Hidden Markov Models

Sanjay Dorairaj (Adapted from HMM tutorials by James Kunz, Professor, UC Berkeley, NLP)

Dated: March 17th 2018

Overview

Hidden Markov Models (HMMs) are a class of probabilistic graphical model that allow us to predict a sequence of unknown (hidden) variables from a set of observed variables. A simple example of an HMM is predicting the weather (hidden variable) based on the type of clothes that someone wears (observed). An HMM can be viewed as a Bayes Net unrolled through time with observations made at a sequence of time steps being used to predict the best hidden sequence or set of states.

The below diagram from Wikipedia shows an HMM and its transitions. The scenario is that a room contains urns X1, X2 and X3, each of which contains a known mix of balls, each ball labeled y1, y2, y3 and y4. A sequence of four balls is randomly drawn. In this particular case, **the user observes a sequence of balls y1,y2,y3 and y4 and is attempting to discern the hidden state which is the right sequence of three urns that these four balls were pulled from.**

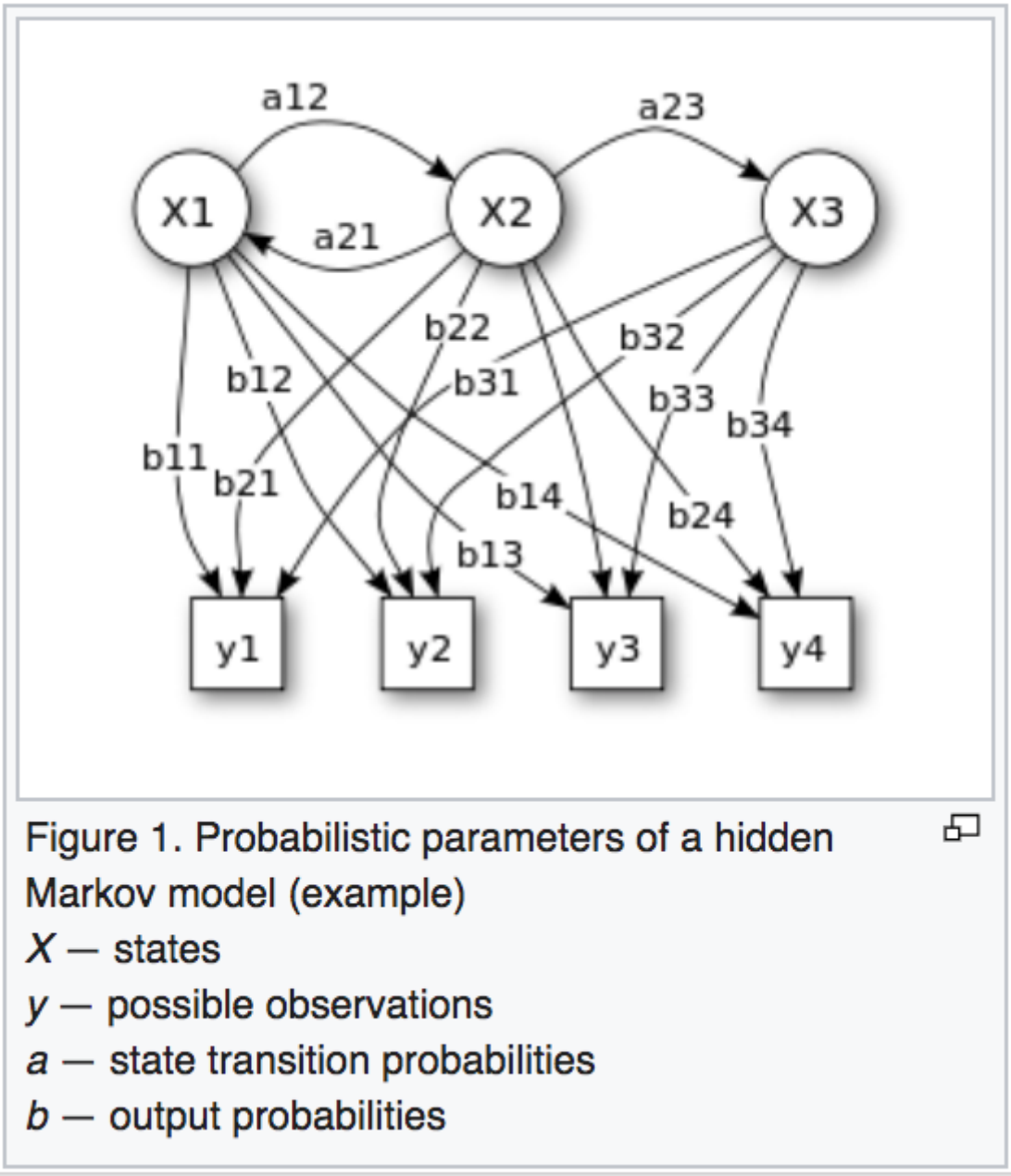

Figure 1. Probabilistic parameters of a hidden Markov model (example)
X — states
y — possible observations
a — state transition probabilities
b — output probabilities

Figure 1: HMM hidden and observed states

Source: https://en.wikipedia.org/wiki/Hidden_Markov_model#/media/File:HiddenMarkovModel.svg (https://en.wikipedia.org/wiki/Hidden_Markov_model#/media/File:HiddenMarkovModel.svg)

Why Hidden, Markov Model?

The reason this is known as a Hidden Markov Model is because we are constructing an inference model based on the assumptions of a Markov process. The Markov process assumption is simply that the "future is independent of the past given the present". In other words, assuming we know our present state, we do not need any other historical information to predict the future state.

To make this point clear, let us consider the scenario below where the weather, the hidden variable, can be hot, mild or cold and the observed variables are the type of clothing worn. The arrows represent transitions from a hidden state to another hidden state or from a hidden state to an

observed variable.

Notice that, true to the Markov assumption, each state only depends on the previous state and not on any other prior states.

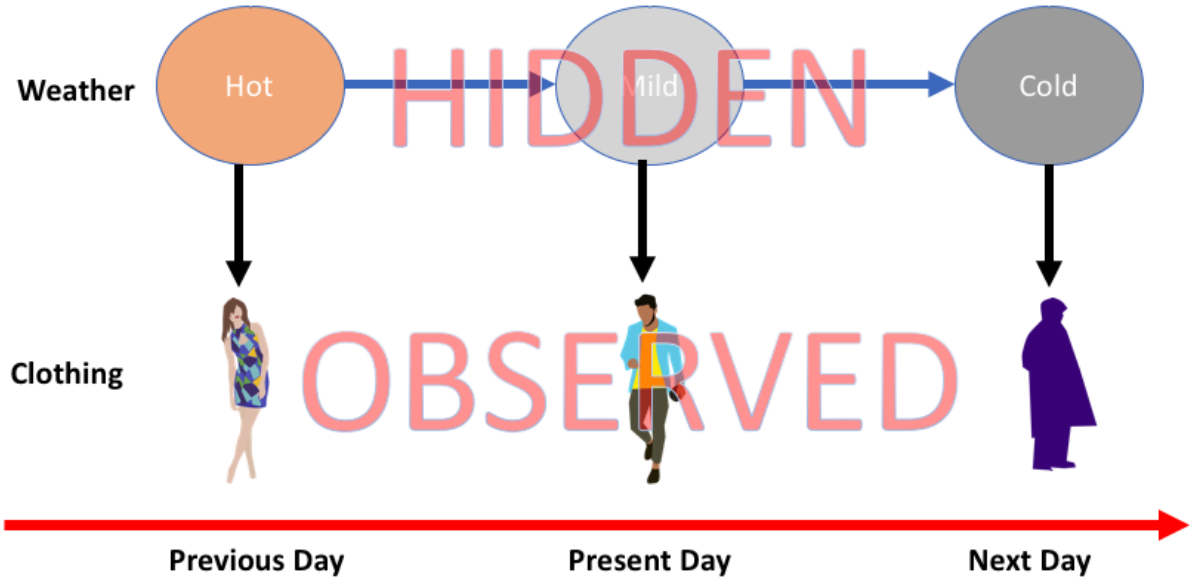

Figure 2: HMM State Transitions

Intuition behind HMMs

HMMs are probabilistic models. They allow us to compute the joint probability of a set of hidden states given a set of observed states. The hidden states are also referred to as latent states. Once we know the joint probability of a sequence of hidden states, we determine the best possible sequence i.e. the sequence with the highest probability and choose that sequence as the best sequence of hidden states.

The ratio of hidden states to observed states is not necessarily 1 is to 1 as is evidenced by Figure 1 above. The key idea is that one or more observations allow us to make an inference about a sequence of hidden states.

In order to compute the joint probability of a sequence of hidden states, we need to assemble three types of information.

Generally, the term states are used to refer to the hidden states and observations are used to refer to the observed states.

1. **Transition data** - the probability of transitioning to a new state conditioned on a present state.
2. **Emission data** - the probability of transition to an observed state conditioned on a hidden state.
3. **Initial state information** - the initial probability of transitioning to a hidden state. This can also be looked at as the prior probability.

The above information can be computed directly from our training data. For example, in the case of our weather example in Figure 2, our training data would consist of the hidden state and observations for a number of days. We could build our transition matrices of transitions, emissions and initial state probabilities directly from this training data.

The example tables show a set of possible values that could be derived for the weather/clothing scenario.

Priors

Hot	0.6
Mild	0.3
Cold	0.1

Transitions

	Hot	**Mild**	**Cold**
Hot	0.6	0.3	0.1
Mild	0.4	0.3	0.2
Cold	0.1	0.4	0.5

Emissions

	Hot	**Mild**	**Cold**
Casual Wear	0.8	0.19	0.01
Semi Casual Wear	0.5	0.4	0.1
Winter apparel	0.01	0.2	0.79

Figure 3: HMM State Transitions - Weather Example

Once this information is known, then the joint probability of sequence, by the conditional probability chain rule and by Markov assumption, can be shown to be proportional to be given by

The probability of observing a sequence

$$Y = y(0), y(1), \ldots, y(L-1)$$

of length *L* is given by

$$P(Y) = \sum_X P(Y \mid X) P(X),$$

where the sum runs over all possible hidden-node sequences

$$X = x(0), x(1), \ldots, x(L-1).$$

Figure 4: HMM - Basic Math

Note that as the number of observed states and hidden states gets large the compution gets more computationally intractable. If there are k possible values for each hidden sequence and we have a sequence length of n, there there are n^k total possible sequences that must be all scored and ranked in order to determine a winning candidate.

Variations of the Hidden Markov Model - HMM EM

Probability distributions of hidden states is not always known. In this case, we use Expectation Maximization (EM) models in order to determine hidden state distributions. A popular algorithm is the Baum-Welch algorithm (https://en.wikipedia.org/wiki/Baum%E2%80%93Welch_algorithm (https://en.wikipedia.org/wiki/Baum%E2%80%93Welch_algorithm))

Primer on Dynamic Programming and Summation Rules

Dynamic Programming

As seen the above sections on HMM, the computations become intractable as the sequence length and possible values of hidden states become large. It has been found that the problem of scoring an HMM sequence can be solved efficiently using dynamic programming, which is nothing but cached recursions.

Shown below is an image of the recursive computation of a fibonnaci series

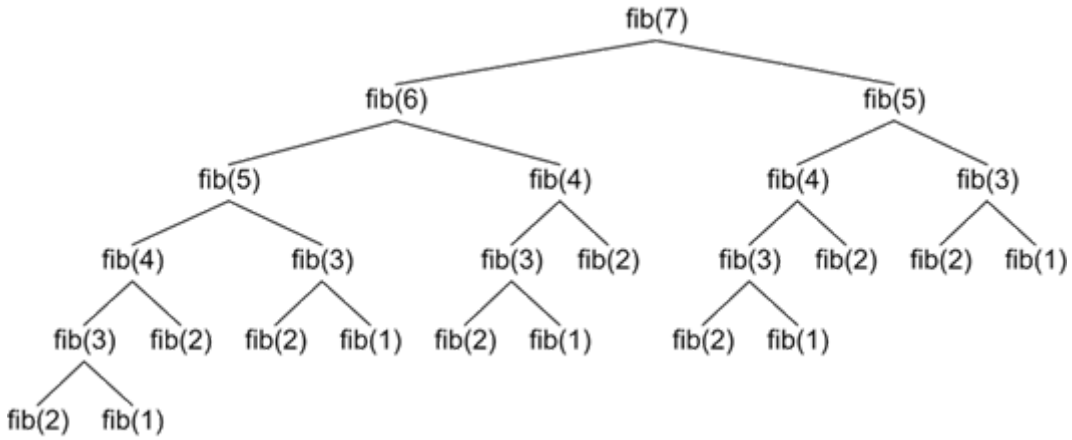

Figure 5: Fibonnacci Series - Tree

One of the things that becomes obvious when looking at this picture is that several results (fib(x) values) are reused in the computation. By caching these results, we can greatly speed up our operations

Fibonnacci Computation without Dynamic Programming

In [35]:
```
import time
```

In [36]:
```
# non-cached recursion
def fib(x):

    if x == 0:
        return 0
    elif x == 1:
        return 1
    else:
        return fib(x-1) + fib(x-2)

startTime = time.time()

print("%-14s:%d" % ("Result",fib(32)))
print("%-14s:%.4f seconds" % ("Elapsed time",time.time() - startTime))
```

```
Result        :2178309
Elapsed time  :2.0289 seconds
```

Fibonnacci Computation with Dynamic Programming

In [37]:
```
fib_cache = {}
def fib(x):

    if fib_cache.get(x) != None:
        return fib_cache[x]

    result = None
    if x == 0:
        result = 0
    elif x == 1:
        result = 1
    else:
        result = fib(x-1) + fib(x-2)

    if fib_cache.get(x) == None:
        fib_cache[x] = result

    return result

startTime = time.time()

print("%-14s:%d" % ("Result",fib(32)))
print("%-14s:%.4f seconds" % ("Elapsed time",time.time() - startTime))
```

```
Result        :2178309
Elapsed time  :0.0005 seconds
```

Notice the significant improvement in performance when we move to dynamic programming or cached recursion. We use this same idea when trying to score HMM sequences as well using an algorithm called the Forward-Backward algorithm which we will talk about later

Manipulating Summations

Here we look at an idea that will be leveraged in the forward backward algorithm. This is idea that double summations of terms can be rearrangeed as a product of each of the individual summation.

The example below explains this idea further.

$$1 \times 10 + 2 \times 10 + 3 \times 10 + 1 \times 20 + 2 \times 20 + 3 \times 20 + 1 \times 30 + 2 \times 30 + 3 \times 30$$

$$= \sum_{x=1,2,3} \sum_{y=10,20,30} x y$$

$$= \left(\sum_{x=1,2,3} x\right) \left(\sum_{y=10,20,30} y\right)$$

Figure 6: HMM - Manipulation Summations

The code below demonstrates this equivalency relationship

In [38]:
```
# double summations

x = [1,2,3]
y = [10,20,30]

total = 0
for i in x:
    for j in y:

        total = total + i*j

print total
```

```
360
```

In [39]:
```
# product of individual summations

total = 0
total1 = 0
for i in x:
    total1 = total1 + i

total2 = 0
for j in y:
    total2 = total2 + j

total = total1*total2

print total
```

```
360
```

Manipulating Maxes

Similar to manipulating double summations, the max of a double maxation can be viewed as the product of each of the individual maxations.

$$\max(1 \times 10, 2 \times 10, 3 \times 10, 1 \times 20, 2 \times 20, 3 \times 20, 1 \times 30, 2 \times 30, 3 \times 30)$$

$$= \max_{x=1,2,3} \max_{y=10,20,30} x\, y$$

$$= (\max_{x=1,2,3} x\, (\max_{y=10,20,30} y))$$

Figure - 7: HMM - Manipulation Maxations

In [49]:
```
# double maxation

x = [1,2,3]
y = [10,20,30]

# form 1
form1 = []
for i in x:
    for j in y:

        form1.append(i*j)

print "form1:",form1
print "max(form1):",max(form1)
```

```
form1: [10, 20, 30, 20, 40, 60, 30, 60, 90]
max(form1): 90
```

In [50]:
```
# product of individual maxations

form2_1 = []
form2_2 = []
for i in x:
    form2_1.append(i)
for j in y:
    form2_2.append(j)

print "\nform2_1",form2_1
print "form2_2",form2_2
print "max(form2_1) * max(form2_2)", max(form2_1) * max(form2_2)
```

```
form2_1 [1, 2, 3]
form2_2 [10, 20, 30]
max(form2_1) * max(form2_2) 90
```

Training an HMM

In this section, we will consider the toy example below and use the information from that example to train our simple HMM model

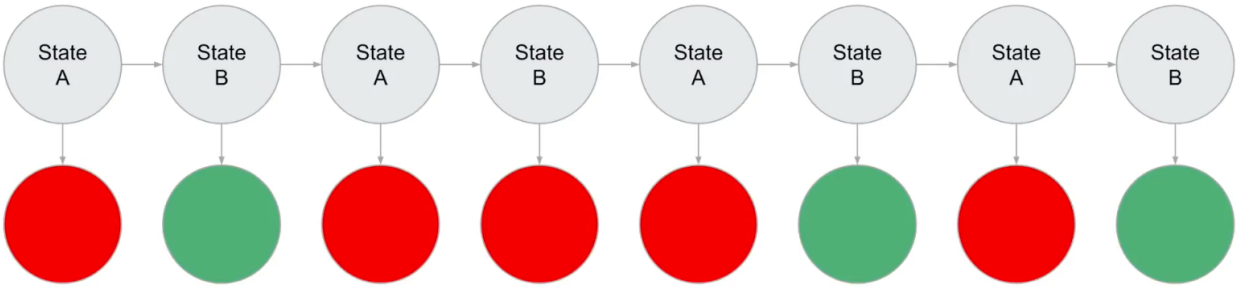

Figure - 8: HMM - Toy Example - Graph

State transition and emission probabilities

Priors

State A	P(A) = 1
State B	P(B) = 0

Transitions

	State A	**State B**
State A	P(A/A) = 0	P(B/A) = 1
State B	P(A/B) = 1	P(B/B) = 0

Emissions

	State A	**State B**
Red	P(A/Red) = 0	P(B/Red) = 1
Green	P(A/Green) = 1	P(B/Green) = 0

Figure - 9: HMM - Toy Example - Transition Tables

Scoring a known POS sequence given observed text

In this example, we score a known sequence given some text

Let us consider the below sequence

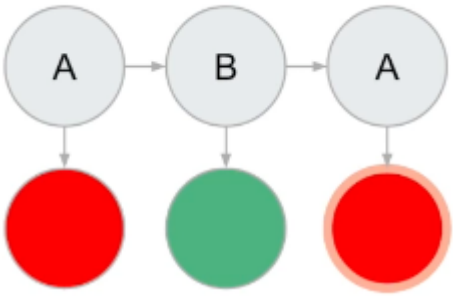

Figure - 10: HMM - Toy Example - Graph

The score for this sequence can be computed as

$$P(y_0, y_1, y_2, x_0, x_1, x_2) = P(y_0) \times P(x_0 \mid y_0) \times P(y_1 \mid y_0) \times P(x_1 \mid y_1) \times P(y_2 \mid y_1) \times P(x_2 \mid y_2)$$

Figure - 11: HMM - Toy Example - Scoring Known Sequence

The joint probability for our unknown sequence is therefore

$$P(A, B, A, Red, Green, Red) = [P(y_0 = A) * P(x_0 = Red/y_0 = A)] * [P(y_1 = B|y_0 = A|) * P(x_1 = \\ * [P(y_2 = A|y_1 = B) * P(x_2 = Red/y_2 = A)]$$

$$= (1 * 1) * (1 * 0.75) * (1 * 1)$$

$$= 0.75$$

Scoring a set of unknown sequences given some text

```
Assuming that we need to determine the parts of speech tags (hidden state)
given some sentence (the observed values), we will need to first score
every possible sequence of hidden states and then pick the best sequence
to determine the parts of speech for this sentence.

We will score this using the below steps

1. Generate the initial, transition and emission probability distribution
from the sample data.
2. Generate a list of all unknown sequence
3. Score all unknown sequences and select the best sequence
```

Generate list of unknown sequences

In [77]:
```
# given states - what are the possible combinations
# total number of combinations is (number of possible states)^(sequence leng

def generate_sequence(states,sequence_length):

    all_sequences = []

    depth = sequence_length

    def gen_seq_recur(states,nodes,depth):

        if depth == 0:
            #print nodes
            all_sequences.append(nodes)
        else:
            for state in states:
                temp_nodes = list(nodes)
                temp_nodes.append(state)
                gen_seq_recur(states,temp_nodes,depth-1)

    gen_seq_recur(states,[],depth)

    return all_sequences
```

Score all possible sequences

In [78]:
```
def score_sequences(sequences,initial_probs,transition_probs,emission_probs,

    best_score = -1
    best_sequence = None

    sequence_scores = []

    for seq in sequences:

        total_score = 1
        total_score_breakdown = []
        first = True
        for i in range(len(seq)):
            state_score = 1

            # compute transitition probability score
            if first == True:
                state_score *= initial_probs[seq[i]]

                # reset first flag
                first = False
            else:
                state_score *= transition_probs[seq[i] + "|" + seq[i-1]]

            # add to emission probability score
            state_score *= emission_probs[obs[i] + "|" + seq[i]]

            # update the total score
            #print state_score
            total_score_breakdown.append(state_score)
            total_score *= state_score

        sequence_scores.append(total_score)

    return sequence_scores
```

In [143]:

```
# pretty printing our  distributions
from sets import Set
import pandas as pd
from tabulate import tabulate

def pretty_print_probs(distribs):

    rows = Set()
    cols = Set()
    for val in distribs.keys():
        temp = val.split("|")
        rows.add(temp[0])
        cols.add(temp[1])

    rows = list(rows)
    cols = list(cols)

    df = []
    for i in range(len(rows)):
        temp = []
        for j in range(len(cols)):

            temp.append(distribs[rows[i]+"|"+cols[j]])

        df.append(temp)

    I = pd.Index(rows, name="rows")
    C = pd.Index(cols, name="cols")
    df = pd.DataFrame(data=df,index=I, columns=C)

    print tabulate(df, headers='keys', tablefmt='psql')

```

Compute the best sequence (Viterbi)

Note that selecting the best scoring sequence is also known as the **Viterbi** score. The alternative approach is the **Minimum Bayes Risk** approach which selects the highest scoring position across all sequence scores.

Example 1

Let us consider the below graph

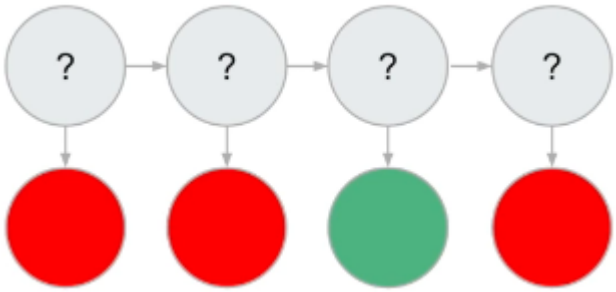

Figure - 12: HMM - Toy Example - Scoring an Unknown Sequence

In [144]:
```
# We can use a dictionary to capture our state transitions

# set of hidden states
states = ['A','B']

# set of observations
obs = ['Red','Green','Red']

# initial state probability distribution (our priors)
initial_probs = {'A':1.0,'B':0.0}

# transition probabilities
transition_probs = {'A|A':0,'A|B':1,'B|A':1,'B|B':0}

# emission probabilities
emission_probs = {'Red|A':1,'Green|A':0,'Red|B':0.25,'Green|B':0.75}

# length of sequence
sequence_length = 3

# Generate list of sequences
nodes = []
sequences = generate_sequence(states,sequence_length)

# Score sequences
sequence_scores = score_sequences(sequences,initial_probs,transition_probs,e

# print results

print("Initial Distributions")
print initial_probs

print("\nTransition Probabilities")
pretty_print_probs(transition_probs)

print("\nEmission Probabilities")
pretty_print_probs(emission_probs)

print("\nScores")

# Display sequence scores
for i in range(len(sequences)):
    print("Sequence:%10s,Score:%0.4f" % (sequences[i],sequence_scores[i]))
```

```
Initial Distributions
{'A': 1.0, 'B': 0.0}

Transition Probabilities
+--------+-----+-----+
| rows   |   A |   B |
|--------+-----+-----|
| A      |   0 |   1 |
| B      |   1 |   0 |
+--------+-----+-----+

Emission Probabilities
```

```
+--------+-----+------+
| rows   |   A |    B |
|--------+-----+------|
| Green  |   0 | 0.75 |
| Red    |   1 | 0.25 |
+--------+-----+------+

Scores
Sequence:['A', 'A', 'A'],Score:0.0000
Sequence:['A', 'A', 'B'],Score:0.0000
Sequence:['A', 'B', 'A'],Score:0.7500
Sequence:['A', 'B', 'B'],Score:0.0000
Sequence:['B', 'A', 'A'],Score:0.0000
Sequence:['B', 'A', 'B'],Score:0.0000
Sequence:['B', 'B', 'A'],Score:0.0000
Sequence:['B', 'B', 'B'],Score:0.0000
```

Example 2

Let us consider the below graph

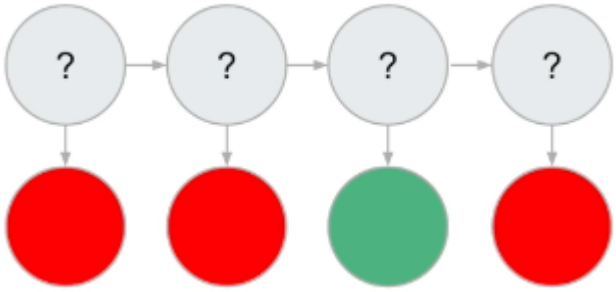

Figure - 12: HMM - Toy Example - Scoring an Unknown Sequence

In [147]:
```
# generate new sequences

states = ['Noun','Verb','Determiner']
sequence_length = 4
initial_probs = {'Noun':0.9,'Verb':0.05,'Determiner':0.05}
transition_probs = {'Noun|Noun':0.1,'Noun|Verb':0.1,'Noun|Determiner':0.8,
                    'Verb|Noun':0.8,'Verb|Verb':0.1,'Verb|Determiner':0.1,
                    'Determiner|Noun':0.1,'Determiner|Verb':0.8,'Determiner|
emission_probs = {'Bob|Noun':0.9,'ate|Noun':0.05,'the|Noun':0.05,'fruit|Noun
                  'Bob|Verb':0.05,'ate|Verb':0.9,'the|Verb':0.05,'fruit|Verb
                  'Bob|Determiner':0.05,'ate|Determiner':0.05,'the|Determine
obs = ['Bob','ate','the','fruit']

# print results

print("Initial Distributions")
print initial_probs

print("\nTransition Probabilities")
pretty_print_probs(transition_probs)

print("\nEmission Probabilities")
pretty_print_probs(emission_probs)

print("\nScores")

# Generate list of sequences
nodes = []
sequences = generate_sequence(states,sequence_length)

# Score sequences
sequence_scores = score_sequences(sequences,initial_probs,transition_probs,e

# Display sequence scores
for i in range(len(sequences)):
    print("Sequence:%-60s Score:%0.6f" % (sequences[i],sequence_scores[i]))

# Display the winning score
print("\n Best Sequence")
print(sequences[sequence_scores.index(max(sequence_scores))],max(sequence_sc
```

```
Initial Distributions
{'Verb': 0.05, 'Noun': 0.9, 'Determiner': 0.05}

Transition Probabilities
+------------+--------+--------+--------------+
| rows       |   Verb |   Noun |   Determiner |
|------------+--------+--------+--------------|
| Verb       |    0.1 |    0.8 |          0.1 |
| Noun       |    0.1 |    0.1 |          0.8 |
| Determiner |    0.8 |    0.1 |          0.1 |
+------------+--------+--------+--------------+

Emission Probabilities
+--------+--------+--------+--------------+
| rows   |   Verb |   Noun |   Determiner |
|--------+--------+--------+--------------|
```

```
| Bob    |   0.05 |   0.9  |          0.05 |
| fruit  |   0.05 |   0.9  |          0.05 |
| the    |   0.05 |   0.05 |          0.9  |
| ate    |   0.9  |   0.05 |          0.05 |
+--------+--------+--------+---------------+

Scores
Sequence:['Noun', 'Noun', 'Noun', 'Noun']                                Sco
re:0.000002
Sequence:['Noun', 'Noun', 'Noun', 'Verb']                                Sco
re:0.000001
Sequence:['Noun', 'Noun', 'Noun', 'Determiner']                          Sco
re:0.000000
Sequence:['Noun', 'Noun', 'Verb', 'Noun']                                Sco
re:0.000015
Sequence:['Noun', 'Noun', 'Verb', 'Verb']                                Sco
re:0.000001
Sequence:['Noun', 'Noun', 'Verb', 'Determiner']                          Sco
re:0.000006
Sequence:['Noun', 'Noun', 'Determiner', 'Noun']                          Sco
re:0.000262
Sequence:['Noun', 'Noun', 'Determiner', 'Verb']                          Sco
re:0.000002
Sequence:['Noun', 'Noun', 'Determiner', 'Determiner']                    Sco
re:0.000002
Sequence:['Noun', 'Verb', 'Noun', 'Noun']                                Sco
re:0.000262
Sequence:['Noun', 'Verb', 'Noun', 'Verb']                                Sco
re:0.000117
Sequence:['Noun', 'Verb', 'Noun', 'Determiner']                          Sco
re:0.000015
Sequence:['Noun', 'Verb', 'Verb', 'Noun']                                Sco
re:0.000262
Sequence:['Noun', 'Verb', 'Verb', 'Verb']                                Sco
re:0.000015
Sequence:['Noun', 'Verb', 'Verb', 'Determiner']                          Sco
re:0.000117
Sequence:['Noun', 'Verb', 'Determiner', 'Noun']                          Sco
re:0.302331
Sequence:['Noun', 'Verb', 'Determiner', 'Verb']                          Sco
re:0.002100
Sequence:['Noun', 'Verb', 'Determiner', 'Determiner']                    Sco
re:0.002100
Sequence:['Noun', 'Determiner', 'Noun', 'Noun']                          Sco
re:0.000015
Sequence:['Noun', 'Determiner', 'Noun', 'Verb']                          Sco
re:0.000006
Sequence:['Noun', 'Determiner', 'Noun', 'Determiner']                    Sco
re:0.000001
Sequence:['Noun', 'Determiner', 'Verb', 'Noun']                          Sco
re:0.000002
Sequence:['Noun', 'Determiner', 'Verb', 'Verb']                          Sco
re:0.000000
Sequence:['Noun', 'Determiner', 'Verb', 'Determiner']                    Sco
re:0.000001
Sequence:['Noun', 'Determiner', 'Determiner', 'Noun']                    Sco
re:0.000262
```

```
Sequence:['Noun', 'Determiner', 'Determiner', 'Verb']                Sco
re:0.000002
Sequence:['Noun', 'Determiner', 'Determiner', 'Determiner']          Sco
re:0.000002
Sequence:['Verb', 'Noun', 'Noun', 'Noun']                            Sco
re:0.000000
Sequence:['Verb', 'Noun', 'Noun', 'Verb']                            Sco
re:0.000000
Sequence:['Verb', 'Noun', 'Noun', 'Determiner']                      Sco
re:0.000000
Sequence:['Verb', 'Noun', 'Verb', 'Noun']                            Sco
re:0.000000
Sequence:['Verb', 'Noun', 'Verb', 'Verb']                            Sco
re:0.000000
Sequence:['Verb', 'Noun', 'Verb', 'Determiner']                      Sco
re:0.000000
Sequence:['Verb', 'Noun', 'Determiner', 'Noun']                      Sco
re:0.000001
Sequence:['Verb', 'Noun', 'Determiner', 'Verb']                      Sco
re:0.000000
Sequence:['Verb', 'Noun', 'Determiner', 'Determiner']                Sco
re:0.000000
Sequence:['Verb', 'Verb', 'Noun', 'Noun']                            Sco
re:0.000000
Sequence:['Verb', 'Verb', 'Noun', 'Verb']                            Sco
re:0.000000
Sequence:['Verb', 'Verb', 'Noun', 'Determiner']                      Sco
re:0.000000
Sequence:['Verb', 'Verb', 'Verb', 'Noun']                            Sco
re:0.000000
Sequence:['Verb', 'Verb', 'Verb', 'Verb']                            Sco
re:0.000000
Sequence:['Verb', 'Verb', 'Verb', 'Determiner']                      Sco
re:0.000000
Sequence:['Verb', 'Verb', 'Determiner', 'Noun']                      Sco
re:0.000117
Sequence:['Verb', 'Verb', 'Determiner', 'Verb']                      Sco
re:0.000001
Sequence:['Verb', 'Verb', 'Determiner', 'Determiner']                Sco
re:0.000001
Sequence:['Verb', 'Determiner', 'Noun', 'Noun']                      Sco
re:0.000000
Sequence:['Verb', 'Determiner', 'Noun', 'Verb']                      Sco
re:0.000000
Sequence:['Verb', 'Determiner', 'Noun', 'Determiner']                Sco
re:0.000000
Sequence:['Verb', 'Determiner', 'Verb', 'Noun']                      Sco
re:0.000000
Sequence:['Verb', 'Determiner', 'Verb', 'Verb']                      Sco
re:0.000000
Sequence:['Verb', 'Determiner', 'Verb', 'Determiner']                Sco
re:0.000000
Sequence:['Verb', 'Determiner', 'Determiner', 'Noun']                Sco
re:0.000006
Sequence:['Verb', 'Determiner', 'Determiner', 'Verb']                Sco
re:0.000000
Sequence:['Verb', 'Determiner', 'Determiner', 'Determiner']          Sco
```

re:0.000000
Sequence:['Determiner', 'Noun', 'Noun', 'Noun'] Sco
re:0.000000
Sequence:['Determiner', 'Noun', 'Noun', 'Verb'] Sco
re:0.000000
Sequence:['Determiner', 'Noun', 'Noun', 'Determiner'] Sco
re:0.000000
Sequence:['Determiner', 'Noun', 'Verb', 'Noun'] Sco
re:0.000000
Sequence:['Determiner', 'Noun', 'Verb', 'Verb'] Sco
re:0.000000
Sequence:['Determiner', 'Noun', 'Verb', 'Determiner'] Sco
re:0.000000
Sequence:['Determiner', 'Noun', 'Determiner', 'Noun'] Sco
re:0.000006
Sequence:['Determiner', 'Noun', 'Determiner', 'Verb'] Sco
re:0.000000
Sequence:['Determiner', 'Noun', 'Determiner', 'Determiner'] Sco
re:0.000000
Sequence:['Determiner', 'Verb', 'Noun', 'Noun'] Sco
re:0.000000
Sequence:['Determiner', 'Verb', 'Noun', 'Verb'] Sco
re:0.000000
Sequence:['Determiner', 'Verb', 'Noun', 'Determiner'] Sco
re:0.000000
Sequence:['Determiner', 'Verb', 'Verb', 'Noun'] Sco
re:0.000000
Sequence:['Determiner', 'Verb', 'Verb', 'Verb'] Sco
re:0.000000
Sequence:['Determiner', 'Verb', 'Verb', 'Determiner'] Sco
re:0.000000
Sequence:['Determiner', 'Verb', 'Determiner', 'Noun'] Sco
re:0.000117
Sequence:['Determiner', 'Verb', 'Determiner', 'Verb'] Sco
re:0.000001
Sequence:['Determiner', 'Verb', 'Determiner', 'Determiner'] Sco
re:0.000001
Sequence:['Determiner', 'Determiner', 'Noun', 'Noun'] Sco
re:0.000000
Sequence:['Determiner', 'Determiner', 'Noun', 'Verb'] Sco
re:0.000000
Sequence:['Determiner', 'Determiner', 'Noun', 'Determiner'] Sco
re:0.000000
Sequence:['Determiner', 'Determiner', 'Verb', 'Noun'] Sco
re:0.000000
Sequence:['Determiner', 'Determiner', 'Verb', 'Verb'] Sco
re:0.000000
Sequence:['Determiner', 'Determiner', 'Verb', 'Determiner'] Sco
re:0.000000
Sequence:['Determiner', 'Determiner', 'Determiner', 'Noun'] Sco
re:0.000001
Sequence:['Determiner', 'Determiner', 'Determiner', 'Verb'] Sco
re:0.000000
Sequence:['Determiner', 'Determiner', 'Determiner', 'Determiner'] Sco
re:0.000000

```
 Best Sequence
(['Noun', 'Verb', 'Determiner', 'Noun'], 0.30233088000000014)
```

Computing the Minimum Bayes Risk Score

In [149]:
```
# we need to first pick what it is we are looking for
# Let's say we are looking for a Noun in the second place i.e. y1
# We examine the set set of sequences and their scores but this
# time we group sequences by the possible values of y1 and compute
# the total scores within each group. The group with the highest
# score is the forward/backward score

forward_backward_index = 1 # 0-based

fb_scores = {}

# display sequence scores
for i in range(len(sequences)):

    if fb_scores.get(sequences[i][forward_backward_index]) == None:
        fb_scores[sequences[i][forward_backward_index]] = sequence_scores[i]
    else:
        fb_scores[sequences[i][forward_backward_index]] += sequence_scores[i

print "Forward Backward Scores:", fb_scores

best_fb_score = max(fb_scores.values())
best_y1 = [key for (key,value) in fb_scores.items() if value == best_fb_scor
print "Best Score:%s, Best y1:%s" % (best_fb_score,best_y1 )
```

```
Forward Backward Scores: {'Verb': 0.3075543675000003, 'Noun': 0.000298763
43750000016, 'Determiner': 0.00029876343750000016}
Best Score:0.3075543675, Best y1:['Verb']
```

In [ ]:
```
```