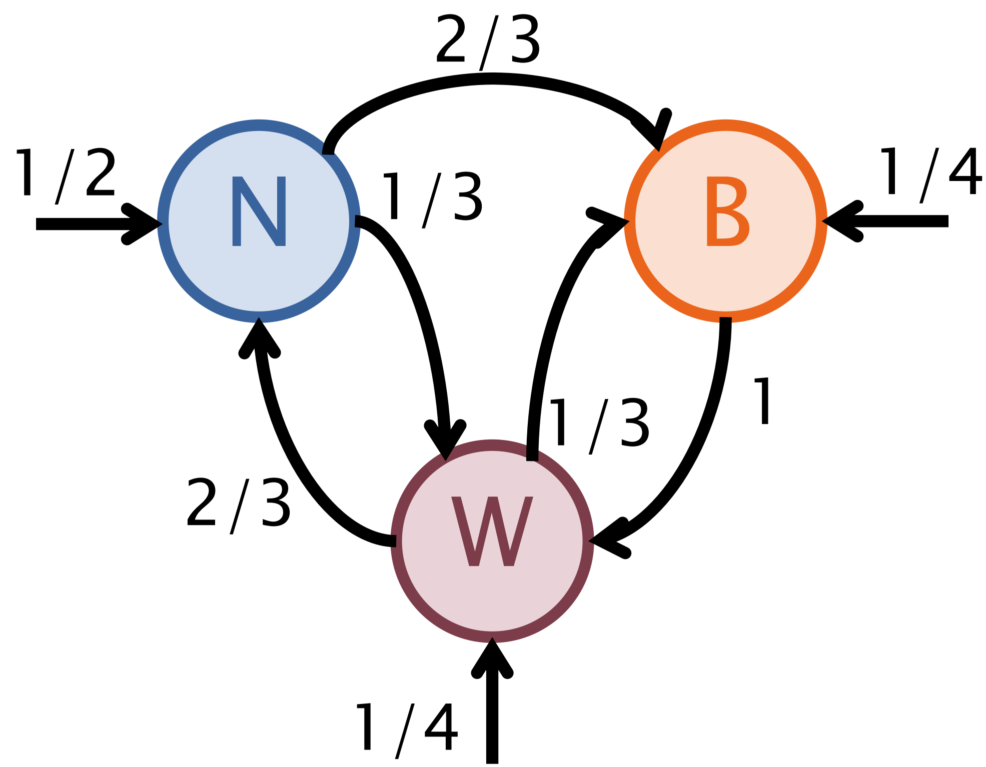
What Makes it Popular?

$$P([B \rightarrow W \rightarrow B]) = 1/4 * 1/3 * 1 = 1/12$$



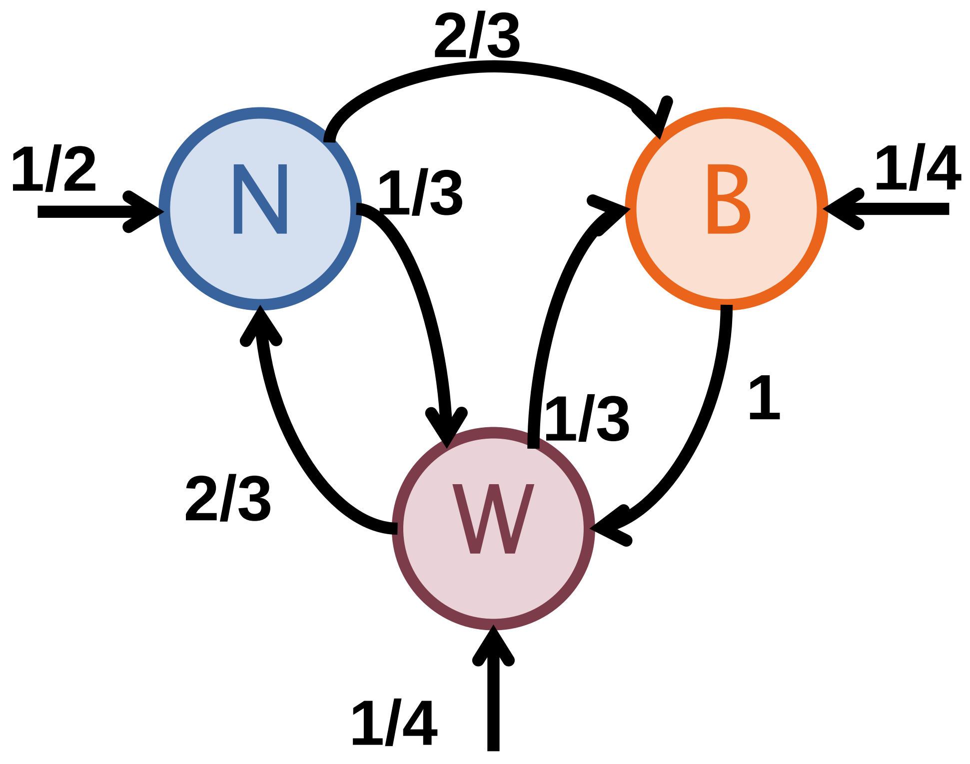
What Makes it Popular?

$$P([N \rightarrow B \rightarrow W]) = 1/2 * 2/3 * 1 = 1/3$$



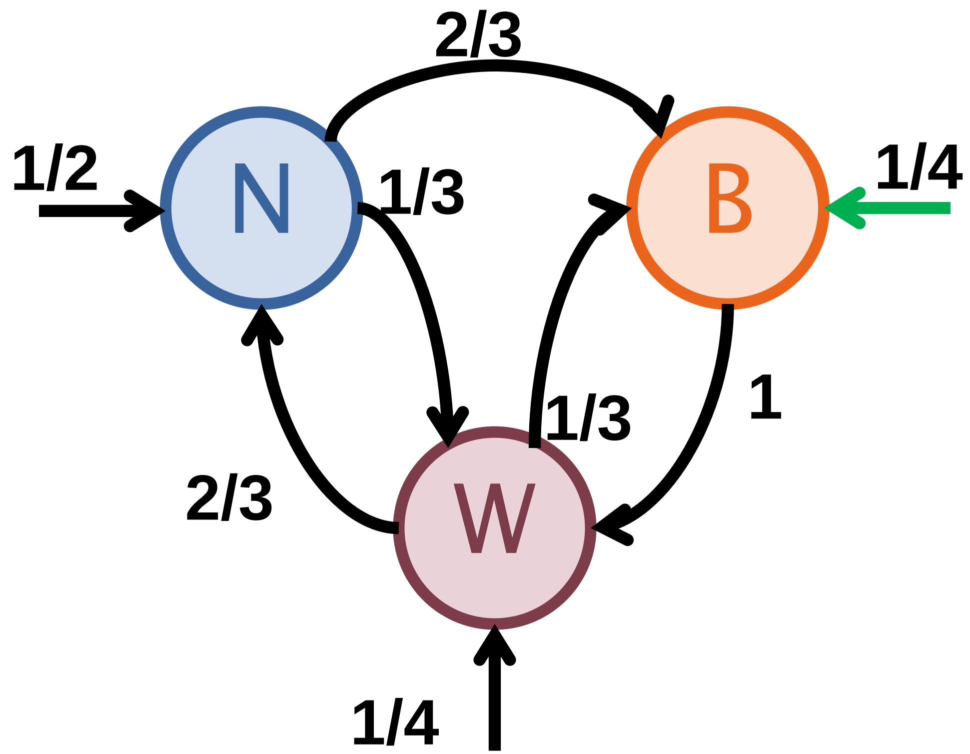
Markov Model

- If we know a person’s location…
- We can make recommendations



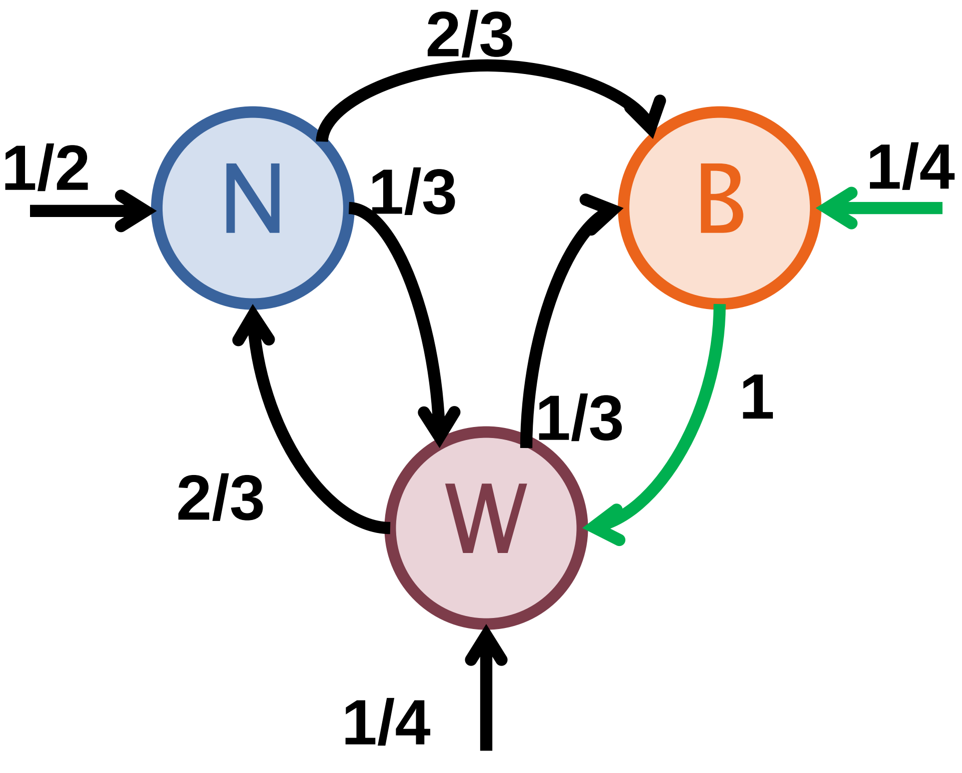
Markov Model

- If we know vacationers are in Boston…
- Recommend Washington DC



Markov Model

- From Washington DC…
- Recommend New York City



Python Implementation

```
class markovmodel:
    #transmat: None
    def __init__(self, transmat = None, startprob = None):
        self.transmat = transmat
        self.startprob = startprob
    # It assumes the state number starts from 0
    def fit(self, X):
        ns = max([max(items) for items in X]) + 1
        self.transmat  = np.zeros([ns, ns])
        self.startprob = np.zeros([ns])
        for items in X:
            n = len(items)
            self.startprob[items[0]] += 1
            for i in range(n-1):
                self.transmat[items[i], items[i+1]] += 1
        self.startprob = self.startprob / sum(self.startprob)
        n = self.transmat.shape[0]
        d = np.sum(self.transmat, axis=1)
        for i in range(n):
            if d[i] == 0:
                self.transmat[i,:] = 1.0 / n
        d[d == 0] = 1
        self.transmat = self.transmat * \
                        np.transpose(np.outer(np.ones([ns,1]), 1./d))
    def predict(self, obs, steps):
        pred = []
        n = len(obs)
        if len(obs) > 0:
            s = obs[-1]
        else:
            s = np.argmax(np.random.multinomial(1,
                          self.startprob.tolist(), size = 1))
        for i in range(steps):
            s1 = np.random.multinomial(1, self.transmat[s,:].tolist(),
                                       size = 1)
            pred.append(np.argmax(s1))
            s = np.argmax(s1)
        return pred
```

Python Implementation

```
label = {0: "New York City", 1: "Boston", 2: "Washington D.C."}
y = [[0, 1, 2],
     [2, 0, 1],
     [0, 2, 1],
     [1, 2, 0]]
```

```
mm = markovmodel()
mm.fit(y)
pred = mm.predict([2], 2)
print [label[s] for s in pred]
```

```
['New York City', 'Boston']
```

Markov Model

- Markov Models are cool but is there something cooler?
- Yes!
- Can be extended to Hidden Markov Models

Back on Vacation

Person A	$800	$400	$600
Person B	$550	$900	$450
Person C	$750	$650	$325
Person D	$425	$600	$750

- Now, the vacationers are on a spending spree
- Each person spends differently in each city
- Patterns emerge

Learning the Patterns

- How can we learn these patterns?
- Unfortunately we need to do some math…

Back on Vacation

$800

$900

$750

$750

- Mean

$$\mu = \frac{1}{k}\sum_{i=1}^{k} o_k = \frac{800 + 900 + 750 + 750}{4} = \$800$$

Back on Vacation

$800

$900

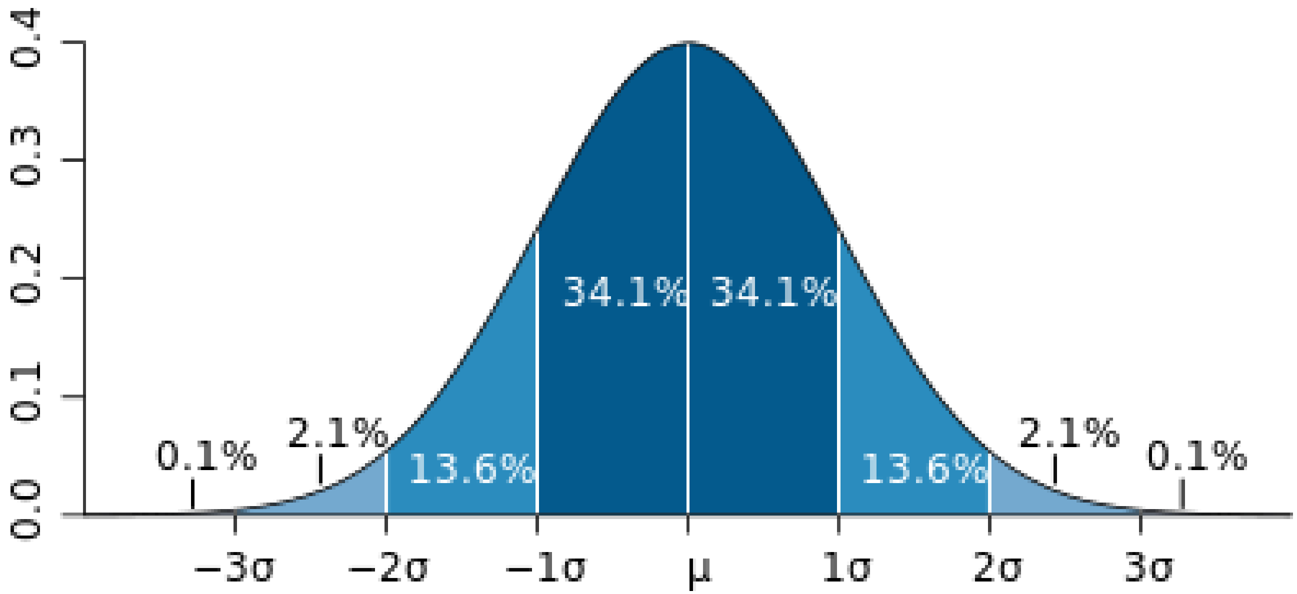
$750

$750

- Standard Deviation

$$\sigma = \sqrt{\frac{1}{k}\sum_{i=1}^{k}(o_k - \mu)^2} = \sqrt{\frac{0+100^2+50^2+50^2}{4}} = \mathbf{\$61}$$

Back on Vacation



- Normal Distribution

$$N(\mu, \sigma\ ) = N(\$800, \$61)$$

Back on Vacation

$400

$450

$325

$425

- Normal Distribution

$$N(\mu, \sigma) = N(\$400, \$47)$$

Back on Vacation

$600

$550

$650

$600

- Normal Distribution

$$N(\mu, \sigma) = N(\$600, \$35)$$

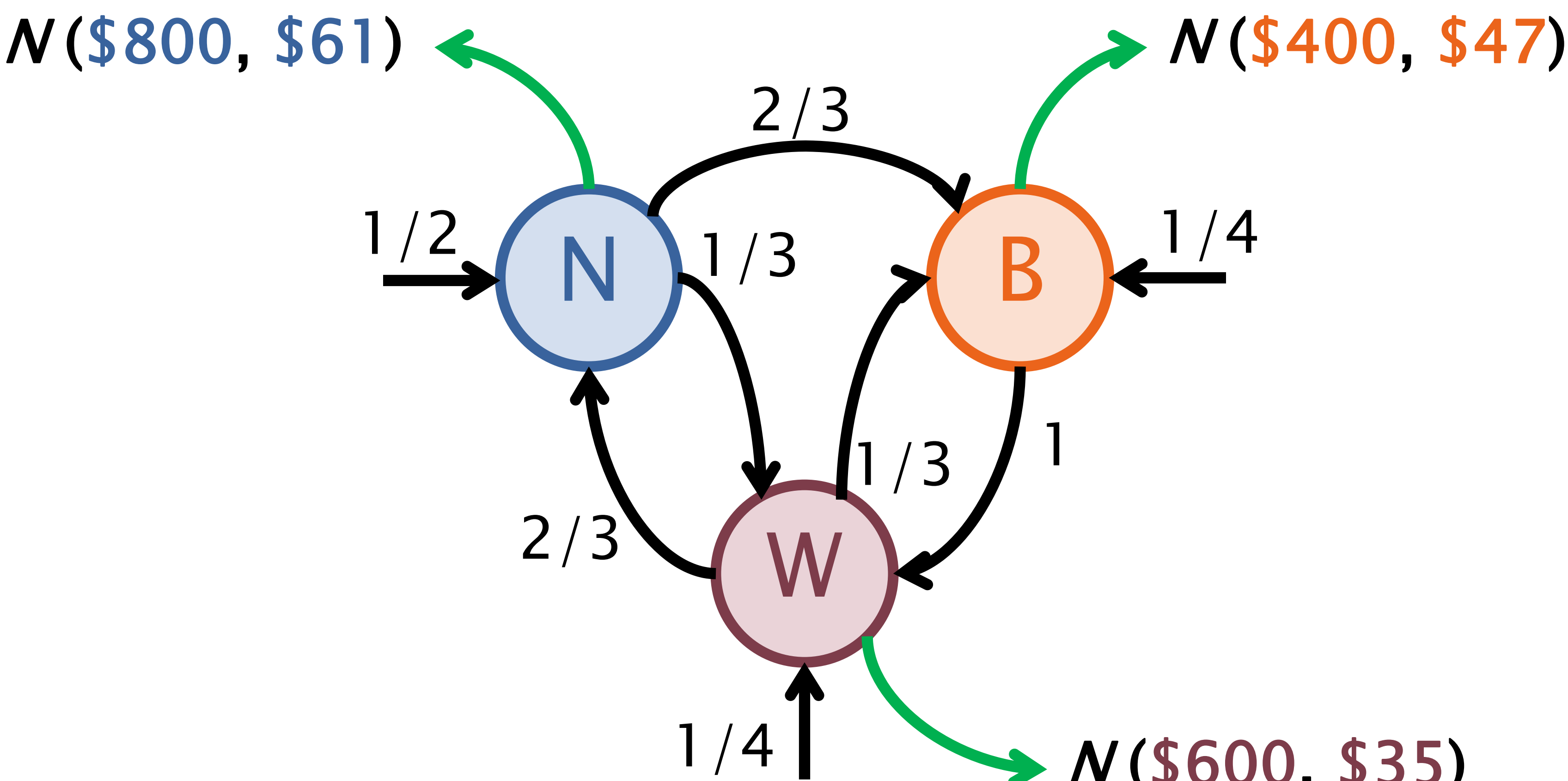
Back on Vacation

- Probable spending functions…

$$B=[N(\$800, \$61),\ N(\$400, \$47),\ N(\$600, \$35)]$$

- We just created a Hidden Markov Model!

Hidden Markov Model



- Now what?

Observations

- What if we don't know which city the vacationers are in?
- Instead, we only have observations of their spending

Observations

- We know only this spending pattern…

$875 → $450 → $550

- How can we determine the sequence of cities visited?

Observations

- Let's check the first observation…

$875

- What is the most probable city?

$$B = [\mathcal{N}(\$800, \$61), \\ \mathcal{N}(\$400, \$47), \\ \mathcal{N}(\$600, \$35)]$$

Observations

- Now the second observation…

$875 → $450

- What is the most probable city?

$$
\begin{aligned}
B=[&N(\$800, \$61),\\
&\boxed{N(\$400, \$47),}\\
&N(\$600, \$35)]
\end{aligned}
$$

Observations

- Now the third observation…

\$875 → \$450 →  \$550

- What is the most probable city?

$$B=[\mathcal{N}(\$800, \$61),$$
$$\mathcal{N}(\$400, \$47),$$
$$\boxed{\mathcal{N}(\$600, \$35)]}$$

Partial Observations

- What if we only have a single observation?

Partial Observations

- A single observation…

$625

Partial Observations

- A single observation…

\$625

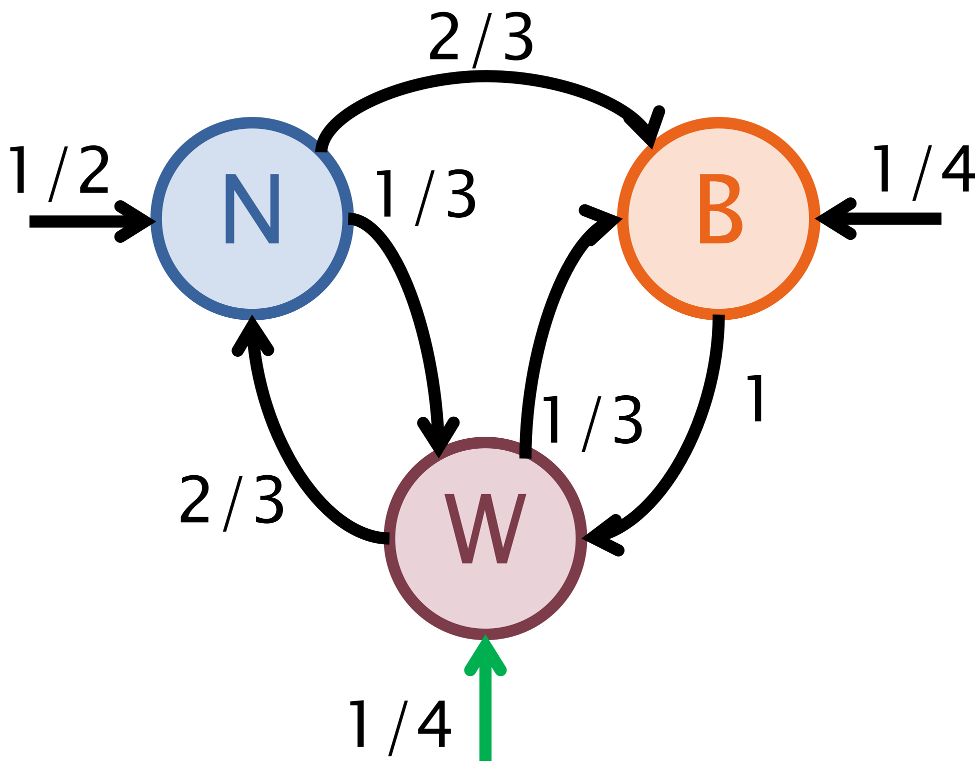
- As earlier, infer current city…

$$B = [\mathcal{N}(\$800, \$61), \mathcal{N}(\$400, \$47), \boxed{\mathcal{N}(\$600, \$35)}]$$

Partial Observations

- Now we can make recommendations

$625 → New York City → Boston



Partial Observations

- Let's try another…

$350

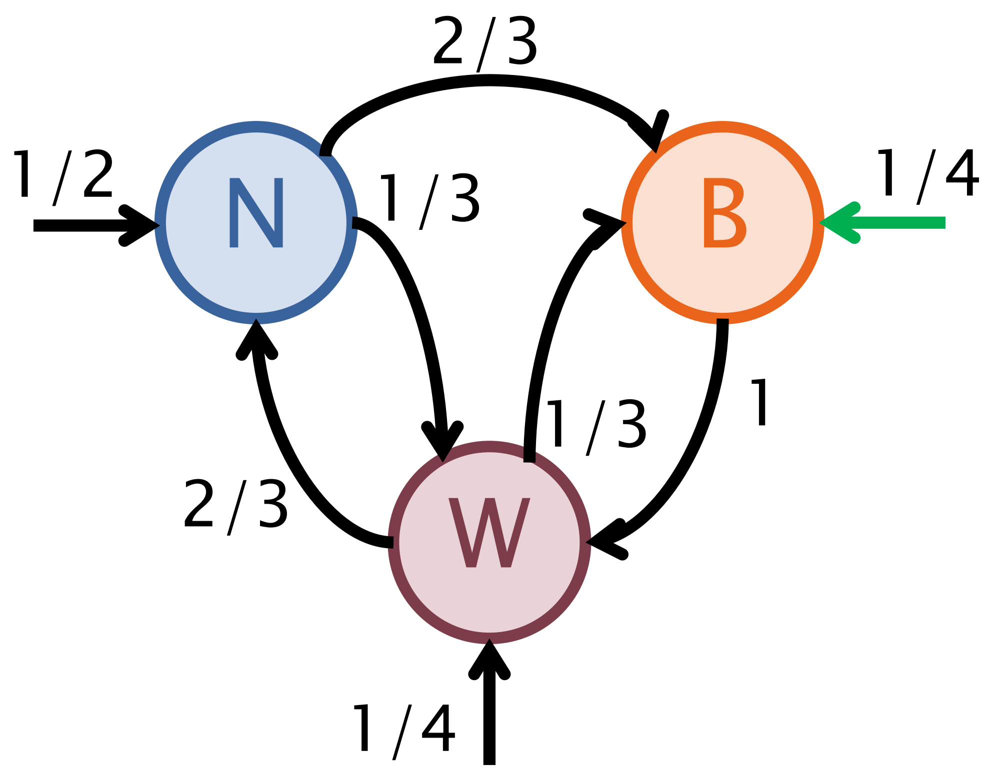
- Infer current city location

$$B=[\mathcal{N}(\$800, \$61), \boxed{\mathcal{N}(\$400, \$47),} \mathcal{N}(\$600, \$35)]$$

Partial Observations

- Make recommendations…

$350 → Washington DC → New York City



Python Implementation

```python
import numpy as np
from hmmlearn import hmm
```

```python
label = {0: "New York City", 1: "Boston", 2: "Washington D.C."}
X = [[[800],  [400], [600]],
     [[550],  [900], [450]],
     [[750],  [650], [325]],
     [[425], [600],  [750]]]
y = [[0,      1,     2],
     [2,      0,     1],
     [0,      2,     1],
     [1,      2,     0]]
```

Python Implementation

- Train a HMM model with Gaussian observations
- Predict the latent cities of training sequences of spendings

```
startprob, transmat, means, covars = estimate_parameters(X, y)
model = hmm.GaussianHMM(3, "full", startprob, transmat)
model.means_  = means
model.covars_ = covars
for x in X:
    y = model.predict(x)
    print [label[s] for s in y]
```

```
['New York City', 'Boston', 'Washington D.C.']
['Washington D.C.', 'New York City', 'Boston']
['New York City', 'Washington D.C.', 'Boston']
['Boston', 'Washington D.C.', 'New York City']
```

Python Implementation

- Given three testing sequences of spendings, apply the trained HMM model to predict the latent cities of two testing sequences

```
X = [[[450], [650]],
      [[850],  [500]]]
```

```
for x in X:
    y = model.predict(x)
    print [label[s] for s in y]
```

```
['Boston', 'Washington D.C.']
['New York City', 'Boston']
```

Python Implementation

- Forecast subsequent cities to be visited
- Forecast subsequent spendings

```
x = [[450], [650]]
y = hmm_predict_states(model, x, 3)
print [label[s] for s in y]
```

```
['Boston', 'Washington D.C.', 'New York City']
```

```
x = [[450], [650]]
cons = hmm_predict_features(model, x, 3)
print [round(con[0], 2) for con in cons]
```

```
[604.94, 594.29, 595.64]
```

Summary

- Markov Models are pretty cool
- Hidden Markov Models are cooler
- Both are used for predictions based on sequences of events
- The order the events occur in must be relevant to the data
- Is there something even cooler???

Cooler than Markov Models

- Ar