#LAB 11 : Hidden Markov Model

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

%matplotlib inline
```

Please refer to the following article (http://www.adeveloperdiary.com/data-science/machine-learning/introduction-to-hidden-markov-model/) to understand Hidden Markov Model

Here we will be dealing with 3 major problems :

1. Evaluation Problem
2. Learning Problem
3. Decoding Problem

1. Evaluation Problem : Implementation of Forward and Backward Algorithm

In [2]:

```python
data = pd.read_csv('data_python.csv') ## Read the data, change the path accordingly

V = data['Visible'].values

# Transition Probabilities
a = np.array(((0.54, 0.46), (0.49, 0.51)))

# Emission Probabilities
b = np.array(((0.16, 0.26, 0.58), (0.25, 0.28, 0.47)))

# Equal Probabilities for the initial distribution
initial_distribution = ((0.5, 0.5))


def forward(V, a, b, initial_distribution):
    alpha = np.zeros((V.shape[0], a.shape[0]))
    alpha[0, :] = initial_distribution * b[:, V[0]]

    for t in range(1, V.shape[0]):
        for j in range(a.shape[0]):
            alpha[t, j] = alpha[t - 1].dot(a[:, j]) * b[j, V[t]]

    return alpha


alpha = forward(V, a, b, initial_distribution)
print('alpha = ', alpha)
print('\n')

def backward(V, a, b):
    beta = np.zeros((V.shape[0], a.shape[0]))
    beta[V.shape[0] - 1] = np.ones((a.shape[0]))

    for t in range(V.shape[0] - 2, -1, -1):
        for j in range(a.shape[0]):
            beta[t, j] = (beta[t + 1] * b[:, V[t + 1]]).dot(a[j, :])

    return beta


beta = backward(V, a, b)
print('beta = ', beta)
```

```
alpha =  [[8.00000000e-002 1.25000000e-001]
 [2.71570000e-002 2.81540000e-002]
 [1.65069392e-002 1.26198572e-002]
 [8.75653677e-003 6.59378003e-003]
 [4.61649960e-003 3.47369232e-003]
 [2.43311103e-003 1.83073126e-003]
 [1.28234420e-003 9.64864889e-004]
 [6.75844805e-004 5.08520930e-004]
 [3.56196241e-004 2.68010114e-004]
 [1.87729137e-004 1.41251652e-004]
 [9.89404851e-005 7.44450603e-005]
 [5.21454461e-005 3.92354139e-005]
 [2.74826583e-005 2.06785741e-005]
 [1.44844194e-005 1.08984050e-005]
 [7.63384683e-006 5.74387913e-006]
 [4.02333128e-006 3.02724551e-006]
 [9.50546790e-007 9.50495728e-007]
 [5.67842140e-007 4.33342042e-007]
 [3.01003967e-007 2.26639558e-007]
```

2. Learning Problem : Implementation of Baum Welch Algorithm

In [3]:
```python
def baum_welch(V, a, b, initial_distribution, n_iter=100):

    M = a.shape[0]
    T = len(V)

    for n in range(n_iter):
        alpha = forward(V, a, b, initial_distribution)
        beta = backward(V, a, b)

        xi = np.zeros((M, M, T - 1))
        for t in range(T - 1):
            denominator = np.dot(np.dot(alpha[t, :].T, a) * b[:, V[t + 1]].T, beta[t + 1, :])
            for i in range(M):
                numerator = alpha[t, i] * a[i, :] * b[:, V[t + 1]].T * beta[t + 1, :].T
                xi[i, :, t] = numerator / denominator

        gamma = np.sum(xi, axis=1)
        a = np.sum(xi, 2) / np.sum(gamma, axis=1).reshape((-1, 1))

        # Add additional T'th element in gamma
        gamma = np.hstack((gamma, np.sum(xi[:, :, T - 2], axis=0).reshape((-1, 1))))

        K = b.shape[1]
        denominator = np.sum(gamma, axis=1)
        for l in range(K):
            b[:, l] = np.sum(gamma[:, V == l], axis=1)

        b = np.divide(b, denominator.reshape((-1, 1)))

    return a,b


data = pd.read_csv('data_python.csv')

V = data['Visible'].values

# Transition Probabilities
a = np.ones((2, 2))
a = a / np.sum(a, axis=1)

# Emission Probabilities
b = np.array(((1, 3, 5), (2, 4, 6)))
b = b / np.sum(b, axis=1).reshape((-1, 1))

# Equal Probabilities for the initial distribution
initial_distribution = np.array((0.5, 0.5))

a,b = baum_welch(V, a, b, initial_distribution, n_iter=100)
print(a)
print('\n')
print(b)
```

```
[[0.53816345 0.46183655]
 [0.48664443 0.51335557]]


[[0.16277513 0.26258073 0.57464414]
 [0.2514996  0.27780971 0.47069069]]
```

3. Decoding Problem : Implementation of Viterbi Algorithm

In [4]:
```
def viterbi(V, a, b, initial_distribution):

    T = V.shape[0]
    M = a.shape[0]

    omega = np.zeros((T, M))
    omega[0, :] = np.log(initial_distribution * b[:, V[0]])

    prev = np.zeros((T - 1, M))

    for t in range(1, T):
        for j in range(M):
            # Same as Forward Probability
            probability = omega[t - 1] + np.log(a[:, j]) + np.log(b[j, V[t]])

            # This is our most probable state given previous state at time t (1)
            prev[t - 1, j] = np.argmax(probability)

            # This is the probability of the most probable state (2)
            omega[t, j] = np.max(probability)

    # Path Array
    S = np.zeros(T)

    # Find the most probable last hidden state
    last_state = np.argmax(omega[T - 1, :])

    S[0] = last_state

    backtrack_index = 1
    for i in range(T - 2, -1, -1):
        S[backtrack_index] = prev[i, int(last_state)]
        last_state = prev[i, int(last_state)]
        backtrack_index += 1

    # Flip the path array since we were backtracking
    S = np.flip(S, axis=0)

    # Convert numeric values to actual hidden states
    result = []
    for s in S:
        if s == 0:
            result.append("A")
        else:
            result.append("B")

    return result


data = pd.read_csv('data_python.csv')

V = data['Visible'].values

# Transition Probabilities
a = np.ones((2, 2))
a = a / np.sum(a, axis=1)

# Emission Probabilities
b = np.array(((1, 3, 5), (2, 4, 6)))
b = b / np.sum(b, axis=1).reshape((-1, 1))

# Equal Probabilities for the initial distribution
initial_distribution = np.array((0.5, 0.5))

a, b = baum_welch(V, a, b, initial_distribution, n_iter=100)

result = viterbi(V, a, b, initial_distribution)

print(a)
print('\n')
print(b)
print('\n')
print(result)
```

```
[[0.53816345 0.46183655]
 [0.48664443 0.51335557]]


[[0.16277513 0.26258073 0.57464414]
 [0.2514996  0.27780971 0.47069069]]


['B', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'B', 'B', 'B', 'B', 'A', 'A', 'A', 'B', 'A', 'B', 'B', 'A', 'A', 'A', 'B', 'A', 'B', 'A', 'A',
'B', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'B', 'A', 'A', 'A', 'A', 'B', 'A', 'A', 'A', 'A', 'A', 'B',
'B', 'B', 'B', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'A', 'B', 'B', 'B', 'B', 'A', 'B', 'B', 'A', 'B',
'B', 'B', 'A', 'B', 'B', 'B', 'A', 'B', 'B', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'B', 'B', 'B', 'B', 'B', 'B', 'A', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'A', 'B', 'A', 'A', 'B', 'B',
'B', 'B', 'B', 'A', 'A', 'B', 'A', 'B', 'B', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'B', 'A', 'B', 'B', 'A', 'B', 'B', 'B', 'B', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'B', 'B', 'B', 'B', 'A', 'B', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'A', 'A', 'A',
'B', 'B', 'B', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'A', 'B', 'B', 'B',
'A', 'B', 'B', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'B', 'A', 'A', 'A', 'B', 'B', 'A', 'B', 'A', 'A',
'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'A', 'A', 'B', 'B', 'A', 'A', 'A', 'B', 'B', 'B', 'A', 'A',
'B', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'A', 'A', 'B', 'A', 'B', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'A', 'B', 'B',
'B', 'B', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'A', 'B', 'B', 'A', 'A', 'A', 'A', 'A', 'A',
'B', 'B', 'B', 'B', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'B', 'B', 'A', 'B', 'B', 'A', 'A', 'A', 'B', 'A', 'B', 'B', 'A', 'A', 'B', 'A', 'A', 'B', 'A', 'A',
'A', 'A', 'B', 'A', 'A', 'A', 'A', 'A', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'B',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'B', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'A', 'B', 'B', 'A', 'A', 'A', 'B', 'B',
'B', 'B', 'A', 'A', 'A', 'A', 'A', 'B', 'A', 'A', 'A', 'A', 'B', 'B', 'A', 'B', 'B', 'B', 'B', 'B',
'B', 'B', 'A', 'B', 'B', 'B', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A']
```

In [5]:

```
#compute accuracy
y_true = data['Hidden'].values

acc = 0
m = len(y_true)

for i in range(m):
    if(y_true[i] == result[i]):
        acc += 1

acc = acc/m
print(acc*100)
```

```
31.0
```

4. Use the built-in **hmmlearn** package to fit the data and generate the result using the decoder

In [6]:

```
!pip install hmmlearn
```

```
Requirement already satisfied: hmmlearn in c:\users\pranay kamal\anaconda3\lib\site-packages (0.2.8)
Requirement already satisfied: scipy>=0.19 in c:\users\pranay kamal\anaconda3\lib\site-packages (fro
m hmmlearn) (1.6.2)
Requirement already satisfied: scikit-learn>=0.16 in c:\users\pranay kamal\anaconda3\lib\site-packag
es (from hmmlearn) (0.24.1)
Requirement already satisfied: numpy>=1.10 in c:\users\pranay kamal\anaconda3\lib\site-packages (fro
m hmmlearn) (1.21.4)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\pranay kamal\anaconda3\lib\site-pack
ages (from scikit-learn>=0.16->hmmlearn) (2.1.0)
Requirement already satisfied: joblib>=0.11 in c:\users\pranay kamal\anaconda3\lib\site-packages (fr
om scikit-learn>=0.16->hmmlearn) (1.0.1)
```

In [7]:
```python
from hmmlearn import hmm
import numpy as np

data = pd.read_csv('data_python.csv')
V = data['Visible'].values

print(data.head())
```

```
  Hidden  Visible
0      B        0
1      B        1
2      B        2
3      B        2
4      B        2
```

In [8]:
```python
model = hmm.CategoricalHMM(n_components=2)
model.startprob_ = np.array([0.5, 0.5])
model.transmat_ = np.array([[0.5, 0.5],
                            [0.5, 0.5]])
model.emissionprob_ = np.array([[0.11111111, 0.33333333, 0.55555556],
                                [0.16666667, 0.33333333 ,0.5]])
```

In [9]:
```python
import math
logprob, sequence = model.decode((np.array(V).reshape(-1,1)).transpose())
out = []
for i in sequence:
    if i == 1:
        i = "B"
    else:
        i = "A"
    out.append(i)

print(out)
```

```
['B', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'A', 'A', 'A',
'A', 'B', 'B', 'B', 'B', 'A', 'B', 'A', 'B', 'A', 'B', 'B', 'A', 'B', 'A', 'B', 'A', 'B', 'A', 'A',
'B', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'B', 'A', 'A', 'A', 'A', 'B', 'A', 'A', 'A', 'B', 'A', 'B',
'B', 'B', 'B', 'A', 'B', 'A', 'B', 'B', 'B', 'B', 'A', 'B', 'B', 'B', 'B', 'A', 'B', 'B', 'A', 'B',
'B', 'B', 'A', 'B', 'B', 'B', 'A', 'B', 'B', 'B', 'A', 'B', 'A', 'B', 'B', 'A', 'B', 'A', 'A', 'A',
'B', 'A', 'B', 'A', 'B', 'B', 'B', 'A', 'A', 'B', 'A', 'A', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'B', 'B', 'B', 'B', 'B', 'B', 'A', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'A', 'B', 'A', 'A', 'B', 'B',
'B', 'B', 'B', 'A', 'A', 'B', 'A', 'B', 'B', 'B', 'A', 'A', 'A', 'A', 'A', 'B', 'A', 'A', 'A', 'A',
'A', 'B', 'A', 'B', 'A', 'B', 'A', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'A', 'B', 'A',
'B', 'B', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'A', 'A', 'A', 'A', 'A', 'B', 'A', 'B', 'A',
'B', 'A', 'B', 'A', 'B', 'B', 'A', 'B', 'B', 'B', 'B', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'B', 'B', 'B', 'B', 'A', 'B', 'B', 'A', 'A', 'B', 'A', 'A', 'A', 'B', 'B', 'B', 'A', 'A', 'A',
'B', 'B', 'B', 'B', 'A', 'A', 'B', 'A', 'A', 'A', 'B', 'A', 'A', 'A', 'A', 'B', 'A', 'B', 'B', 'B',
'A', 'B', 'B', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'B', 'A', 'B', 'A', 'B', 'B', 'A', 'B', 'A', 'A',
'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'A', 'A', 'B', 'B', 'A', 'A', 'A', 'B', 'B', 'B', 'A', 'A',
'B', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'A', 'A', 'B', 'A', 'B', 'A', 'A', 'A', 'B', 'A',
'A', 'A', 'B', 'B', 'B', 'B', 'A', 'B', 'B', 'B', 'A', 'A', 'B', 'B', 'A', 'B', 'B', 'A', 'B', 'B',
'B', 'B', 'B', 'A', 'B', 'A', 'B', 'A', 'A', 'A', 'B', 'A', 'B', 'B', 'A', 'A', 'A', 'A', 'A', 'A',
'B', 'B', 'B', 'B', 'B', 'A', 'B', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'A', 'A', 'A', 'A', 'A', 'A',
'B', 'B', 'A', 'B', 'B', 'A', 'B', 'A', 'B', 'A', 'B', 'B', 'A', 'A', 'B', 'A', 'A', 'B', 'A', 'A',
'A', 'A', 'B', 'A', 'A', 'A', 'B', 'A', 'B', 'A', 'B', 'A', 'A', 'A', 'B', 'A', 'A', 'A', 'B', 'B',
'A', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'A', 'A', 'A', 'B', 'A', 'A', 'B', 'A', 'A', 'A',
'A', 'B', 'A', 'B', 'A', 'B', 'A', 'A', 'B', 'B', 'B', 'B', 'A', 'B', 'B', 'A', 'A', 'A', 'B', 'B',
'B', 'B', 'A', 'B', 'A', 'A', 'A', 'B', 'A', 'B', 'A', 'A', 'B', 'B', 'A', 'B', 'B', 'B', 'B', 'B',
'B', 'B', 'A', 'B', 'B', 'B', 'A', 'B', 'A', 'A', 'B', 'B', 'B', 'A', 'B', 'B', 'A', 'A', 'B', 'A']
```

```
In [10]: #compute accuracy
y_true = data['Hidden'].values

acc = 0
m = len(y_true)

for i in range(m):
    if(y_true[i] == out[i]):
        acc += 1

acc = acc/m
print(acc*100)
```

```
32.2
```

```
In [11]: model = hmm.CategoricalHMM(n_components=2)
model.startprob_ = np.array([0.5, 0.5])
model.transmat_ = np.array([[0.5, 0.5],
                            [0.5, 0.5]])
model.emissionprob_ = np.array([[0.11111111, 0.33333333, 0.55555556],
                                [0.16666667, 0.33333333 ,0.5]])

model.fit((np.array(V).reshape(-1,1)))
logprob, sequence = model.decode((np.array(V).reshape(-1,1)).transpose())
out_new = []
for i in sequence:
    if i == 1:
        i = "B"
    else:
        i = "A"
    out_new.append(i)

print(out_new)
```

```
Even though the 'startprob_' attribute is set, it will be overwritten during initialization because
'init_params' contains 's'
Even though the 'transmat_' attribute is set, it will be overwritten during initialization because
'init_params' contains 't'
Even though the 'emissionprob_' attribute is set, it will be overwritten during initialization becau
se 'init_params' contains 'e'

['B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'B',
'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A']
```

In [12]:
```
#compute accuracy
y_true = data['Hidden'].values

acc = 0
m = len(y_true)

for i in range(m):
    if(y_true[i] == out_new[i]):
        acc += 1

acc = acc/m
print(acc*100)
```

```
70.0
```

In [ ]:
```

```