Homework 3

Manyara Bonface Baraka - mbaraka

March 31, 2025

Problem 1: Value Iteration for Markov Decision Processes

Value iteration is a dynamic programming aligorithm tha tis used to fid the optimal value function and policy for a markov decision process.

Understanid MDPs

MDP is defined by:

- **State (S)** possible configurations of the environment
- **Actions (A)** possible decisions or moves the agent can make
- **Transition Model (T)** probability of moving from one state to another given an action
- **Reward Function (R)** immediate reward received after transitioning from one state to another
- **Policy** (π) mapping from states to actions that defines the agent's behavior

Value funcion and policy that results:

Optimal Value Function and Policy:

The optimal value function and policy obtained from the value iteration algorithm are as follows:

Optimal Value Function:

$$\begin{bmatrix} 0.0689 & 0.0614 & 0.0744 & 0.0558 \\ 0.0919 & 0.0000 & 0.1122 & 0.0000 \\ 0.1454 & 0.2475 & 0.2996 & 0.0000 \\ 0.0000 & 0.3799 & 0.6390 & 0.0000 \end{bmatrix}$$

Optimal Policy:

$$\begin{bmatrix} 0 & 3 & 0 & 3 \\ 0 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 \end{bmatrix}$$

Episode Outcome: The episode finished with a reward of 1.0.

Problem 2: SARSA for Markov Decision Processes

SARSA(State-Action-Reward-State-Action) is an on-policy reignforcement learning alifgorithm used to lean the optimal policy for an MDP without knowing transition probabilities. Unlike value iteration which is a dynamic programming method requiring a known transition model, SARSA is am odel-free Learning aligorithm taht learns from direct experiment.

Understanding SARSA

SARSA is defined by the following key components:

- **State (S)**: The current configuration of the environment.
- **Action (A)**: The decision or move the agent takes in a given state.
- **Reward (R)**: The immediate feedback received after taking an action in a state.
- **Next State (S')**: The state the agent transitions to after taking an action.
- **Next Action (A')**: The action the agent plans to take in the next state.

The SARSA algorithm updates the Q-value (state-action value) using the following equation:

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma Q(S', A') - Q(S, A)\right]$$

where:

- $Q(S, A)$: The current estimate of the Q-value for state S and action A.
- α: The learning rate, which determines how much new information overrides old information.
- γ: The discount factor, which determines the importance of future rewards.
- R: The reward received after taking action A in state S.
- $Q(S', A')$: The Q-value of the next state-action pair.

SARSA is called an on-policy algorithm because it updates the Q-value based on the action A' that the agent actually takes, following its current policy.

Comparizon of SARSA vs Value Iteration

- **Value Iteration:**
  - Requires a known transition model and reward function.
  - Computes the optimal value function and policy by iteratively updating the value of each state.
  - Guarantees convergence to the optimal policy for MDPs.
  - Computationally expensive for large state spaces.
- **SARSA:**
  - Does not require a known transition model; learns from direct interaction with the environment.
  - Updates the Q-value based on the current policy, making it an on-policy algorithm.
  - Can adapt to changes in the environment during learning.
  - May converge to a suboptimal policy if the exploration strategy is not well-designed.

Value funcion and policy that results:

SARSA Optimal Value Function:

$$\begin{bmatrix} 0.068 & 0.111 & 0.209 & 0.098 \\ 0.063 & 0.000 & 0.373 & 0.000 \\ 0.082 & 0.197 & 0.412 & 0.000 \\ 0.000 & 0.317 & 0.643 & 0.000 \end{bmatrix}$$

SARSA Optimal Policy:

$$\begin{bmatrix} 1 & 3 & 3 & 3 \\ 0 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ 0 & 2 & 2 & 0 \end{bmatrix}$$

Why use SARSA instead of Value Iteration

Differences:

- **Policy:** SARSA's policy differs slightly due to exploration ($\epsilon = 0.1$), leading to safer paths avoiding holes while for Value Iteration's policy is deterministic and model-based.
- **Value Function:** SARSA's values are estimates based on experience, which are lower due to exploration while for Value Iteration it provides exact values using known transitions.

Why Use SARSA?

SARSA is model-free, making it suitable for environments where transition probabilities are unknown. It learns through interaction, essential for real-world applications without a known model. Value Iteration requires full knowledge of the environment's dynamics, which is often unavailable.

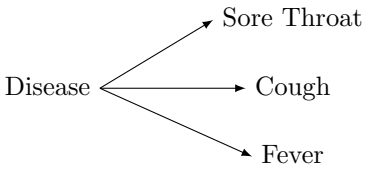
Problem 3: Bayesian Network

Bayesian network is the probabilistic graphical model that represents set of random vaiable and their Conditional independecies using the Directed Acyclic Graph. It helps in reasoning under uncertaninty by usind Bayes Theorom.
Each node of the BN represent a random vvaiable and directed edges between nodes represent casual dependecies. The Conditional probablity Tables(CPT) are used to quanityfy the relationships.

BNS Structure for this problem

Since BN representsvariable as nodes and dependecies as directe edges, Thereforer this problem:

parent node: Disease flu or Step throst **Child NOdes:** Fever, Cough and sore throat that is the syptoms depend on the Disease



Conditional probablity Tables

Each node has a Conditional Probability Table (CPT) that defines probabilities conditioned on its parents.
Assuming equal probabilities since no prior information is provided:

$$P(\text{Flu}) = 0.5, \quad P(\text{Strep}) = 0.5$$

The conditional probabilities of symptoms given the disease are as follows:

Disease	**P(Fever = Yes)**	**P(Cough = Yes)**	**P(Sore Throat = Yes)**
Flu	0.6	0.5	0.3
Strep	0.4	0.8	0.6

How are the probabilities stored and how are they used for inference?

The probabilities are stored in Conditional Probability Tables (CPTs) for each node in the Bayesian Network. Each CPT specifies the probability of a node given its parent(s). For example:

- $P(\text{Fever}|\text{Disease})$ - $P(\text{Cough}|\text{Disease})$ - $P(\text{Sore Throat}|\text{Disease})$

These probabilities are used for inference by applying Bayes' theorem and the chain rule of probability. For example, to compute the probability of a disease given observed symptoms, we use:

$$P(\text{Disease}|\text{Symptoms}) \propto P(\text{Symptoms}|\text{Disease})P(\text{Disease})$$

where $P(\text{Symptoms}|\text{Disease})$ is computed as the product of the conditional probabilities of individual symptoms given the disease:

$$P(\text{Symptoms}|\text{Disease}) = P(\text{Fever}|\text{Disease}) \cdot P(\text{Cough}|\text{Disease}) \cdot P(\text{Sore Throat}|\text{Disease})$$

This allows us to perform reasoning and make predictions based on the given Bayesian Network.

Output

Disease	ϕ(Disease)
Disease(Flu)	0.3333
Disease(Strep)	0.6667

Problem 4: Bayesian Network

Wil use BN for spam filtering because spam filtering is a classification problem where we determin whether an email is spam or not based on its content.

Why BN

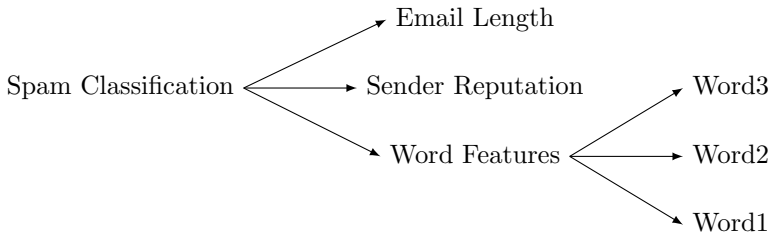
- **probabilistic Reasoning:** BN allow reasoning under unccertanity this is useful becasue the spam words may not always appear in the same context.
- **Explainability:** BN clearly shows how different words contibute to the probability of the spam
- **Handling missing data:** BN can make prediction using avaibleable data to predict missing words
- **Incremental Learning:** BN updates their knowledge as new spam mails are encountered.

How Bayesian Network work for Spam filtering

Graphical Representation of Bayesian Network for Spam Filtering

The Bayesian Network for spam filtering can be represented as a directed acyclic graph (DAG), where:

- The parent nodes represent features of the email (e.g., presence of specific words, sender reputation, etc.). - The child node represents the classification (Spam or Not Spam).



This graphical representation shows how different features contribute to the classification decision in the Bayesian Network.

How would you extend the concept of using keywords to get the essence of the next text for the document classification?

- **Latent Semantic Analysis (LSA):** Use LSA to reduce the dimensionality of the document-term matrix and extract latent topics. These topics can serve as features for classification.

- **Word Embeddings:** Utilize word embeddings such as Word2Vec, GloVe to represent words in a continuous vector space. Aggregate these embeddings through by averagin or any other metric to represent the document.
- **Topic Modeling:** Apply topic modeling techniques like Latent Dirichlet Allocation (LDA) to identify the main topics in the text. The topic distribution will be used as features for classification.
- **TF-IDF Features:** Compute Term Frequency-Inverse Document Frequency (TF-IDF) scores for words in the document. Afterwards the scores are used to to identify the most important words and classify the document based on their presence.
- **Neural Networks:** Train a neural network like RNN, Transformer among others to learn the semantic relationships between words and classify the document based on its content.

Can Bayesian Networks be used for document classification?

Yes, Bayesian Networks can be used for document classification. Here's how:

- **Feature Representation:** Represent the document features (e.g., word frequencies, presence of specific keywords, etc.) as nodes in the Bayesian Network.
- **Class Node:** Add a node representing the document class (e.g., categories like sports, politics, etc.).
- **Conditional Probabilities:** Define the conditional probability tables (CPTs) for each feature node given the class node. These probabilities can be learned from training data.
- **Inference:** Use the Bayesian Network to compute the posterior probability of each class given the observed features of a document. The class with the highest posterior probability is selected as the predicted class.

Bayesian Networks are particularly useful for document classification when the relationships between features and classes are complex and when reasoning under uncertainty is required.

Write a pseudo code

```
# Pseudo code for Bayesian Network-based Spam Filtering

# Step 1: Define the Bayesian Network structure
# Nodes: Features (e.g., Word1, Word2, Sender Reputation, Email Length),
Spam Classification
# Edges: Directed edges from features to Spam Classification
```

```
# Step 2: Initialize Conditional Probability Tables (CPTs)
# Example:
# P(Spam) = 0.4, P(Not Spam) = 0.6
# P(Word1 | Spam) = 0.8, P(Word1 | Not Spam) = 0.2
# P(Word2 | Spam) = 0.7, P(Word2 | Not Spam) = 0.3
# ...

# Step 3: Input email features
# Example: Email contains Word1 and Word2, Sender Reputation = High, Email Length = Short

# Step 4: Compute posterior probabilities using Bayes' theorem
# For each class (Spam, Not Spam):
# P(Class | Features) \propto P(Features | Class) * P(Class)
# P(Features | Class) = P(Word1 | Class) * P(Word2 | Class) * ... * P(Sender Reputation | Cl

# Step 5: Normalize probabilities
# P(Class | Features) = P(Class | Features) / Sum(P(Class | Features) for all classes)

# Step 6: Classify email
# Predicted class = argmax(P(Class | Features))

# Step 7: Output classification result
# Example: Email is classified as Spam with probability 0.85
```

Problem 5: Applications of Hidden Markov Models

The Chosen Paper and Applications

Paper: *"A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition" by Lawrence R. Rabiner (1989)*
Application: Predicting the movement of endangered species using GPS sensor data.
How HHMs are Used:

- **Hidden States:** The species' location in one of 10 geographic regions.
- **Observed States:** Discretized sensor readings (e.g., GPS coordinates mapped to 5 zones). HMMs model the transition between regions (persistence or movement) and the likelihood of sensor data given the true location.

HMM Parameters

Hidden States: 10 regions: $\{R_0, R_1, \ldots, R_9\}$.

Observed States: 5 sensor outputs: $\{O_0, O_1, \ldots, O_4\}$.

Transition Probabilities: Model persistence and adjacency:

$$P(R_i \to R_j) = \begin{cases} 0.6 & \text{if } i = j, \\ 0.2 & \text{if } j = (i \pm 1) \mod 10, \\ 0 & \text{otherwise.} \end{cases}$$

Emission Probabilities: Each region R_i primarily emits $O_{i \mod 5}$:

$$P(O_k \mid R_i) = \begin{cases} 0.7 & \text{if } k = i \mod 5, \\ 0.075 & \text{otherwise.} \end{cases}$$

Initial State Distribution: Uniform:

$$P(R_i) = 0.1 \quad \text{for all } i.$$

Results and interpretation

Results and Explanation

Output: The results output sequence ([0, 1, 0, 3, 2, 0, 4, 1, 2, 3, 0, 1]), outputs the most likely region at each time step.

Interpretation: The forward-backward algorithm computes the posterior probability

$$P(\text{Region}_t \mid \text{Observations}_{1:T})$$

by combining:

- **Forward messages (α)**: Likelihood of observations up to t and being in region R_i.
- **Backward messages (β)**: Likelihood of future observations given R_i.

The result smooths over the entire sequence, leveraging past and future data to infer intermediate states . For example, if the sensor data is noisy, the algorithm corrects it by considering the species' tendency to stay in adjacent regions.

Problem 6: Implementing a Simple Kalman Filter

Kalman Filter is a rucurvsive Bayesian filter that provides the optimal estimate of the hidden state of a linear dynamical systemn in the presence of Gaussian noise.

Kalman Math fund.

A linear dynamical system evolves in discrete time according to the following equations:

(A) State Equation (Process Model)

$$x_{k+1} = Ax_k + w_k$$

where:

- x_k is the hidden state (e.g., position and velocity of a moving object).
- A is the state transition matrix, which determines how the state evolves.
- w_k is the process noise, assumed to be Gaussian with mean zero and covariance Q:
$$w_k \sim \mathcal{N}(0, Q)$$

(B) Measurement Equation (Observation Model)

$$z_k = Hx_k + v_k$$

where:

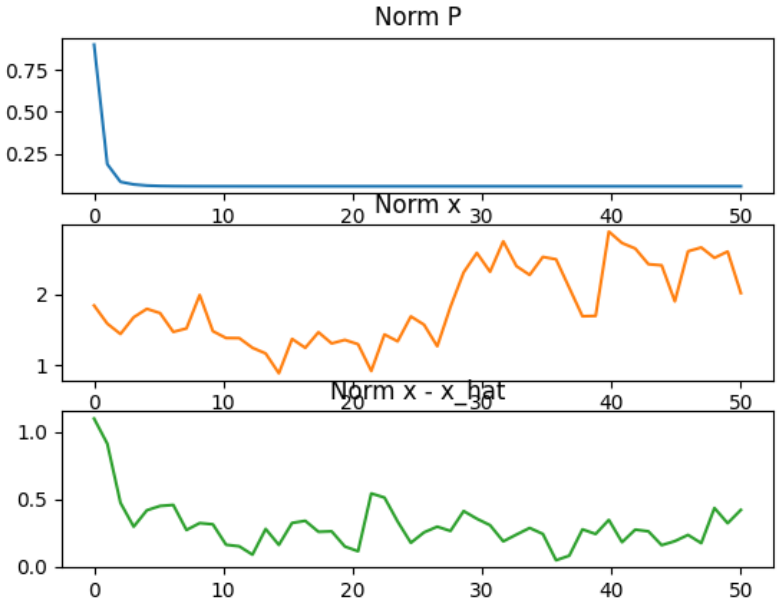
- z_k is the observation (measurement) at time k.
- H is the observation matrix, which maps the hidden state to the measurement space.
- v_k is the measurement noise, assumed to be Gaussian with mean zero and covariance R:
$$v_k \sim \mathcal{N}(0, R)$$

The goal of the Kalman Filter is to estimate x_k, given the noisy measurements z_k.

Questions before implimentation

- **What are the connenection between Kalman Filter and HMM?what are the similarites and most important differences**
  - **Connection:** Both Kalman Filters and Hidden Markov Models (HMMs) are probabilistic models used for state estimation in dynamic systems. They share the following similarities and differences:
    - **Similarities:**
      - Both are used for sequential data and involve hidden states.
      - Both rely on probabilistic reasoning and Bayesian inference.
      - Both use transition models to describe how states evolve over time.
    - **Differences:**
      - Kalman Filters assume linear dynamics and Gaussian noise, while HMMs can handle discrete states and non-Gaussian noise.
      - Kalman Filters provide continuous state estimates, whereas HMMs focus on discrete state sequences.
      - Kalman Filters are computationally efficient for linear systems, while HMMs require more complex algorithms like the Forward-Backward algorithm for inference.
- **What is the goal of Kalman Filter**
  - The goal of the Kalman Filter is to estimate the hidden state x_k of a linear dynamical system at each time step k, given noisy measurements z_k. This involves combining prior knowledge about the system's dynamics (process model) with the observed data (measurement model) to produce an optimal estimate of the state. The Kalman Filter achieves this by minimizing the mean squared error of the state estimate.

Implimentation output



Questions after implimentation

- **Describe the significance/differences between the time update and measurement update.**
  - **Time Update (Prediction Step):** This step predicts the state of the system at the next time step based on the current state estimate and the process model. It involves projecting the state and covariance forward in time. The time update is crucial for estimating the system's state before incorporating new measurements.
  - **Measurement Update (Correction Step):** This step updates the predicted state using the new measurement. It adjusts the state estimate and covariance based on the difference between the predicted measurement and the actual measurement. The measurement update ensures that the state estimate is corrected using the latest observed data.
- **What is the significance of the measurement z having different dimensions to the signal vlues X**

  - Measurements (z) often observe a subset or linear combination of the state (x). The matrix C maps the state to the measurement space, enabling partial observations.

- **From the plot above, what can you say about shape of the curve P with respect to number of iteration? What does it show us about estimate of P and the Kalman Filter gain matrix K? In two sentences describe the meeningd of P and K matrices**

  - P represents the covariance of the state estimate, which indicates the uncertainty in the estimate. As the number of iterations increases, P typically decreases and seems to be stabilizing this indicastes convergence to the steady-state error covariance showing that the estimate becomes more certain over time.
  - K, the Kalman Gain, determines how much weight is given to the measurement versus the prediction. As P decreases, K also decreases, indicating that the filter relies more on the prediction and less on the measurement. A stable P indicates an optimal K.

- **Proide the same analysis for the plot of the norm of x -x_hat**

  - The plot of the norm of $x - \hat{x}$ typically shows a decreasing trend over iterations, indicating that the estimate $\hat{x}$ is converging to the true state x. This demonstrates the effectiveness of the Kalman Filter in reducing the estimation error over time. The norm of $x - \hat{x}$ represents the magnitude of the estimation error, and its decrease signifies improved accuracy of the state estimate.

Problem 7

Predicting the sales of its product over the next year based on the historical sales data.

Best Model to perform the task?

1. **Hidden Markov Models**
    - Is a probabilisic model that assumes hiden states for sales levels such as low, medium orhigh and infere from the observed data such as advaerticememtn and pricing.
    - It also has observations that measure variables like season, price and advertising spend.
    - It also assumes the Markov propery that the current stat depends only on the previous state.
        - However HMM cannot hanndle multiple dependecies, limited to the seasonal trend modeling and its only limited to one hidden variable
2. **Dynamic Bayesian Network**
    - Generalizes HMM by allowing more flexible dependecies and multi-variable state represntation insstead of a single hidden variable.
    - Unlike HMM it can explicitly model the seasonal trends.
        - Howver, its more complex and computationally expensive
3. **Markov Decision Process**
    - Its best in making decision on how much to spend on advertisin to maximize the future sales.
    - It optimized the long term revenue also helps in deciding how much to invest in marketing, and accounts for uncertinity.
        - However it requires defining awards which might be comples and computationally expensive for large state spaces.
4. **Kalman Filter**
    - Kalman Filter is best for real time tracking of the sales in the noisy sales.
    - Its also computernally efficienen and can make real-time prediction.
        - However,it assumes linear relationship and struggles with non-linear dependecies.
        - It cannot model easily external variables like advertising.

Best model fit to the task and why?

Since sales are influenced by mulitple varibles and factors such as price, season, adverticing and others, DBN is the better choice as i allows modelling for multiple relationships.

Why DBN is the best fit

- Multiple variable: DBN facotrs in and can easily model multiple varibles such as advertising, price and season that other Models cannont explicitly model.
- Long-term dependecies: DBN can model how external variables like advertising how it affects the sales over time.
- Better Accuracy: DBN captures the real-world dependecies.

Implimentation of DBN

- Defining the sales varibales: Sales, Adverts, Price and season
- Construct Bayesian Network at each time step
- Define the Conditional probablities.
- Use pomegranes to impliment DBN
- For future sales prediction use forward sampling.

Problem 8: First-Order Markov Chain Model

First order markov chain is the probabilistic model that assumes that the probability of transisioning to a new state depends only the current state and not on any past states.

Math for Markov Chains

A Markov chain consists of:

- A set of states $S = \{s_1, s_2, \ldots, s_n\}$, where each s_i represents a possible state.
  - In this task we can defin two states:
    - s_1 - no purchase
    - s_2 - purchase
- A transition probability matrix P, where $P_{ij} = P(s_j|s_i)$ represents the probability of transitioning from state s_i to state s_j.
- An initial state distribution $\pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$, where $\pi_i = P(s_i)$ is the probability of starting in state s_i.

Forward and Backward aligorithm

To predict the probability of a customer making a purchase at time t, we use the two key aligorithm

Forward aligorithm

It computes the probability of observind a sequence of the use activited up to time t, given the state at t.

$$\alpha_t(i) = P(o_1, o_2, \ldots, o_t, s_t = i) = \sum_{j=1}^{N} \alpha_{t-1}(j) P(s_t = i|s_{t-1} = j) P(o_t|s_t = i)$$

where:

- $o_1, o_2, \ldots, o_t$ are the observations up to time t,
- $s_t = i$ is the state at time t,
- $P(s_t = i|s_{t-1} = j)$ is the transition probability from state j to state i,
- $P(o_t|s_t = i)$ is the observation likelihood for state i at time t.

Backward aligorithm

It computes the probability of observing a sequence of user activities from time $t+1$ to the end, given the state at t.

$$\beta_t(i) = P(o_{t+1}, o_{t+2}, \ldots, o_T | s_t = i) = \sum_{j=1}^{N} \beta_{t+1}(j) P(s_{t+1} = j | s_t = i) P(o_{t+1} | s_{t+1} = j)$$

where:

- $o_{t+1}, o_{t+2}, \ldots, o_T$ are the observations from time $t+1$ to the end,
- $s_t = i$ is the state at time t,
- $P(s_{t+1} = j | s_t = i)$ is the transition probability from state i to state j,
- $P(o_{t+1} | s_{t+1} = j)$ is the observation likelihood for state j at time $t+1$.

Compute Posteriori probablities

To compute the posterior probabilities of being in a particular state at time t, given the observations up to time T, we combine the forward and backward probabilities:

$$P(s_t = i | o_1, o_2, \ldots, o_T) = \frac{\alpha_t(i) \beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j) \beta_t(j)}$$

where:

- $\alpha_t(i)$ is the forward probability for state i at time t,
- $\beta_t(i)$ is the backward probability for state i at time t,
- The denominator $\sum_{j=1}^{N} \alpha_t(j) \beta_t(j)$ ensures normalization.

This posterior probability represents the likelihood of being in state i at time t, given all observations.

Problem 9: Parameter Learning for Gaussian Mixture Models

GMM are probabilistic models that assume that the data is generetaed from the mixture of multiple Gaussian distributions. The models are useful for clustering and density estimation whtn the data exhibits subgroups that can be approximated by Gaussian Distribution.

GMM Fundamentals

Mathematical Formulation

A GMM assumes the dataset is generated from K different Gaussian distributions, each with its own mean vector μ_k, covariance matrix Σ_k, and mixing coefficient π_k. The probability density function for a GMM is given by:

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)$$

where:

- $\mathcal{N}(x|\mu_k, \Sigma_k)$ is the Gaussian distribution with mean μ_k and covariance Σ_k,
- π_k is the mixing coefficient for the k-th Gaussian, such that $\sum_{k=1}^{K} \pi_k = 1$ and $\pi_k \geq 0$.

Parameter Estimation using Expectation-Maximization (EM)

The EM algorithm is used to estimate the parameters $\{\pi_k, \mu_k, \Sigma_k\}$ of the GMM when we don't know which Gaussian components each data point belongs to. Since the data points in the Gaussian are latent(hidden), EM is used to iteratively estimate both the parameters and the hidden labels. It consists of two steps:

1. **E-step:** Compute the responsibility γ_{nk}, which represents the probability that data point x_n belongs to the k-th Gaussian:

$$\gamma_{nk} = \frac{\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_n|\mu_j, \Sigma_j)}$$

2. **M-step:** Update the parameters $\{\pi_k, \mu_k, \Sigma_k\}$ using the responsibilities:

$$\pi_k = \frac{1}{N} \sum_{n=1}^{N} \gamma_{nk}, \quad \mu_k = \frac{\sum_{n=1}^{N} \gamma_{nk} x_n}{\sum_{n=1}^{N} \gamma_{nk}}, \quad \Sigma_k = \frac{\sum_{n=1}^{N} \gamma_{nk} (x_n - \mu_k)(x_n - \mu_k)^\top}{\sum_{n=1}^{N} \gamma_{nk}}$$

   where N is the total number of data points.

How EM aligorithm works for parameter Learning in GMMS

The EM algorithm alternates between the E-step and M-step until convergence. Here's how it works:

1. **Initialization**: Start with initial guesses for the parameters $\{\pi_k, \mu_k, \Sigma_k\}$.

2. **E-step**: Compute the responsibilities γ_{nk} for each data point x_n and each Gaussian component k:

$$\gamma_{nk} = \frac{\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_n|\mu_j, \Sigma_j)}$$

This step assigns a soft cluster membership to each data point.

3. **M-step**: Update the parameters $\{\pi_k, \mu_k, \Sigma_k\}$ using the responsibilities:

$$\pi_k = \frac{1}{N} \sum_{n=1}^{N} \gamma_{nk}, \quad \mu_k = \frac{\sum_{n=1}^{N} \gamma_{nk} x_n}{\sum_{n=1}^{N} \gamma_{nk}}, \quad \Sigma_k = \frac{\sum_{n=1}^{N} \gamma_{nk} (x_n - \mu_k)(x_n - \mu_k)^\top}{\sum_{n=1}^{N} \gamma_{nk}}$$

4. **Convergence**: Repeat the E-step and M-step until the log-likelihood of the data under the model stops changing significantly.

The EM algorithm ensures that the log-likelihood increases or remains constant at each iteration, leading to a locally optimal solution for the GMM parameters.

How do you determine Convergence

Convergence in the EM algorithm determined through monitoring the change in the log-likelihood of the data under the model. The algorithm stops when the change in log-likelihood between successive iterations falls below a predefined threshold ϵ, or after a maximum number of iterations is reached. Mathematically:

$$\text{Convergence Criterion: } |\mathcal{L}^{(t)} - \mathcal{L}^{(t-1)}| < \epsilon$$

where:

- $\mathcal{L}^{(t)}$ is the log-likelihood at iteration t,
- ϵ is a small positive value (e.g., 10^{-6}) that defines the tolerance for convergence.

The impact of Initialization on the final parameters

The initialization of parameters in the EM algorithm impact the final parameters and the convergence of the algorithm. Poor initialization may lead to convergence to a local optimum that does not represent the true underlying distribution of the data. Common initialization strategies include:

- **Random Initialization:** Randomly assign initial values to the parameters $\{\pi_k, \mu_k, \Sigma_k\}$. This approach is simple but may lead to poor convergence if the initial values are far from the true parameters.
- **K-Means Clustering:** Use the centroids and cluster assignments from K-Means as the initial means μ_k and mixing coefficients π_k. Covariance matrices Σ_k can be initialized based on the spread of points in each cluster. This method often provides a good starting point.
- **Hierarchical Clustering:** Use hierarchical clustering to determine initial cluster assignments and compute initial parameters based on these assignments.
- **Multiple Restarts:** Run the EM algorithm multiple times with different initializations and select the solution with the highest log-likelihood.

Choosing an appropriate initialization strategy impacts the aligorithm by improving the robustness and accuracy of the EM algorithm for GMMs.

How does the number of components K affect the model fit

The number of components K in a GMM can affects the model fit as shown below:

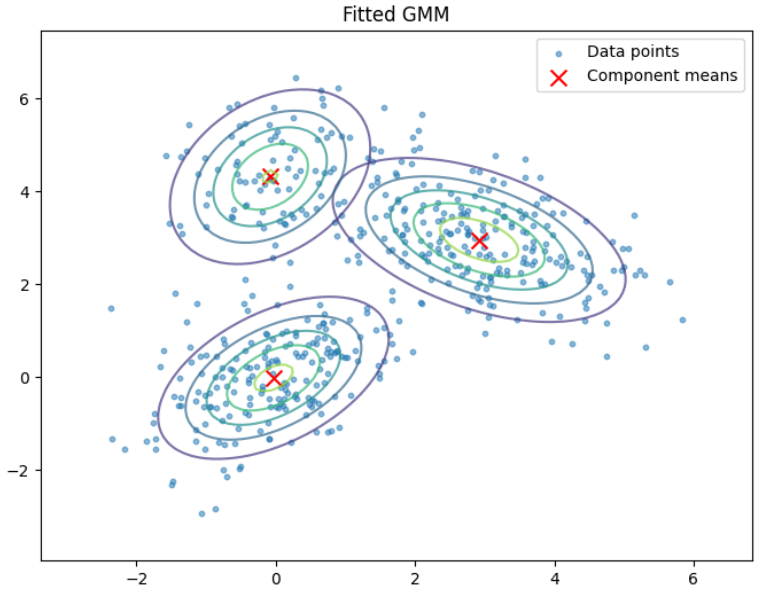
- **Underfitting:** If K is too small, the model may not capture the complexity of the data, leading to underfitting.
- **Overfitting:** If K is too large, the model may overfit the data by capturing noise or minor variations as separate components. This can lead to poor generalization to new data.
- **Model Selection:** The optimal K can be determined using model selection criteria such as:
  - **Bayesian Information Criterion (BIC):** Penalizes the model complexity to avoid overfitting.
  - **Akaike Information Criterion (AIC):** Balances model fit and complexity.
  - **Cross-Validation:** Splits the data into training and validation sets to evaluate the model's performance for different K.

Choosing the right K ensures a balance between model complexity and generalization, leading to better performance on unseen data.

Applications of GMM

GMMs are widely used in:

- Clustering: Grouping data into K clusters based on the Gaussian components.
- Density Estimation: Modeling the probability distribution of data.
- Anomaly Detection: Identifying data points with low likelihood under the GMM.



Problem 10: Hidden Markov Model

HMM is a probabilistic model used to describe systesm that undergo changes over time in a way that depends on hidden states. In the problem given the patient condition is hidden thatis the viral and bacteria but we can observe the temperature that is High or nomral tha tgives us the inderect evidence about the condition.

Key Components of an HMM

1. **Hidden States ($\mathcal{S}$)**: These are the actual states of the system, which we cannot directly observe. For this problem, the two hidden states are:
   - **"Viral" (V)**: Patient has a viral infection
   - **"Bacterial" (B)**: Patient has a bacterial infection
2. **Observations ($\mathcal{O}$)**: These are the observable variables dependent on the hidden states. For this problem, the observations are the patient's **temperature**, which can be:
   - **"Normal" (N)**: Patient's temperature is normal.
   - **"High" (H)**: Patient has a high temperature.
3. **Initial Probabilities (π)**: This represents the probability of the patient being in each state at **time** $t = 0$ (Day 1).

$$\pi(V) = 0.7, \quad \pi(B) = 0.3$$

4. **Transition Probabilities (A)**: These represent the probability of transitioning from one state to another between consecutive time steps.

$$A = \begin{bmatrix} P(V \to V) & P(V \to B) \\ P(B \to V) & P(B \to B) \end{bmatrix} = \begin{bmatrix} 0.8 & 0.2 \\ 0.4 & 0.6 \end{bmatrix}$$

5. **Emission Probabilities (B)**: These represent the probability of observing a symptom **given the hidden state**.

$$B = \begin{bmatrix} P(N|V) & P(H|V) \\ P(N|B) & P(H|B) \end{bmatrix} = \begin{bmatrix} 0.6 & 0.4 \\ 0.3 & 0.7 \end{bmatrix}$$

We need to **compute the probability** of observing the sequence **(Normal, High, High)** over three days using two algorithms:

1. **Forward Algorithm (α)**.
2. **Backward Algorithm (β)**.

1. Forward Algorithm (α)

The **Forward Algorithm** is used to compute the probability of an observation sequence **efficiently** using **dynamic programming**.

The forward probability $\alpha_t(s)$ represents the probability of being in state s at time t given the observed sequence up to t.

1. **Initialization** ($t = 0$):

$$\alpha_0(s) = \pi(s)B(s, O_0)$$

2. **Recursion** ($t > 0$):

$$\alpha_t(s) = \sum_{s'} \alpha_{t-1}(s')A(s', s)B(s, O_t)$$

3. **Termination**:

$$P(O) = \sum_{s} \alpha_T(s)$$

2. Backward Algorithm (β)

The **Backward Algorithm** is similar but works in **reverse order**, computing the probability of the remaining observations given a state at time t.

1. **Initialization** ($t = T$):

$$\beta_T(s) = 1$$

2. **Recursion** ($t < T$):

$$\beta_t(s) = \sum_{s'} A(s, s')B(s', O_{t+1})\beta_{t+1}(s')$$

3. **Compute** $P(O)$ **using** β:

$$P(O) = \sum_{s} \pi(s)B(s, O_0)\beta_0(s)$$

Generally

- **Forward algorithm** computes the probability **incrementally**.
- **Backward algorithm** computes the probability **from the end**.
- Both methods should **yield the same result**.

Steps for Implementation

1. Define the transition matrix A and emission matrix B.
2. Implement the forward algorithm step-by-step.
3. Implement the backward algorithm step-by-step.
4. Verify that both methods produce the same probability.
5. Print intermediate steps for debugging.

How are the probabilities stored and how are they used for inferer-ence? Please Explain

The probabilities are stored in matrices or tables, such as the transition matrix A, emission matrix B, and initial probabilities π. These matrices are used for inference by applying Forward and Backward algorithms to compute the likelihood of observation sequences or to determine the most probable sequence of hidden states.

From the above explanation how i came up with the solution we concluded that forward aligorithm computes probability incrementally and the backward aligorithm from the end.

These probabilities enables us to perform the following: 1. Predicting the likelihood of a sequence of observations. 2. Decoding the most probable sequence of hidden states using the Viterbi algorithm. 3. Estimating model parameters using the Baum-Welch algorithm.

The implimentation output

Forward Algorithm Table

Time (t)	**Viral** (α[**Viral**][t])	**Bacterial** (α[**Bacterial**][t])
0 (Normal)	0.4200	0.0900
1 (High)	0.1488	0.0966
2 (High)	0.0631	0.0614

Probability of the observation sequence (N, H, H): 0.124476

Backward Algorithm Table

Time (t)	**Viral** (β[**Viral**][t])	**Bacterial** (β[**Bacterial**][t])
2 (High)	1.0000	1.0000
1 (High)	0.4600	0.5800
0 (Normal)	0.2284	0.3172

Backward Probability: 0.124476

Problem 11: Maximum A Posteriori estimation

Maths of MAP estimation

MAPis away of estimating the probablity distribution by incoprating the prior beliefs, by that it means given a set of observed data d, MAP estimation chooses the class C that maximizes the Posteriori probablity.

$$P(C|d) = \frac{P(d|C)P(C)}{P(d)}$$

- $P(C|d)$ - is the posterior probability of class C given document d.
- $P(d|C)$ - is the likelihood of document d occuring given class C.
- $P(C)$ - is the prior probability of class C.
- $P(d)$ - is the evidence or the probability of document d.

Since $P(d)$ is the same across all the classes we igone it and compute:

$$\arg\max_C P(d|C)P(C)$$

Therefore we need to calculate:

- $P(C)$ - prior probablity of each class
- $P(d|C)$ - The likelihood of the document given the class

MAP and Niave Bayes relationship

This task is similar to NB classifiere since it assumes:

- **Conditional Independence** the words in the document are Independent given the class.
- **MultinomialModel** the probablity of the document is the product of indivisdual word probablities.

Applying the Naive Bayes (NB) assumption, the likelihood $P(d|C)$ is given by:

$$P(d|C) = \prod_{i=1}^{n} P(w_i|C)$$

where:

- w_i represents the i-th word in the document d,
- $P(w_i|C)$ is the probability of word w_i given class C,
- n is the total number of words in the document.

Takind the log to avaoid underflow of issues:

$$\log P(d|C) = \sum_{i=1}^{n} \log P(w_i|C)$$

Thus, the MAP estimation becomes:

$$\arg\max_{C} \left(\log P(C) + \sum_{i=1}^{n} \log P(w_i|C) \right)$$

Multinomial Distribution and Text classification

- document represents a bag o f words i.e word frequencies
- probability of a document given the class is the product of word occuracne probablity
- Each word's probability in a class is estimated as:

$$P(w_i|C) = \frac{\text{count}(w_i, C) + \alpha}{\text{count}(C) + \alpha \cdot V}$$

  where:

  - $\text{count}(w_i, C)$ is the number of times word w_i appears in documents of class C,
  - $\text{count}(C)$ is the total number of words in documents of class C,
  - α is the smoothing parameter (Laplace smoothing),
  - V is the size of the vocabulary.

Implimentation approach

1. **Preprocessing Text Data**
   - convert text to lowercase
   - remove punctuation and stop words
   - tokenize into words
   - represent documnt as word count vectors
2. **Train the MAP-Based Classifier**
   - Compute class priors: $P(C) = \frac{countofdoccsinC}{totaldocs}$
   - Compute word probability for each class using laplace smoothing
3. **Predict the new document's class**

- Compute the posterior probability for each class using:

$$\log P(C|d) = \log P(C) + \sum_{i=1}^{n} \log P(w_i|C)$$

- Select the class C with the highest posterior probability:

$$C_{\text{predicted}} = \arg\max_{C} \left(\log P(C) + \sum_{i=1}^{n} \log P(w_i|C) \right)$$

The solution is in MAP.py script