CS362
Lab Assignment 6
Week-7

Group: Spirit

Devang Patel - 202051062
Anik Bepari - 202051024
Aditya Tomar - 202051012
Manoj kumar gautam – 202051114

Learning Objective:

To implement Expectation Maximization routine for learning parameters of a Hidden Markov Model, to be able to use the EM framework for deriving algorithms for problems with hidden or partial information.

Reference:

1. A Revealing Introduction to Hidden Markov Models, Mark Stamp, 2018
2. What is the expectation maximization algorithm? Chuong B Do and Serafim Batzoglou, Nature Biotechnology, Vol 26, Num 8, August 2008

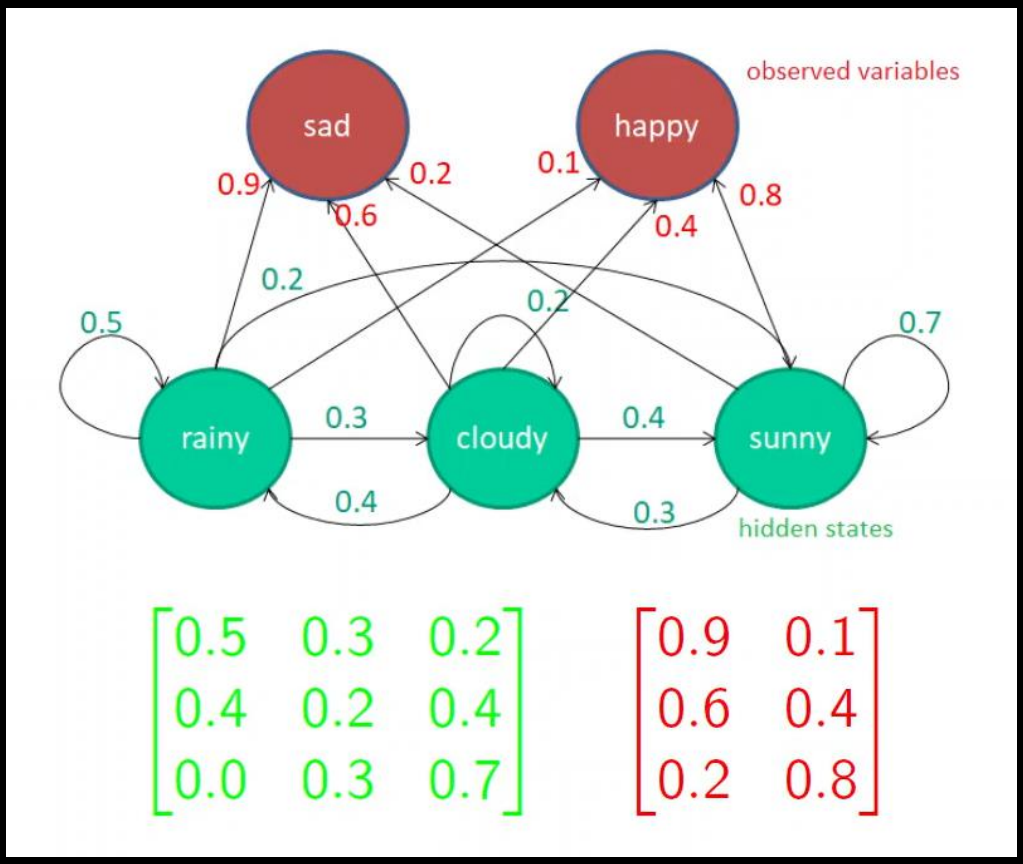
Problem Statement:

A. Read through the reference carefully. Implement routines for learning the parameters of HMM given in section 7. In section 8, “A not-so-simple example”, an interesting exercise is carried out. Perform a similar experiment on “War and Peace” by Leo Tolstoy.

Hidden Markov Model

Hidden markov model is a statistical model, in which system being modeled is assumed to be a markov chain - call it X - with unobserved ("hidden") states.

HMM assumes that there is another process Y(observable), whose behavior depends on X.



An HMM is characterized by

States: A set of states $s=s_1 ,.....s_n$
n: number of states.
m: The number of distinct observation symbols per state.
Transition probabilities: $aij=a_{11},a_{1,2},.....a_{n,n}$. Each $a_{i,j}$ represents the probability of transitioning from state s_i to s_j.

Emission probabilities: set B of functions of form $b_i(o_t)$ which is the probability of observation o_t being emitted by s_i.
Initial state distribution: π_i is the initial probability that s_i is the start state.
$\lambda=(aij,b_i(o_t) ,\pi_i)$: Complete parameter set of HMM.

HMM can be used to solve 3 problems:

1. evaluation problem-

Compute the probability of given observation sequence. (given λ).
We solve it using forward-Backward algorithm - Baum-welch algorithm.
Forward probablity:
What is the probablity that, given an HMM λ, at a time t the state is I and the partial observation $o_{1....}o_t$ has been generated?
$\alpha_t(i)$: $=P(o_{1....}o_t ,q_t =s_i \mid \lambda)$

Backward probablity:
What is the probablity that, given an HMM λ, at a time t the state is I and the partial observation $o_{1....}o_t$ has been generated?
$\beta_t(i):=P(o_{t+1....}o_T ,q_t =s_i \mid \lambda)$

2. decoding problem-

Given an observation sequence, compute the most likely hidden state sequence (given λ).
We solve it using Viterbi's algorithm.

3. Learning problem-

Given an observation sequence, we find the model parameters λ,to maximize P(O| λ).
(find λ).
We use Expectation maximization (EM) algorithm.

Algorithm:

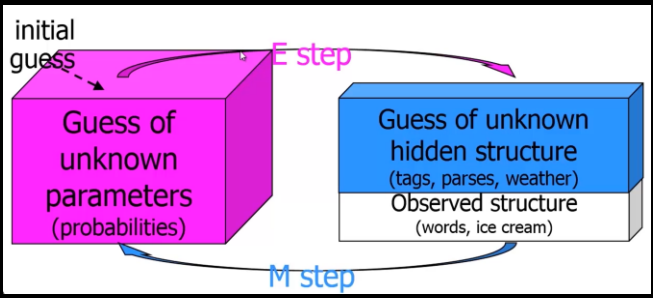
The E-step:

- Making an initial guess for a and b.
- Compute the forward and backward probabilities for a given model and reconstruct the hidden structure.

The M-step:

- Using that hidden structure re-estimate the model parameters.

- Repeat this until convergence



Review of probablities :

- Forward probablitiy, $\alpha_t(i)$:
  The probablity of being in state s_i given the observation $o_1,\ldots o_t$.
- Backward probablity, $\beta_t(i)$:
  The probablity of being in state s_i given the observation $o_{t+1},\ldots o_T$.
- State probablity, $\gamma_t(i)$:
  The probablity of being instate s_i, given the complete observation $o_1,\ldots o_T$.
- Transition probablity,$\xi_t(i)$:
  The probablity of goinf from state s_i to state s_j ,given the complete observation $o_1,\ldots o_T$.

HMM Scaling :

Inference for Hidden Markov Models (HMMs) using the standard recursions is numerically unstable. One approach to resolving this issue is to use an idea called *scaling factors.* The basic idea of scaling factors is to modify the standard recursions in the forward-backward algorithm to track conditional distributions that are numerically stable
The same scale factor is used for $\beta_t(i)$ as was used for $\alpha_t(i)$, namely c_t ,so that we have $\beta_t(i) = c_t\beta_t(i)$. We then compute $\gamma_t(i, j)$ and $\gamma_t(i)$ using the formulae of the previous section with $\alpha_t(i)$ and $\beta_t(i)$ in place of $\alpha_t(i)$ and $\beta_t(i)$, respectively. The resulting gammas and di-gammas are then used to re-estimate π, A and B.

Implementing EM to learn our parameters

Steps in code:

- Parameter estimation (intialization).
- Computing α, β and γ.
- HMM scaling
- And using them to re Estimate our parameters.

B. Ten bent (biased) coins are placed in a box with unknown bias values. A coin is randomly picked from the box and tossed 100 times. A file containing results of five hundred such instances is presented in tabular form with 1 indicating head and 0 indicating tail. Find out the unknown bias values. (2020_ten_bent_coins.csv) To help you, a sample code for two bent coin problem along with data is made available in the work folder: two_bent_coins.csv and embentcoinsol.m

Expectation Maximization Algorithm (EM)

What is Estimation?

Estimate our expectation from machines and then classify the data into some classes.

What is Maximization?

Whatever we estimated,should now be maximized.

How it works?

From given data EM learn a theory which tells that how each example to be classified and how to predict the feature value of each class.

For this it starts from a random classify data and repeat the two steps untl a clear result is formed.

Two steps?

E: Classify the data using the current theory.

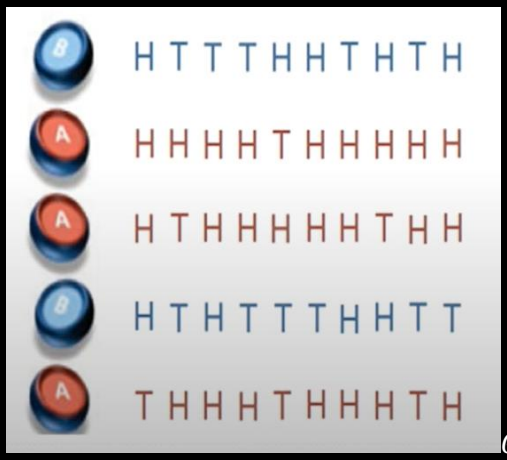
M:Generate the best theory using the theory using the current classification of the data.

- Step E generates the expected classification for each example .
- Step M generates the most likely theory given the classified data.

Applications:
Artificial intelligence, Natural language processing

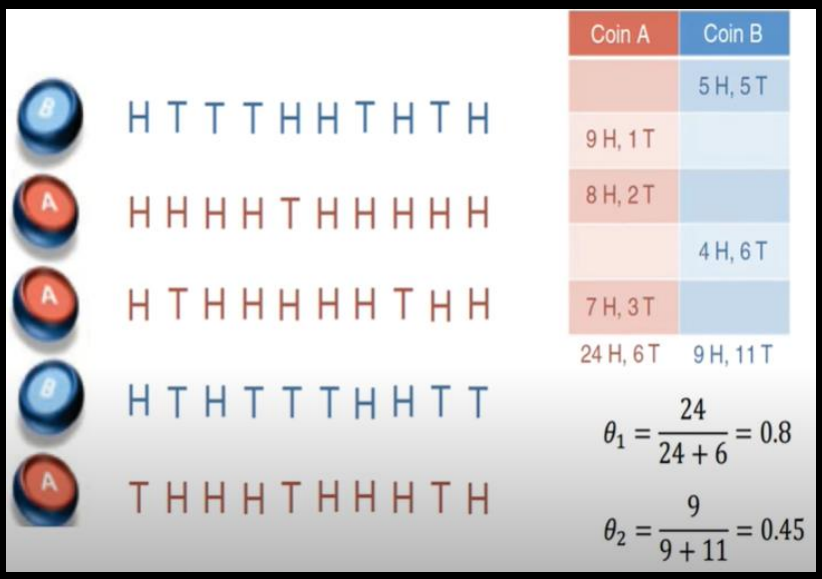
Method:
Assume that we have two coins,C1 and C2
Assume the bias of C1 is θ1(i.e., probability of getting heads with C1)
Assume the bias of C2 is θ2(i,e., probability of getting heads with C2)
We want to find θ1,θ2 by performing a number of trials(i.e, coin tosses)
Lets
We choose 5 times one of the coins.

We toss the chosen coin 10 times.



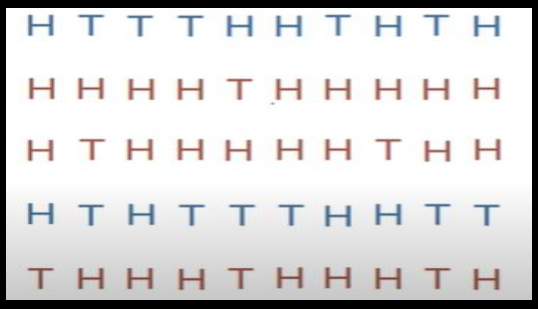
$\theta 1 = (number\ of\ heads\ using\ C1)\ /\ (total\ number\ of\ flips\ using\ C1)$

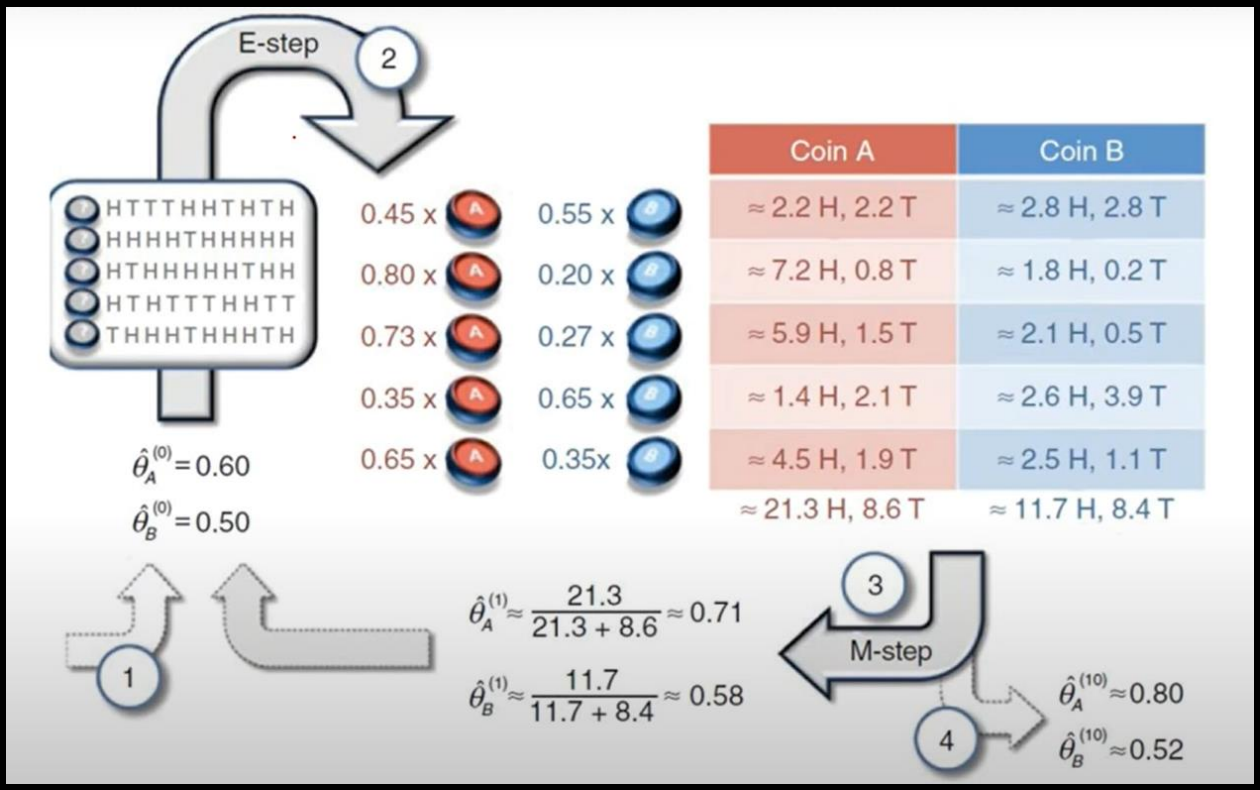
$\theta 2 = (number\ of\ heads\ using\ C2)/\ (\ total\ number\ of\ flips\ using\ C2)$



Assume a more problem

In here we do not know the identities of the coins used for each set of tosses (we treat them as hidden variables).





· Θ_A = .6
· Θ_b = .5
· First Round = 5 Head and 5 tails
· Compute likelihood using binomial distribution

$$P(K) = \theta^k (1-\theta)^{n-k}$$

Liklyhood of A $= {}^{P_A}(h)^h \ (1-{}^{P_A}(h))^{10-h} \rightarrow .6^5(1-.6)^{10-5}$
$= .0007962624$

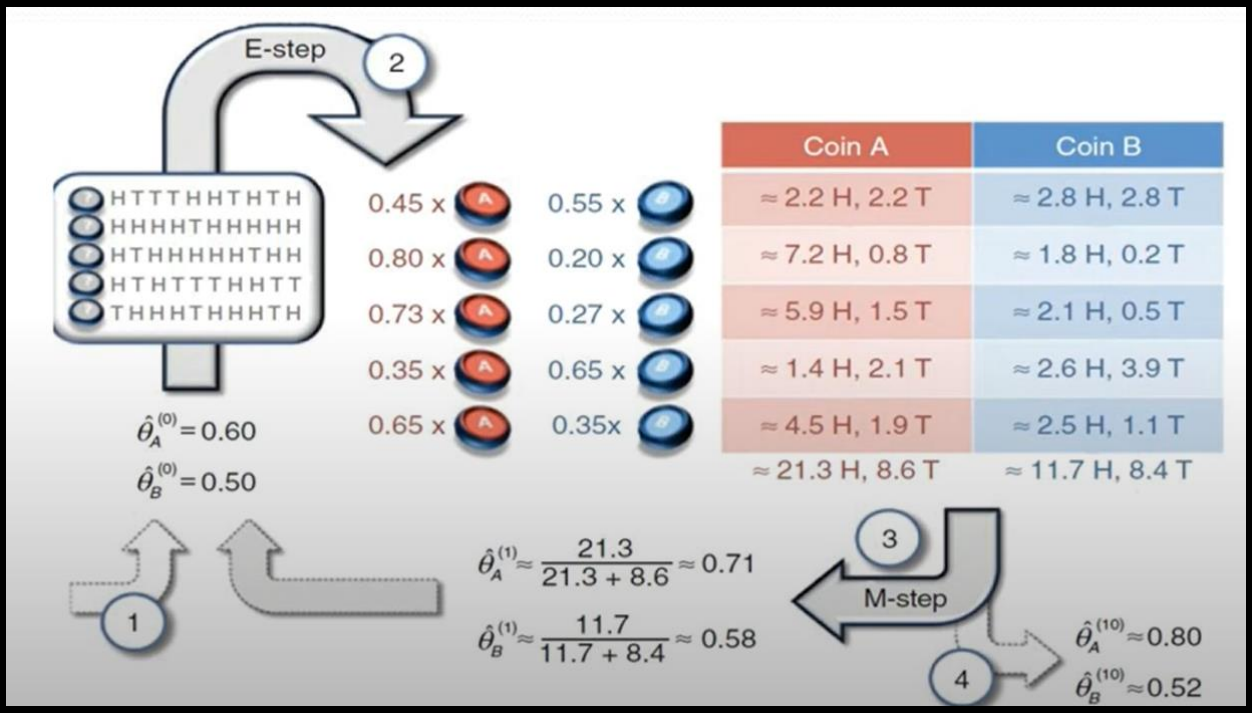
Liklyhood of B $= {}^{P_b}(h)^h \ (1-{}^{P_b}(h))^{10-h} \rightarrow .5^5(1-.5)^{10-5}$

$= 0.0009765625$

Normalize by using A/A+B = .00079624/(.00079624+0.0009765625) = .45
For B = .55

· Estimate Likely No of Heads and Tails for First toss

· For A -: H=.45*5=2.2,T=.45*5=2.2
· For B-: H == .55*5= 2.8,T=.55*5=2.8



CODE:

```
import numpy as np
import pandas as pd
from scipy.special import comb
```

```
np.random.seed(0)
df = pd.read_csv("2020_ten_bent_coins.csv").transpose()
head_counts = df.sum().to_numpy()
tail_counts = 100 - head_counts
coin_selected = np.random.randint(0, 10, size=(500,))
_, count_coins_selected = np.unique(coin_selected, return_counts=True)
MLE_vector = np.zeros(10)
```

```
head_counts
```

```
array([96, 26, 40, 21, 42, 89, 44, 31, 49, 10, 89, 42, 60, 65, 24, 83, 62,
       75, 79, 38,  1, 41, 53, 69, 14, 65,  0, 45, 88, 81, 48,  9, 77, 59,
        2, 68,  1, 12,  9, 89, 28, 82, 32, 77, 36,  0,  5, 35,  0, 72, 77,
        7,  0, 62, 80, 16, 90, 38, 78, 31, 38, 94, 15, 13, 11,  3, 92, 51,
       57, 10, 66,  7, 71, 37, 43, 45, 17, 29,  0, 77,  6, 57, 92, 88, 79,
        7, 25, 62, 86, 62, 54, 53, 52, 19, 10, 66, 23,  1, 40, 49, 17, 13,
       17, 18, 50, 87, 32, 49, 49, 66,  0,  0, 72, 88,  9,  4, 40, 61, 11,
       71, 77, 11, 49, 75, 92, 75, 16, 50, 21,  1, 65, 36,  9, 56, 72, 86,
        2, 77, 10, 64, 33, 95, 21, 70, 66,  2, 48, 59, 12,  2, 93,  2, 48,
       53, 38, 34, 25,  2, 10, 63,  5, 40, 30, 16, 90,  7, 71, 89,  9, 47,
       71, 65, 78, 72,  0, 27,  1, 52, 50, 17, 30, 56, 25,  0, 84, 14, 41,
       93, 41, 36, 34, 72, 68, 75, 17,  0, 72, 44, 59, 89, 78, 88, 70,  7,
       67, 30, 61, 35, 60,  0, 11,  1, 45,  8, 82, 48, 13, 56, 86, 23, 24,
       51, 38, 42, 68,  0, 22, 79, 69, 67, 79, 67, 18,  2, 40, 46, 72, 92,
       85, 46, 56,  2, 27, 84, 30, 59, 38,  3, 29, 78,  0, 84, 70, 36, 34,
       44, 51, 68, 39, 77, 58, 90, 87, 10, 75, 76, 25, 47, 89, 13, 78, 71,
       40, 77, 18, 27, 89, 53, 52, 75, 36, 48, 53, 41,  1, 49,  1, 49,  0,
       60,  9,  0, 64, 90, 74, 71, 63, 81, 47, 33, 48,  8, 12, 52, 50, 44,
       15, 19,  1, 88, 19, 82, 51, 94, 61, 58, 86, 48, 12, 20, 45, 45, 62,
        1, 57, 74, 43, 91, 15, 25, 43, 88, 91, 90, 60, 24, 41, 82,  1, 47,
       56, 66, 85, 80, 27, 77, 70, 68, 10, 92,  1, 45, 52, 30,  9, 10, 89,
       94, 10, 84,  7, 58, 22, 45, 18, 55, 54, 57, 50, 67, 91,  2, 54,  1,
       38, 26, 25, 58, 50, 80, 15, 70, 69, 28, 36, 37,  0, 90, 29, 57, 20,
       92, 88,  0, 36, 10, 88, 78, 44, 74, 83,  1, 86, 57,  0, 66, 38, 40,
       34, 40, 61,  3, 20, 23,  6, 16, 71, 78, 43, 37, 34, 78, 16, 33, 36,
       96, 52, 89, 67, 80, 69, 36, 72, 84,  3, 80, 30, 81, 29,  7, 23,  1,
       25, 11,  9, 38, 45, 77, 33, 49,  9, 56, 16, 21, 36, 79, 44, 31, 14,
       89,  1, 17, 90, 25, 17, 43, 48, 16, 29, 54, 35, 69, 19, 72, 17, 92,
        7,  3,  1,  9, 17, 78, 32, 88,  0, 91, 78, 89, 95, 27, 65, 89, 64,
        7, 57, 25,  1, 52, 27, 46], dtype=int64)
```

```
for i,j in zip(head_counts, coin_selected):
    MLE_vector[j]+=i
```

```
MLE_vector = MLE_vector/(count_coins_selected*100)
```

```
def compute_likelihood(obs, n, pheads):

    likelihood = comb(n, obs, exact=True)*(pheads**obs)*(1.0-pheads)**(n-obs)

    return likelihood
```

```
np.random.seed(0)
p_heads = np.zeros((100,10))
p_heads[0] = np.random.random((1,10))
print(p_heads[0])
eps = 0.001
improvement = float('inf')

epoch = 0
while improvement>eps:

    expectation = np.zeros((10, 500, 2))

    for i in range(500):
        eH = head_counts[i]
        eT = tail_counts[i]

        likelihood = np.zeros(10)
```

```
        for j in range(10):

            likelihood[j] = compute_likelihood(eH, 100, p_heads[epoch][j])

        weights = likelihood/np.sum(likelihood)
        for j in range(10):
            expectation[j][i] = weights[j]*np.array([eH, eT])

    thetas = np.zeros(10)
    for i in range(10):
        thetas[i] = np.sum(expectation[i], axis = 0)[0]/ np.sum(expectation[i])


    p_heads[epoch+1] = thetas
    print(f'Epoch: {epoch}\nthetas: {thetas}')

    improvement = max(abs(p_heads[epoch+1] - p_heads[epoch]))
    epoch+=1
```

Output:

```
[0.5488135  0.71518937 0.60276338 0.54488318 0.4236548  0.64589411
 0.43758721 0.891773   0.96366276 0.38344152]
Epoch: 0
thetas: [0.54030055 0.74700458 0.60774813 0.53612429 0.39390495 0.66432208
 0.42455597 0.87913902 0.9408577  0.16211897]
Epoch: 1
thetas: [0.5302535  0.75955475 0.61332103 0.5258201  0.35874312 0.67708556
 0.40729141 0.87384035 0.92352302 0.10546534]
Epoch: 2
thetas: [0.52018909 0.76576297 0.61839452 0.5155784  0.31426446 0.68593045
 0.39912291 0.87020696 0.9136364  0.08328092]
Epoch: 3
thetas: [0.51163503 0.76889438 0.62203863 0.50672204 0.27727676 0.69178149
 0.39765551 0.86830699 0.90930678 0.07144459]
Epoch: 4
thetas: [0.50591337 0.77056051 0.62378851 0.50050643 0.24532125 0.69564221
 0.39109125 0.86756872 0.90765306 0.0584879 ]
Epoch: 5
thetas: [0.50177514 0.77150992 0.62403936 0.49571276 0.22302672 0.69828232
 0.38031544 0.86742743 0.90706774 0.04969973]
```

In [48]:

```
for i, j in enumerate(thetas):
    print(f"{i+1} & {j:.3f} \\\\ \\hline")
```

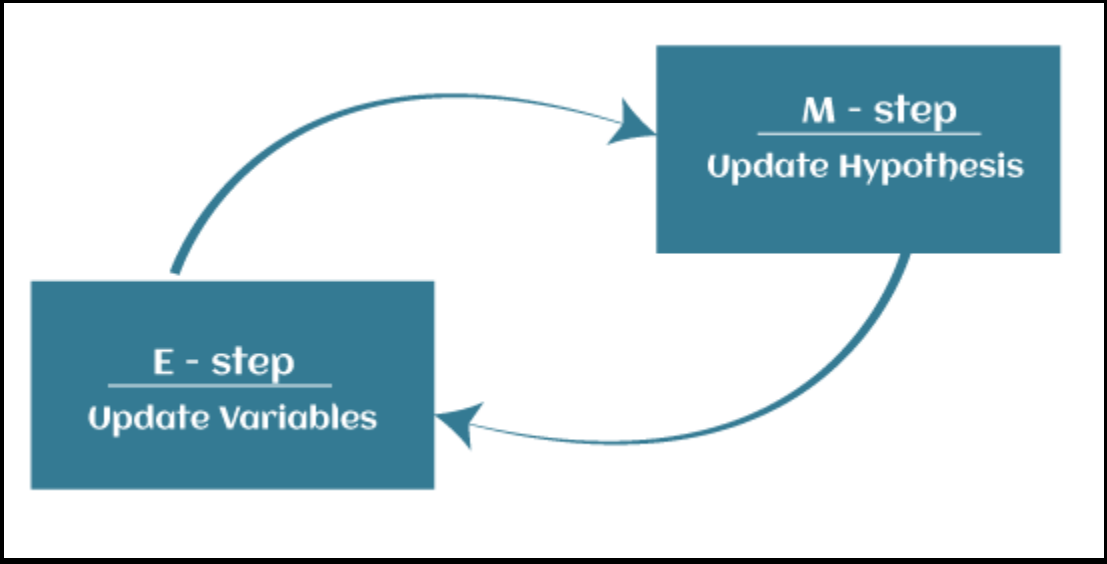
```
1 & 0.409 \\ \hline
2 & 0.702 \\ \hline
3 & 0.493 \\ \hline
4 & 0.305 \\ \hline
5 & 0.095 \\ \hline
6 & 0.595 \\ \hline
7 & 0.187 \\ \hline
8 & 0.789 \\ \hline
9 & 0.898 \\ \hline
10 & 0.010 \\ \hline
```

C. A point set with real values is given in 2020_em_clustering.csv. Considering that there are two clusters, use EM to group together points belonging to the same cluster. Try and argue that k-means is an EM algorithm.

- **Expectation Maximization algorithms: -**

The EM algorithm is the combination of various unsupervised algorithms, **being** an iterative approach, it consists of two modes.

1) **Expectation step (E - step):** we estimate the missing or latent variables.
2) **Maximization step (M - step):** This step involves the use of estimated data in the E-step and updating the parameters.
3) **Repeat E-**step and M-step until the convergence of the values occurs.

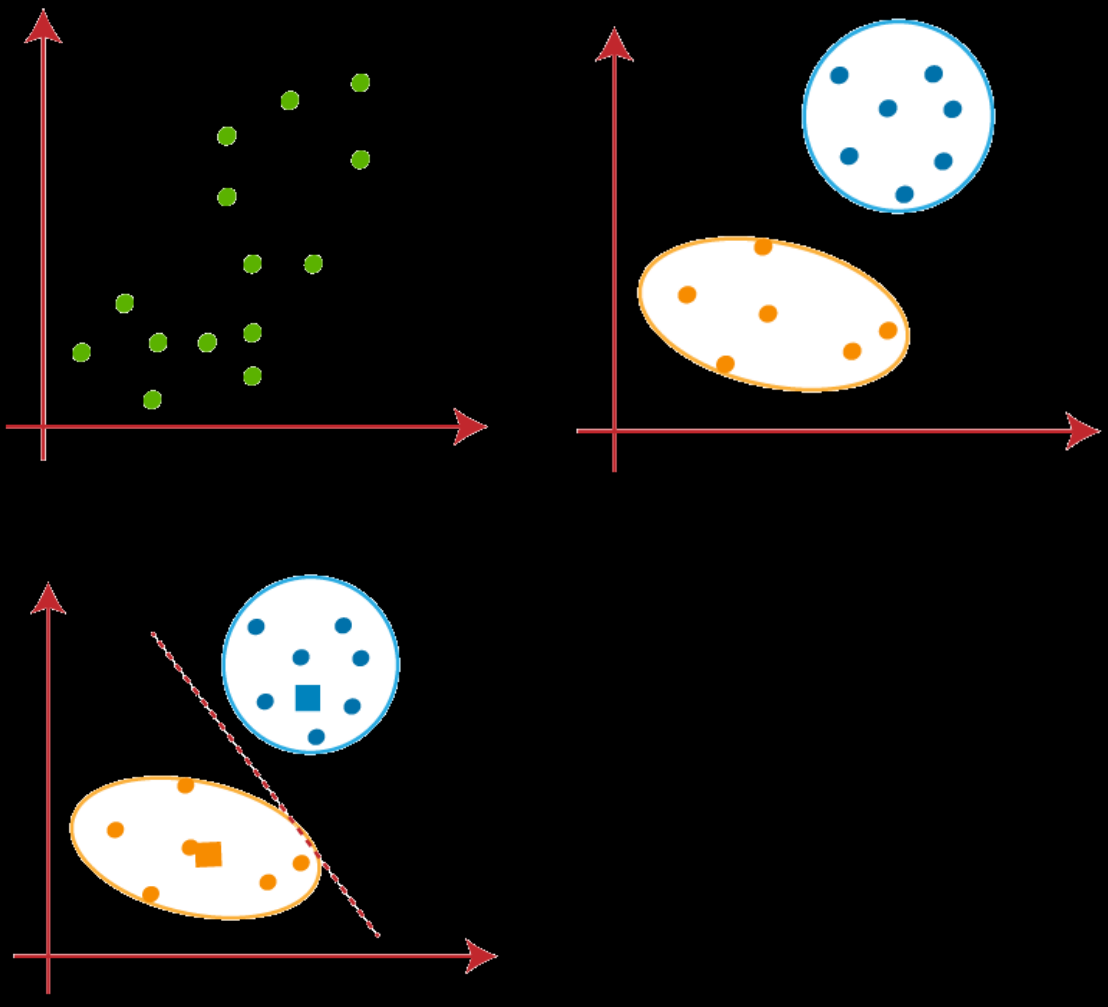


Gaussian Mixture Model (GMM)

GMM is defined as a mixture model that has a combination of the unspecified probability distribution function. GMM also requires estimated statistics values such as mean and standard deviation or parameters.

K-Means Clustering Algorithm

- **K – means** algorithm is an iterative algorithm that tries to partition the dataset into k per – defined disnict non- overlapping subgroups where each data point belongs to only one group.
- It tries to make the intra-cluster data points as possible while also keeping the clusters as diffent as possible.
- It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of the all points that belong to that cluster) is at the minimum.



Reference:

1. A Revealing Introduction to Hidden Markov Models, Mark Stamp, 2018
2. What is the expectation maximization algorithm? Chuong B Do and Serafim Batzoglou, Nature Biotechnology, Vol 26, Num 8, August 2008
3. HMM AND EM https://youtu.be/h22nGEF8PUo
4. https://github.com/nlok5923/CS302
5. https://github.com/TanmayAmbadkar/CS302-AI/tree/master/Lab6
6. https://youtu.be/bSPSZyOfKH0