Machine Learning-II(BAI702)

Internal Assessment Test II– November 2025

Branch: AIML,CSE(AIML)

1.a.Explain the main steps involved in the K-Means algorithm. Write a short note on dealing with noise in K-Means clustering.(5 Marks)

The k-Means Algorithm (3 marks)

• Initialisation

– choose a value for k

– choose k random positions in the input space

– assign the cluster centres μj to those positions

• Learning

– repeat

* for each datapoint xi:

· compute the distance to each cluster centre

· assign the datapoint to the nearest cluster centre with distance

$$d_i = \min_j d(\mathbf{x}_i, \boldsymbol{\mu}_j).$$

* for each cluster centre:

· move the position of the centre to the mean of the points in that cluster

(Nj is the number of points in cluster j):

$$\boldsymbol{\mu}_j = \frac{1}{N_j}\sum_{i=1}^{N_j} \mathbf{x}_i$$

– until the cluster centres stop moving

• Usage

– for each test point:

* compute the distance to each cluster centre

* assign the datapoint to the nearest cluster centre with distance

$d_i = \min_j d(\mathbf{x}_i, \boldsymbol{\mu}_j).$

Effect of Noise on K-Means(2 marks)

- K-Means relies on the **mean** of points in a cluster.
- The mean is **sensitive to outliers**. Even a few noisy datapoints can pull the cluster centre away from its true position.
- As a result, noise can seriously distort cluster boundaries.

Methods to Handle Noise:

1. *Replace the Mean with a Median (Robust K-Means)*
   Marsland explains that the **median** is a *robust statistic* that does not get influenced strongly by outliers. For example, in the data (1, 2, 1, 2, 100),
   - Mean = 21.2 (strongly affected by the outlier 100)
   - Median = 2 (not affected)
     By recomputing the cluster centre using the median instead of the mean, noise effects are significantly reduced. This modification handles noisy inputs but increases computational cost.
     Machine-learning- Stephen Marsh…
2. *Cluster Replacement as Noise Removal*
   K-Means naturally performs a kind of **denoising** by replacing each datapoint with its cluster centre. When clusters are correctly chosen, noisy datapoints get mapped to a cleaner prototype value.

1.b.Explain Vector Quantisation and show how it is related to SOM(5 marks)

Vector Quantisation (2 Marks)

Vector Quantisation is a technique used to represent a large set of continuous input vectors using a smaller set of prototype vectors (codebook). Marsland explains that **cluster centres represent typical values** in the input space, and every input vector is replaced by the cluster centre closest to it. This forms the essence of vector quantisation.

Relationship Between Vector Quantisation and SOM (3 Marks)

(i) Both replace input vectors with representative prototypes

In VQ, the input is replaced by the nearest codebook vector.
In SOM, the input activates the closest neuron (winner), whose weight is the prototype vector.

(ii) Both use competitive learning

Vector quantisation typically uses competitive learning where the closest prototype updates.
SOM extends this idea: not only the winning neuron but **its neighbours** also update.

(iii) SOM adds topology to vector quantisation

This is the key difference.
In VQ (like K-Means):

- Cluster centres are independent of each other.

In SOM:

- Neurons are arranged in a grid.
- Updating one neuron also updates its neighbours → this preserves spatial relationships.

(iv) SOM produces an ordered codebook

VQ gives a set of prototype vectors with no structure.
SOM produces a **topologically ordered** set of prototypes, which makes it useful for:

- visualisation,
- dimensionality reduction,
- interpreting high-dimensional data.

(v) Learning Rule Connection

SOM weight update rule:

$$w_j(t+1) = w_j(t) + \eta(t)\, h_{j,i(x)}(t)\, (x - w_j(t))$$

This is an extension of competitive learning used in vector quantisation, but with the added neighbourhood function *h* controlling topological adaptation

2. Describe network dimensionality and boundary conditions in SOM (10 marks)

SOM Dimensionality (2 Marks)

SOMs are commonly implemented on a 2D rectangular grid, the algorithm itself places no restriction on using only two dimensions.

- A SOM can be 1D (a line), 2D (a grid), or even 3D, depending on the nature of the input data.
- The choice depends on the intrinsic dimensionality of the data, not the dimension in which the data is embedded.

Example:
If data points lie on a plane inside a 3D room, their intrinsic dimensionality is 2, although they appear in a 3D space. Using a 2D SOM is therefore sufficient.

Noise may increase the apparent dimensionality of data, and identifying intrinsic dimensionality helps reduce such noise.

Boundary Conditions in SOM (2 Marks)

The edges of the SOM grid represent boundaries. In some tasks, boundaries make sense.

- Example: Arranging sounds from low pitch to high pitch, where the endpoints are fixed.

However, not all datasets have clear boundaries. In such cases, keeping a rectangular map creates edge neurons that behave differently from inner neurons.

Removing Boundaries: Circular and Torus Topologies (3 Marks)

To avoid edge effects, SOMs can be modified using wrap-around boundaries.

(i) 1D SOM → Circle

Tie the ends together so the first and last neurons become neighbours.

(ii) 2D SOM → Torus

Join the top–bottom and left–right edges.

- First bend the map into a tube (top meets bottom).
- Then bend the tube into a donut shape (torus) by joining the ends.

This ensures no neuron lies on an edge, giving more uniform neighbourhood behaviour.

Marsland notes that torus-based SOMs often perform better than rectangular ones, though the reason is not always obvious.

Distance Calculation with Wrap-Around (2 Marks)

Wrap-around topologies complicate distance computation because we must consider the shortest path across boundaries.

Two ways to compute distances:

1. Modulo arithmetic (wrap index around map size).
2. Map tiling method:
    - Imagine copies of the original map placed above, below, left, right, and diagonally.
    - For any pair of neurons, compute distances to all copies of the target neuron.
    - Choose the smallest distance as the true toroidal distance.

By treating x and y distances separately, the number of required computations decreases.

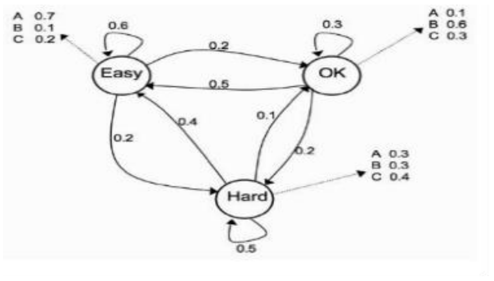
SOM Size and Resolution (1 Mark)

The number of neurons is fixed before training.

- Few neurons → coarse, generalised representation.
- Many neurons → fine-grained detail, but higher computational cost.

Thus, SOM size must balance representation quality and efficiency.

3.Compute the most likely path through the HMM using the Viterbi algorithm.(10 Marks)



Viterbi computation

States order used: E (Easy), O (OK), H (Hard). Observations o1=A, o2=B, o3=C.

Initialization (t = 1)

$$\delta_1(s) = \pi(s)\, b_s(o_1)$$

- $\delta_1(E) = (1/3)\cdot 0.7 = 0.2333333$
- $\delta_1(OK) = (1/3)\cdot 0.1 = 0.0333333$
- $\delta_1(H) = (1/3)\cdot 0.3 = 0.1000000$

Backpointers ψ_1 are `None`.

Recursion (t = 2, observation B)

$$\delta_2(s) = \max_{s'} \delta_1(s')\, a_{s's}\, b_s(o_2)$$

Compute for each target s:

- $\delta_2(E)$ = max over prev {
  E→E: 0.2333333·0.6·0.1 = 0.01399999998 (prev=E)
  OK→E: 0.0333333·0.5·0.1 = 0.0016666667
  H→E: 0.1000000·0.4·0.1 = 0.004
  } ⇒ **$\delta_2(E)$=0.01399999998**, $\psi_2(E)$=E
- $\delta_2(OK)$ = max {
  E→OK: 0.2333333·0.2·0.6 = 0.027999999996 (prev=E)
  OK→OK: 0.0333333·0.3·0.6 = 0.00599999994
  H→OK: 0.1000000·0.1·0.6 = 0.006
  } ⇒ **$\delta_2(OK)$=0.028**, $\psi_2(OK)$=E
- $\delta_2(H)$ = max {
  E→H: 0.2333333·0.2·0.3 = 0.01399999998 (prev=E)
  OK→H: 0.0333333·0.2·0.3 = 0.00199999998
  H→H: 0.1000000·0.5·0.3 = 0.015
  } ⇒ **$\delta_2(H)$=0.015**, $\psi_2(H)$=H

So after t=2: δ_2 = [0.0140, 0.0280, 0.0150], ψ_2 = [E, E, H].

Recursion (t = 3, observation C)

- $\delta_3(E)$ = max {
  E→E: 0.0140·0.6·0.2 = 0.00168 (prev=E)
  OK→E: 0.0280·0.5·0.2 = 0.0028 (prev=OK)
  H→E: 0.0150·0.4·0.2 = 0.0012
  } ⇒ **$\delta_3(E)$=0.0028**, $\psi_3(E)$=OK
- $\delta_3(OK)$ = max {
  E→OK: 0.0140·0.2·0.3 = 0.00084 (prev=E)
  OK→OK: 0.0280·0.3·0.3 = 0.00252 (prev=OK)
  H→OK: 0.0150·0.1·0.3 = 0.00045
  } ⇒ **$\delta_3(OK)$=0.00252**, $\psi_3(OK)$=OK
- $\delta_3(H)$ = max {
  E→H: 0.0140·0.2·0.4 = 0.00112 (prev=E)
  OK→H: 0.0280·0.2·0.4 = 0.00224 (prev=OK)
  H→H: 0.0150·0.5·0.4 = 0.003 (prev=H)
  } ⇒ **$\delta_3(H)$=0.003**, $\psi_3(H)$=H

So δ_3 = [0.0028, 0.00252, 0.003], ψ_3 = [OK, OK, H].

Termination

Choose final state with max δ_3: max is $\delta_3(H)$=0.003 ⇒ final state = **H (Hard).**

Backtrack:

- t=3: state H. $\psi_3(H)$=H (prev at t=2 is H)
- t=2: state H. $\psi_2(H)$=H (prev at t=1 is H)
- t=1: state H.

→ **Most likely path =** `H → H → H` **(Hard, Hard, Hard).**

4.Explain Approximate Inference techniques in Bayesian Networks(10 marks)

Sampling-based (Stochastic) Approximate Inference

These methods use random sampling to estimate posterior probabilities.

(i) Monte Carlo Sampling (2 Marks)

Monte Carlo methods use repeated random sampling to approximate a distribution.
The accuracy improves with the number of samples.

P(X = x) \approx \frac{\text{# of samples where } X=x}{\text{total samples}}

Widely used when the BN is large and exact inference is infeasible.

(ii) Prior Sampling (1 Mark)

- Samples are generated from the BN following the topological order.
- Simple but inefficient when evidence has low probability → many samples get rejected.

(iii) Rejection Sampling (1 Mark)

- Generate full samples from the BN.
- Reject samples inconsistent with evidence.
- Works poorly if evidence is rare → almost all samples are rejected.

(iv) Likelihood Weighting (2 Marks)

One of the most popular approximate inference algorithms.

- Evidence variables are fixed.
- Each sample receives a weight equal to the likelihood of the evidence.
- Eliminates rejection and improves efficiency.

w=∏evidence EiP(Ei∣ parents(Ei))w = \prod_{\text{evidence } E_i} P(E_i \mid \text{parents}(E_i))w=evidence Ei∏P(Ei∣ parents(Ei))

(v) Markov Chain Monte Carlo (MCMC), especially Gibbs Sampling (2 Marks)

Gibbs Sampling:

- Keeps evidence fixed.
- Repeatedly samples one variable at a time conditioned on all others.
- Eventually converges to the true posterior distribution.

Advantages:

- Works well on large and connected networks.
- Handles complex evidence.

2. Deterministic Approximate Inference Methods

These provide fast, approximate answers without random sampling.

(i) Loopy Belief Propagation (1 Mark)

- Extension of belief propagation to networks with cycles (loops).
- Passes messages iteratively until convergence.
- Not guaranteed to be exact, but performs surprisingly well in practice (e.g., error-correcting codes).

(ii) Variational Inference (1 Mark)

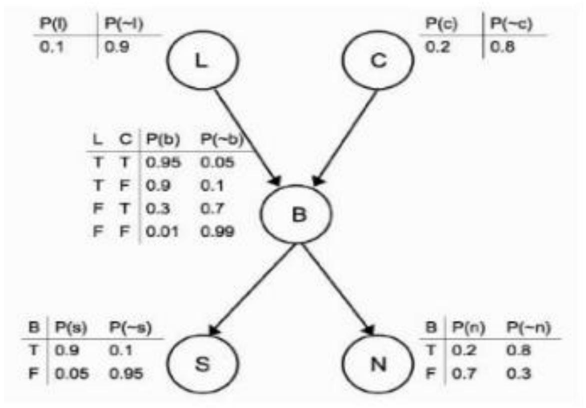
Transforms the original complex distribution into a simpler, tractable distribution by minimizing KL divergence.

Approximate the true distribution PPP with a simpler QQQ:

Q≈Psuch that computation becomes feasibleQ \approx P \quad \text{such that computation becomes feasible}Q≈Psuch that computation becomes feasible

Used in large BNs and graphical models because it scales well.

5.Compute the probability of taking notes (N) in the Bayesian network. The problem describes the chance of you taking notes in the lecture or sleeping (S) according to whether or not the course was boring (B)based on whether or not the professor is boring (L) and the content is dull (C). Compute the chance of falling asleep in a lecture given that both the professor and the course are boring.(10 marks)



Conditional probabilities (5 marks)

- Prior: $P(L) = 0.1,\ P(\neg L) = 0.9.\ P(C) = 0.2,\ P(\neg C) = 0.8.$
- $P(B \mid L, C)$ table:
  - $P(B{=}T \mid L{=}T, C{=}T) = 0.95$
  - $P(B{=}T \mid L{=}T, C{=}F) = 0.90$
  - $P(B{=}T \mid L{=}F, C{=}T) = 0.30$
  - $P(B{=}T \mid L{=}F, C{=}F) = 0.01$
- Emissions:
  - $P(S{=}T \mid B{=}T) = 0.90,\ P(S{=}T \mid B{=}F) = 0.05.$
  - $P(N{=}T \mid B{=}T) = 0.20,\ P(N{=}T \mid B{=}F) = 0.70.$

1) Marginal probability of taking notes $P(N)$

Use the law of total probability over L, C, B. A compact form:

$$P(N) = \sum_{l \in \{T,F\}} \sum_{c \in \{T,F\}} P(l)P(c)\Big[P(B{=}T \mid l, c)P(N \mid B{=}T) + (1 - P(B{=}T \mid l, c))P(N \mid B{=}F)\Big]$$

Plugging the numbers (computing each (l, c) term) gives:

- For $L = T, C = T$: $0.1 \times 0.2\big[0.95 \cdot 0.2 + 0.05 \cdot 0.7\big] = 0.02 \cdot (0.19 + 0.035) = 0.02 \cdot 0.225 = 0.0045$
- For $L = T, C = F$: $0.1 \times 0.8\big[0.90 \cdot 0.2 + 0.10 \cdot 0.7\big] = 0.08 \cdot (0.18 + 0.07) = 0.08 \cdot 0.25 = 0.020$
- For $L = F, C = T$: $0.9 \times 0.2\big[0.30 \cdot 0.2 + 0.70 \cdot 0.7\big] = 0.18 \cdot (0.06 + 0.49) = 0.18 \cdot 0.55 = 0.099$
- For $L = F, C = F$: $0.9 \times 0.8\big[0.01 \cdot 0.2 + 0.99 \cdot 0.7\big] = 0.72 \cdot (0.002 + 0.693) = 0.72 \cdot 0.695 = 0.5004$

Sum all terms: $P(N) = 0.0045 + 0.020 + 0.099 + 0.5004 = 0.6239$.

Answer: $\boxed{P(N) \approx 0.6239\ (\approx 0.624).}$

Falling asleep condition (5 marks)

2) Probability of falling asleep given both professor and course are boring:

Compute $P(S \mid L{=}T, C{=}T)$.

Given L and C, only uncertainty is B. So

$$P(S \mid L, T, C, T) = \sum_{b \in \{T,F\}} P(b \mid L, T, C, T)\, P(S \mid b).$$

From the CPT: $P(B{=}T \mid L{=}T, C{=}T) = 0.95$. Thus

$$P(S \mid L, T, C, T) = 0.95 \cdot 0.90 + 0.05 \cdot 0.05 = 0.855 + 0.0025 = 0.8575.$$

Answer: $\boxed{P(S \mid L{=}T, C{=}T) = 0.8575\ (\approx 0.857).}$

6.a.Explain the Proposal Distribution and its importance in Markov Chain Monte Carlo(MCMC) sampling. (10 Marks)

MCMC (2 Marks)

Markov Chain Monte Carlo (MCMC) sampling is a family of algorithms used to draw samples from complex probability distributions when direct sampling is difficult or impossible.
The goal is to construct a Markov Chain whose *stationary distribution* is the target distribution P(x)P(x)P(x).

To move through the state space, MCMC does not sample directly from P(x)P(x)P(x).
Instead, it relies on a proposal distribution.

Proposal Distribution

A proposal distribution, usually denoted as q(x′∣ x)q(x'|x)q(x′∣ x), is a probability distribution used to propose the next candidate state x′x'x′ given the current state xxx.

Key ideas:

- It defines how the Markov chain *moves* through the state space.
- It does *not* need to resemble the target distribution.
- It only needs to be easy to sample from.
- The proposal distribution is accepted or rejected using an acceptance rule (e.g., Metropolis–Hastings).

Role of Proposal Distribution in Metropolis–Hastings

In the Metropolis–Hastings algorithm, a proposed state x′x'x′ is accepted with probability:

α=min(1,P(x′) q(x∣ x′)P(x) q(x′∣ x))\alpha = \min\left(1, \frac{P(x') \, q(x|x')}{P(x) \, q(x'|x)}\right)α=min(1,P(x)q(x′∣ x)P(x′)q(x∣ x′))

Thus the proposal distribution directly affects:

- How often new states are accepted,
- How efficiently the chain explores the distribution.

Importance of the Proposal Distribution (3 Marks)

(i) Controls Speed of Convergence

A good proposal distribution allows the chain to explore the state space quickly.
A poor proposal distribution leads to:

- very high rejection rates,
  OR
- tiny steps and very slow mixing.

(ii) Balances Exploration and Local Search

A good proposal distribution balances:

- Local moves (exploit local high-probability regions), and
- Global moves (jump to distant regions to avoid getting stuck).

(iii) Reduces Autocorrelation of Samples

If proposals are too small, consecutive samples are highly correlated.
Low autocorrelation = more *effective* samples = better approximation.

(iv) Ensures Ergodicity

The proposal must allow the chain to eventually reach all parts of the state space. Otherwise, the Markov chain fails to represent the true distribution.

6.b. State advantages of the Particle Filter over the Kalman Filter.

1. **Handles Non-linear Models (Major Advantage)**
    - The Kalman Filter assumes *linear* state-transition and observation models.
    - Particle Filters can handle *any* non-linear system because they use sampling instead of linear equations.
2. **Works with Non-Gaussian Noise**
    - Kalman Filter requires *Gaussian* process and measurement noise.
    - Particle Filter can approximate **arbitrary** noise distributions (multi-modal, skewed, heavy-tailed, etc.) using weighted particles.
3. **Can Represent Multi-modal Distributions**

- Kalman Filter always maintains a **single Gaussian** belief (unimodal).
- Particle Filters can represent beliefs with **multiple peaks**, uncertainty pockets, or complex shapes—important in tracking and robotics.

4. **Suitable for Complex, Real-world Systems**
   - Particle Filters can track moving objects, handle occlusion, missing data, and cluttered environments.
   - Kalman Filters struggle when the system is highly dynamic or unpredictable.
5. **More Robust to Model Inaccuracies**
   - Small modelling errors or wrong assumptions make Kalman Filters unstable.
   - Particle Filters are more robust because they rely on **sampling** rather than analytical equations.
6. **Scales Naturally to Higher-level Bayesian Filtering**
   - Particle filters can incorporate additional parameters, unknown variables, or changing model structures.
   - Kalman Filters require strict linear–Gaussian assumptions and cannot adapt flexibly.
7. **Capable of Variable-Level Approximation**
   - Increasing particle count improves accuracy without redesigning the model.
   - Kalman Filters do not have such tunable flexibility.