18662 Homework 3

Due date: March 31th, 5 pm EST

Problem 1: Value Iteration for Markov Decision Processes

(10 points)

In this problem, you will implement the value iteration algorithm for the Frozen Lake environment from HW 1. Use your Homework 1 code to set up the environment, make actions, and observe states (if you did not finish this problem in HW 1, please see the recitation recording from 2/24.) For a good review, please consult the MDP slides on Canvas.

Problem specifications:

- Frozen Lake environment can be either 4x4 or 8x8.
- The avatar can move up, down, left, or right.
- Reward associated with each state:
  - Goal location: +1
  - Hole: 0
  - All other locations: 0
- Assume that the rightwards action has probability 0.3, down has probability 0.3, up has probability 0.2, and left has probability 0.1.
- Use a reasonable value for gamma. Explain why you think the value you choose
- is a reasonable one.

Deliverables:

- A Value iteration function that outputs the optimal value function and optimal policy.
- A document containing the value function and policy that results from value iteration on the 4x4 environment (these can simply be copied and pasted arrays from the code output).
- A Python file that, when run on any 4x4 or 8x8 Frozen Lake environment, shows the avatar traveling from its start location to the goal location (optimally) according to the value iteration results. The start and end locations should be defined at the top and the rest of the code should still work if these locations are changed.

Problem 2: SARSA for Markov Decision Processes
(10 points)

As an alternative to Value Iteration in the previous problem, there exists another algorithm, SARSA, first proposed by Richard Sutton and detailed here: http://incompleteideas.net/book/ebook/node64.html. In this problem, you will implement SARSA on the Frozen Lake environment and compare the algorithm to Value Iteration from the previous problem.

Problem specifications:

- Frozen Lake environment can be either 4x4 or 8x8.
- The avatar can move up, down, left, or right.
- Reward associated with each state:
    - Goal location: +1
    - Hole: 0
    - All other locations: 0
- As part of SARSA, you will have to implement an epsilon-greedy helper function to determine the next actions. You should use a value of `epsilon = 0.1`.
- Use a reasonable value for gamma.

Deliverables:

- A SARSA function that outputs the optimal value function and optimal policy.
- A document containing the value function and policy that results from SARSA on the 4x4 environment (these can simply be copied and pasted arrays from the code output).
    - Describe the difference between the optimal value functions and optimal policies from the previous problem (value iteration) and SARSA. What are the differences and why?
- A Python file that, when run on any 4x4 or 8x8 Frozen Lake environment, shows the avatar traveling from its start location to the goal location (optimally) according to the value iteration results. The start and end locations should be defined at the top and the rest of the code should work if these locations are changed.

Why should one use SARSA instead of the Value Iteration algorithm? You may have noticed that in SARSA we do not assume we know the state transition probabilities as in Value Iteration. In the case that we do not have this information, Value Iteration would not work. **In fact, we call this model-free reinforcement learning.**

Problem 3: Bayesian Network

(10 points)

A patient comes to a doctor with a cough, fever, and sore throat. The doctor suspects the patient might have either the flu or strep throat. The doctor also knows that patients with the flu have a 60% chance of having a fever, while patients with strep throat have a 40% chance of having a fever. Additionally, patients with the flu have a 50% chance of having a cough, while patients with strep throat have an 80% chance of having a cough. Finally, patients with the flu have a 30% chance of having a sore throat, while patients with strep throat have a 60% chance of having a sore throat.

Implement a Bayesian network in python to find the probability that the patient has strep throat (you may use pgmpy library).

The dependencies are as follows:

- The probability of having a fever depends on the disease (flu or strep throat).
- The probability of having a cough depends on the disease.
- The probability of having a sore throat depends on the disease.
- The probability of having the disease (flu or strep throat) depends on the symptoms.

Add comments to your code at every step (this will be used for grading).
State clearly:

- How you came up with the solution (A figure for the Bayesian network should be added with a good explanation).
- How are the probabilities stored and how are they used for inference? Please explain.

Problem 4: Bayesian Network

(10 points)

Words/ key words in the text often tell us a lot about the type of text. For example, Email containing text such as “get rich fast” would most likely be a fraud email and must be marked as spam. Explain how you can use Bayesian Networks for spam filtering and write a python program for spam filtering. This question expects you to think beyond just implementing the Bayesian network; you must create your own conditional probability table using the data and the spam keywords.
The input to your program could be any text, the output must be the probability of that text being spam.

For example,

You could use the email text in the following dataset for working on your model or use text from elsewhere (but clearly mention what is being used in your code):
https://www.kaggle.com/datasets/venky73/spam-mails-dataset

Make sure you have a function for preprocessing the text (stop words removal, de-capitalization, etc.)

Example keywords to look for:

spam_keywords = ['Viagra', 'Cialis', 'Levitra', 'Vicodin', 'Xanax', 'Ambien', 'OxyContin', 'Percocet', 'Valium', 'Tramadol', 'Adderall', 'Ritalin', 'Modafinil', 'Phentermine', 'Ativan', 'Klonopin', 'Zoloft', 'Prozac', 'Paxil', 'Wellbutrin', 'Celexa', 'Lexapro', 'Effexor', 'Zyprexa', 'Abilify', 'Seroquel', 'Lamictal', 'Depakote', 'Lithium', 'Nigerian', 'Lottery', 'Free', 'Discount', 'Limited time offer', 'Money-back guarantee', 'Investment opportunity', 'Secret', 'Unsubscribe', 'Click here', 'Double your money', 'Cash','Urgent', 'Get rich quick', 'Work from home', 'Multi-level marketing', 'Pyramid scheme', 'Enlarge', 'Buy now', 'Online pharmacy', 'Weight loss', 'Casino', 'Credit report', 'Debt relief']

This is not an exhaustive list. You can use fuzzy logic to improve your model to recognize more such keywords.

Fuzzy logic is a mathematical approach that deals with uncertainty and imprecision in natural language processing (NLP). It allows reasoning with uncertain or ambiguous

variables and can be applied in various NLP tasks such as sentiment analysis and text classification.

In the current scenario, say you have the word “Urgent” in your list of keywords for spam filtering, but not “Immediate”. And you have the following emails:

- One containing the sentence “needs Urgent action/attention”
- Other containing the sentence “needs Immediate action/attention”

The model would mark only the first email as spam since it would have a higher probability for “urgent” being spam. However, If we had a way to identify that urgent and immediate are similar, our model would certainly do better. Fuzzy logic can help identify this similarity.

You can use libraries such as nltk and spacy that provide fuzzy string-matching capability.

How would you extend the concept of using keywords to get the essence of the text for document classification? Can Bayesian Networks be used for document classification?

If yes, explain how this can be done and write a pseudo code for it
Else, explain clearly why it is not possible to use Bayesian Networks in this case.

Problem 5: Applications of Hidden Markov Models
(10 points)

Hidden Markov Models (HMMs) can be used to model sequences of events with hidden causes. As described in the textbook on page 576, an example of such a model could be whether or not someone brings an umbrella with them on a particular day, where the sequence of events is a Boolean list of whether or not an umbrella was brought with them and the hidden variable is whether or not it rained on that day. The HMMs can encode natural tendencies, (such as, for example, that weather tends to persist and that therefore people are more likely to bring an umbrella on any particular day if it rained on the previous day), with transition probabilities. We use the forward-backward algorithm to determine the *most-likely hidden state* given these transition probabilities.

In this problem, you are to choose a conference or journal paper that uses HMMs for some sort of application. Design an HMM, including the observed/hidden states and a set of transition probabilities for this application (you may choose to loosely follow or exactly follow the model from the paper of choice). Implement a ≥10-state HMM and implement the forward-backward algorithm on it to determine the most likely hidden state for any intermediate time step.

Some possible applications:

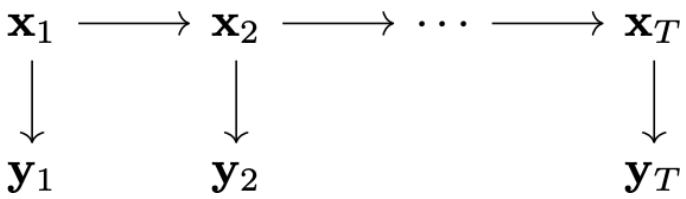
- Speech processing
- Autofill for typing
- Filling in incomplete data sets
- Predicting the movement of endangered species
- Many others!

A word of advice: start with the simple HMM problem in the textbook on page 576 and implement the forward-backward algorithm for it. Then, find an interesting paper and make any necessary changes to make it fit your application. Hints are given on the next page.

Deliverables:

- A description of the paper you chose and how HMMs can be used for the application of choice.
- A write-up of the HMM parameters (the observed/hidden variables and the transition probabilities you use in the algorithm).
- An implementation of the forward-backward algorithm in Python.
- The most likely hidden state for any intermediate time step (the output of the forward-backward algorithm). Include also a written explanation of the meaning of this result.

Hints:



- The above diagram is a visualization of the HMM where x_i are the hidden variables and y_i are the observed states. The variable T above must be ≥10 in your implementation.
- In order to implement the forward-backward algorithm, you will need to first establish (either using your best judgment or from the paper if available):
    - The value of x_1. If x is binary and $x_1 = 0$, this can be encoded as $p(x_1 = 1) = 0$.
    - The posterior probability of x_i given x_{i-1}, i.e., $p(x_i|x_{i-1})$.
    - The posterior probability of y_i given any particular observation of x_i. In the case that x is binary, this would be $p(y_i|x_i = 0)$ and $p(y_i|x_i = 1)$. **It is crucial to an HMM problem that y_i has multiple possible states (each with an associated probability) for *each* value of the observed variable, for example,**

      $$p(y_i = 75|x_i = 1) = 1/3$$
      $$p(y_i = 15|x_i = 1) = 2/3.$$
- In the forward-backward algorithm implementation, β_i are the backward messages and α_i are the forward messages. We say that $\beta_T = 1$ and $\alpha_1(x_1) = p(x_1)p(y_1|x_1)$. The rest of the forward and backward messages can be determined using Bayes' rule and the Law of Total Probability (see the handwritten notes on Canvas for these definitions!).

Problem 6: Implementing a Simple Kalman Filter
(10 points)

In this problem you will implement a simple Kalman Filter (with lots of guidance). To start, review Section 15.4 in Russell & Norvig and read the following guide: http://bilgin.esme.org/BitsAndBytes/KalmanFilterforDummies.

Questions to answer before implementing (submit your answers in a document):

- What is the connection between Kalman Filters and HMMs? What are the similarities and what are the most important differences? Section 15.4 in Russell & Norvig will help with this question.
- What is the goal of Kalman Filtering?

In the implementation, we assume we have some signal x, that has the following discrete-time model with Gaussian process noise w_k (the mean and variance of this noise is left unspecified):

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

Process noise results from natural stochasticity in the system we are measuring. The above linear system can represent any linear process–in fact, this is how robot dynamics and signals are represented. We also have measurements of this signal, z_k, which is related to x_k through another linear system. Our method of measuring the signal may also have natural stochasticity, which we represent as v_k (also Gaussian noise with unspecified parameters). The measurement model is defined as follows:

$$z_{k+1} = Hz_k + v_k$$

You will now implement the steps detailed in the guide linked above, with the help of the starter code provided in the file kalman_filter_prob.py. Follow the flow chart in Step 3 in the guide. The model parameters are given in .mat files and are loaded for you. You will need to use numpy tools to perform the necessary matrix multiplications.

Questions to answer after implementing (submit your answers in a document):

- Describe the significance/differences between the Time Update and Measurement Update steps in Step 3 of the guide.
- What is the significance of the measurement z having a different dimension to the signal values x?
- Plot the norm of the error covariance matrix P with respect to the number of iterations elapsed. What can you say about the shape of the curve? What does it show us about our estimate of P and the Kalman Filter gain matrix K? In two sentences describe the meanings of the P and K matrices.
- Provide a similar analysis for the plot of the norm of x - x_hat.

Problem 7:

(10 points)

A company wants to predict the sales of its products over the next year based on historical sales data. The company knows that sales are influenced by various factors, such as advertising spending, price, and seasonal trends. Using probabilistic reasoning over time, how can the company build a model that takes these factors into account and predicts future sales? Some possible models to consider would be Dynamic Bayesian Network or HMM.

a) Which model can be used in this case? (discuss different possibilities and choose the best option giving a justification of why it is better)
b) Why is it a good fit for the scenario?
c) Implement the chosen model in python (state clearly in your code the states of the model)[Choose the correct relationships, probabilities can be assumed]

You could use libraries like pomegranate if you wish to for this task.

Problem 8: First-Order Markov Chain Model
(10 points)

You now wish to analyze whether a customer will make a purchase or not. You can use any dataset of your choice that contains information about customer purchase histories and online stores. Using a first-order Markov Chain Model, design a probabilistic model to predict the probability of a customer making a purchase at a given time, based on their past purchase history and other relevant variables. Your implementation must contain:

- The forward algorithm to compute alpha
- The backward algorithm to compute beta
- Function for calculating the posterior probabilities

Useful Theory from Russell and Norvig:

In a first-order Markov Chain Model, the probability of transitioning to a new state depends only on the current state, and not on any past states. We can use this model to predict the probability of a customer making a purchase at a given time, based on their past purchase history and other relevant variables. Here are the formulas for alpha, beta, and posterior probabilities:

Forward Algorithm (to compute α):
$\alpha_{t(i)} = P(O_1, O_2, ..., O_t, X_t = i|\theta) =$ [sum over j from 1 to N] $\alpha_{t-1}(j) * P(X_t = i|X_{t-1} = j, \theta) * P(O_t|X_t = i, \theta)$
where:
$O_1, O_2, ..., O_t$ are the observed data up to time t
X_t is the hidden state at time t (e.g. purchase or not purchase)
N is the number of possible states
θ is the set of model parameters

Backward Algorithm (to compute β):
$\beta_{t(i)} = P(O_{t+1}, O_{t+2}, ..., O_T|X_t = i, \theta)$
$=$ [sum over j from 1 to N] $P(X_{t+1} = j|X_t = i, \theta) * P(O_{t+1}|X_{t+1} = j, theta) * \beta_{t+1}(j)$
where:
T is the total number of time steps
j is the next state at time t+1

Function for calculating the posterior probabilities:
$\gamma_{t(i)} = P(X_t = i|O_1, O_2, ..., O_T, \theta)$
$= (\alpha_{t(i)} * \beta_{t(i)})/$[sum over j from 1 to N]$\alpha_{t(j)} * \beta_{t(j)}$
where:
$\gamma_{t(i)}$ is the posterior probability of being in state i at time t, given the observed data and model parameters

Problem 9: Parameter Learning for Gaussian Mixture Models

In this problem, you will implement the Expectation-Maximization (EM) algorithm for learning the parameters of a Gaussian Mixture Model (GMM). GMMs are probabilistic models that assume data points are generated from a mixture of several Gaussian distributions.

Problem specifications:

- You will implement the EM algorithm to learn the parameters of a GMM with a given number of components K.
- The parameters to learn are:
  - Mixing coefficients (weights) for each Gaussian component
  - Mean vectors for each Gaussian component
  - Covariance matrices for each Gaussian component
- You should generate synthetic data from a known GMM to test your implementation.
- For simplicity, you can assume data is 2-dimensional.

Deliverables:

- Implementation of the EM algorithm for GMMs in Python.
- A function that generates synthetic data from a GMM with known parameters.
- Visualizations showing:
  - The original data and initial GMM parameters
  - The final GMM parameters after convergence
  - A plot of the log-likelihood across iterations
- A brief analysis (2-3 paragraphs) explaining:
  - How EM algorithm works for parameter learning in GMMs
  - How you determined convergence
  - The impact of initialization on the final parameters
  - How the number of components K affects the model fit

Starter code is provided in the file `gmm_parameter_learning.py`, which includes the basic structure and helper functions.

Problem 10: Hidden Markov Model

(10 points)
A doctor is monitoring a patient over three days to determine whether the patient's underlying condition is a viral infection or a bacterial infection. The patient's observable symptom each day is their temperature, which can be either "Normal" or "High." The doctor knows that the patient's condition (viral or bacterial) can change day-to-day based on transition probabilities, and the temperature depends on the condition each day.

The model details are as follows:

- **Hidden States**: The patient's condition can be "Viral" (V) or "Bacterial" (B).
- **Observations**: The patient's temperature can be "Normal" (N) or "High" (H).
- **Initial Probabilities**: On day 1, there's a 70% chance the patient has a viral infection and a 30% chance of a bacterial infection.
- **Transition Probabilities**:
    - If the patient has a viral infection on one day, there's a 80% chance it remains viral the next day and a 20% chance it switches to bacterial.
    - If the patient has a bacterial infection on one day, there's a 60% chance it remains bacterial the next day and a 40% chance it switches to viral.
- **Emission Probabilities**:
    - If the patient has a viral infection, there's a 60% chance of a normal temperature and a 40% chance of a high temperature.
    - If the patient has a bacterial infection, there's a 30% chance of a normal temperature and a 70% chance of a high temperature.

The doctor observes the following sequence of temperatures over three days: Normal (N), High (H), High (H).

Implement a Hidden Markov Model in Python to compute the probability of this observation sequence occurring, assuming the described HMM.

Since there are only a few states in the problem, we ask you to solve this problem manually and implement the forward function without using libraries such as *hmmlearn*.

Then, implement a backward function going from t = 2 to t = 0 and confirm that these two algorithms give the same probability. For the backward algorithm, initialize the probabilities at the final time step (day 3) as 1.0 for each state (i.e., $\beta_3(\text{Viral}) = 1.0$, $\beta_3(\text{Bacterial}) = 1.0$), since there are no future observations beyond day 3.

The dependencies are as follows:

- The condition (viral or bacterial) on each day depends on the condition of the previous day.
- The temperature (normal or high) on each day depends on the condition on that day.

Add comments to your code at every step (this will be used for grading). In your writeup, please answer the following questions clearly:

- How you came up with the solution (can also be a quick paragraph and a sketch of HMM network)
- How are the probabilities stored and how are they used for inference? Please explain.

Please complete your implementation in the provided hmmvb.py file provided.

Problem 11: Maximum A Posteriori estimation

(10 points)

Maximum A Posteriori (MAP) estimation is a widely used approach to approximate Bayesian Learning by selecting the hypothesis that maximizes the posterior probability given the observed data. One common application is text classification, where we classify documents into predefined categories based on their content.
For example,

Problem Specifications:
You are given a dataset containing labeled text samples for two categories: Sports and Politics. Your goal is to train a classifier using MAP estimation to determine the most likely class for a new document. Assume that word occurrences follow a multinomial distribution, similar to a Naïve Bayes classifier.

- Each document is represented as a bag-of-words (word counts).
- The dataset consists of labeled training samples: (text, label), where the label is either Sports or Politics.

Given a new document d, predict the class C that maximizes the posterior probability

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

Since P(d) is the same for all classes, use

$$\hat{C} = \arg\max_{C} P\,(d|C)P(C)$$

Assume P(C) follows a prior estimated from the training data.
Use **Laplace smoothing** to handle unseen words.

Deliverables:

- A function that trains the MAP-based classifier using a given labeled dataset.
- A function that predicts the most likely class for a new document.
- A well-documented Python script with:
    - Code for training and prediction.
    - Example input and output for testing.
- Comments explaining how Laplacian smoothing is used:

Hints:

- Refer to the provided skeleton code **MAP.py** template to get started with structuring your implementation.
- You can use NLTK, scikit-learn, or pandas to preprocess text data.

- Convert text to lowercase and remove stop words before training.
- Construct a vocabulary from training data to compute word probabilities.