CS 362 / 562 - Assignment 4 Fall, 2025

Assignment 4 - Belief networks & Hidden Markov Models

This assignment has two goals:

- To demonstrate a practical belief network
- To apply your knowledge of Hidden Markov Models to a practical problem

… and two corresponding parts.

Find the starter code and data files via https://classroom.github.com/a/s_9vNi-u.

Part 1: Belief Networks

For this part of the assignment, you'll be using the pgmpy library for probabilistic inference. Install this if you haven't already. There are three steps to this part.

To start, take a look at the alarm.py file, which encodes the earthquake example from class. At the bottom is an example of how to query the network to find the probability that John calls given an Earthquake.

Step 1. Add additional queries to determine:

- The probability of Mary calling given that John called
- The probability of both John and Mary calling given Alarm
- The probability of Alarm, given that Mary called

Include each of your queries in alarm.py. Add a main that executes each of your queries.

Step 2. Next, consider the carnet.py file. This contains the Bayesian network representing the car starting problem. To begin, ask the following queries:

- Given that the car will not move, what is the probability that the battery is not working?
- Given that the radio is not working, what is the probability that the car will not start?
- Given that the battery is working, does the probability of the radio working change if we discover that the car has gas in it?
- Given that the car doesn't move, how does the probability of the ignition failing change if we observe that the car does not have gas in it?
- What is the probability that the car starts if the radio works and it has gas in it? Include each of your queries in carnet.py. Also, please add a main that executes your queries.

Step 3. Last, we will add an additional node to the network, called KeyPresent, that indicates whether or not we have the key for the car. This is a Categorical variable with two state values: yes and no. The prior for 'yes' is 0.7.

KeyPresent should only affect Starts. Add an edge to starts and update the CPD to indicate that:
P(starts | gas, ignition, keyPresent) = 0.99
P(starts | gas, !ignition, keyPresent) = 0.01
P(starts | !gas, ignition, keyPresent) = 0.01
P(starts | gas, ignition, !keyPresent) = 0.01
P(starts | !gas, !ignition, keyPresent) = 0.01
P(starts | !gas, ignition, !keyPresent) = 0.01
P(starts | gas, !ignition, !keyPresent) = 0.01
P(starts | !gas, !ignition, !keyPresent) = 0.01

Add a query showing the probability that the key is not present given that the car does not move.

Part 2: Hidden Markov Models

This assignment is to write a spelling fixer using a Hidden Markov model. You will take user input and correct the spelling using the Viterbi algorithm. This part of the assignment is derived from an Spelling Fixer assignment by Rasika Bhalerao in AAAI's Model Assignments workshop.

Learning goals:

- Calculate emission and transition probabilities for a Hidden Markov model
- Use those probabilities to correct spelling errors using the Viterbi algorithm

What to do:

1. Write code to calculate the emission probabilities from aspell.txt. To do this, iterate through each word character-by-character. For each “correct” letter, calculate the frequency with which each typed letter is emitted. Remember, in most cases, the highest emission probability for each “correct” letter should be the letter itself.
2. Write code to calculate the transition probabilities. Using the “correct” letters, calculate the probabilities for going from the start state to each letter, from each letter to each other letter, and from each letter to the end state.
3. Write code to correct user text.
    a. Take some user input text, and split it into words using whitespace.
        i. Python’s split() function is useful.
    b. For each word, use the Viterbi algorithm to decode it.
        i. The typed letters are the emissions. The “correct” letters are the states.
        ii. Print the decoded word.
4. Test it out!

Reflection

Answer the following four (4) questions for Part 2 (Hidden Markov Models):

- Give an example of a word which was correctly spelled by the user, but which was incorrectly “corrected” by the algorithm. Why did this happen?
- Give an example of a word which was incorrectly spelled by the user, but which was still incorrectly “corrected” by the algorithm. Why did this happen?

- Give an example of a word which was incorrectly spelled by the user, and was correctly corrected by the algorithm. Why was this one correctly corrected, while the previous two were not?
- How might the overall algorithm's performance differ in the "real world" if that training dataset is taken from real typos collected from the internet, versus synthetic typos (programmatically generated)?

Submission

Submit the following to your github repository:

- Your Python code for Part 1 (Belief Networks) and Part 2 (Hidden Markov Models)
- A README.md file with your answers to the Reflection questions