Artificial Intelligence

Tutorial week 9 – Uncertain reasoning

COMP9414

1 Background

This tutorial focuses on reasoning with uncertain information. Two experiments will be carried on. The first one consists of using a Bayesian Net to compute joint and conditional probabilities using the `pomegranate` package. The second experiment involves the simulation of a Fuzzy Logic Controller using the package `skfuzzy`.

1.1 Belief networks

A Bayesian or belief network is a probabilistic graphical model used to represent and analyze the dependencies between variables. In this network, nodes represent random variables, and directed edges between nodes indicate conditional dependencies. The strength of these dependencies is captured by conditional probability tables associated with each node.

Bayes Nets are widely used in various fields, including artificial intelligence, machine learning, statistics, and expert systems. They enable reasoning and inference by calculating probabilities based on evidence and updating beliefs in a systematic way using Bayesian probability principles.

1.2 Fuzzy logic

Fuzzy logic is a form of logic that allows for reasoning and decision-making in situations that involve ambiguity, imprecision, or uncertainty. Unlike classical binary logic, which deals with crisp true/false values, fuzzy logic allows for partial truths, represented by values between 0 and 1, which are referred to as degrees of membership.

The framework operates based on fuzzy rules, which use fuzzy information to model relationships between inputs and outputs. These rules are expressed in the form of if-then statements. Fuzzy logic provides a flexible and intuitive method for dealing with uncertainty and imprecision, allowing machines to emulate human-like decision-making and problem-solving processes.

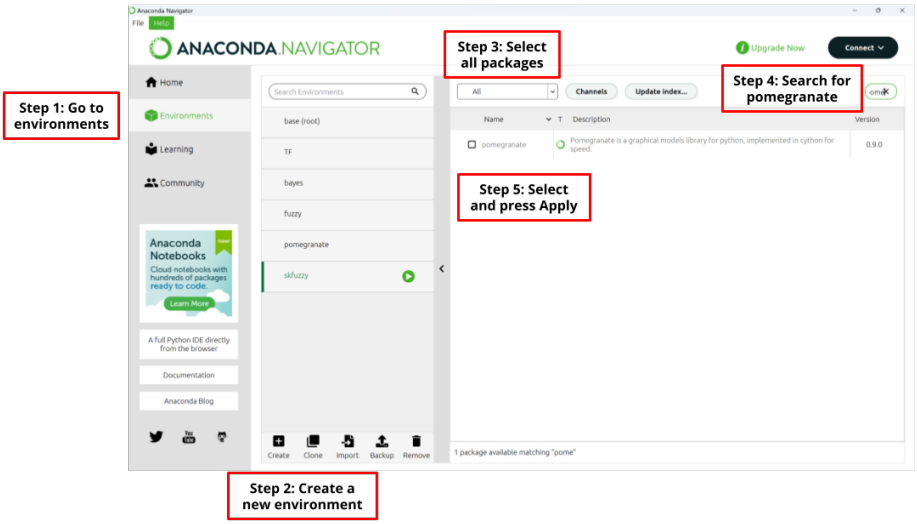

Figure 1: Package installation steps in Anaconda for `pomegranate`.

2 Setup

We need to install two libraries for these experiments: `pomegranate` and `skfuzzy`. To install the packages on your machine, you can follow the instructions given in this section.

2.1 Pomegranate

To install `pomegranate` package, Fig. 1 shows a schematic with a sequence of steps in Anaconda. We describe these steps next:

- **Step 1:** We recommend working in a new environment to avoid any package conflicts. Therefore, the first step is going to the Environments section in Anaconda.
- **Step 2:** To create a new environment, we need to assign a name and select a Python version, please use Python 3.8.19 or older as pomegranate 0.x will not work with newer versions. Have in mind that the new environment will have only a few libraries installed by default, however, libraries required or dependencies will be installed when you chose a package.
- **Step 3:** List all the packages available, otherwise, if you search for a particular one, Anaconda will try to find it in the list shown (e.g., the installed packages only).
- **Step 4:** Search for `pomegranate` package. You should be able to find a 0.x version (e.g., 0.9.0, 0.13.0, or 0.14.x should be fine). **Important:** if

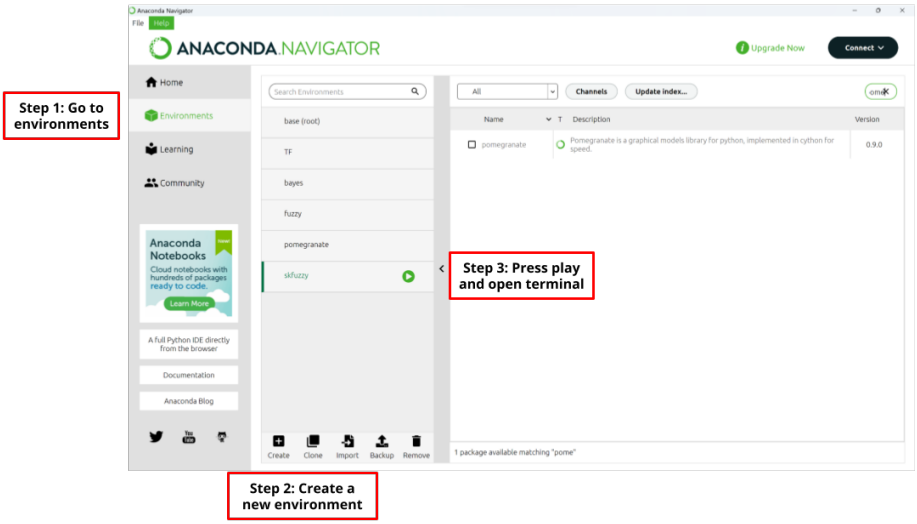

Figure 2: Initial package installation steps in Anaconda for `skfuzzy`.

you install the package manually, do not install the last version 1.0.1, as this has been recently released using a different back-end and API and some functions are used differently and some for Bayes nets are not yet implemented in the new version.

- **Step 5:** Select the library to install and press Apply in the right bottom corner.

In case you are using Spyder, note that it is not installed by default in the new environment, therefore, you need to install it the first time before using it.

Alternative: if you are familiar with YAML environments, you can install the provided YAML environment by either importing it in Anaconda Navigator or using `conda env create -f pomegranate_env.yml` in Anaconda Prompt.

2.2 Skfuzzy

To install `skfuzzy` package, we will follow a slightly different approach. Fig. 2 shows the initial steps needed for this. We describe all steps for installation next:

- **Step 1:** We also recommend working in a new environment to avoid any package conflicts. Therefore, the first step is going to the Environments section in Anaconda.
- **Step 2:** To create a new environment, we need to assign a name. Have in mind that the new environment will have only a few libraries installed by default, however, libraries required or dependencies will be installed when you chose a package.

- **Step 3:** Press play and open a new terminal.
- **Step 4:** In the terminal, enter `pip install -U scikit-fuzzy`. This will start the package installation along with some dependent packages.
- **Step 5:** As `matplotlib` is not installed by default, in the terminal enter `python -m pip install -U matplotlib`. Once finished, you can exit the terminal and use Anaconda as usual.

In case you are using Spyder, note that it is not installed by default in the new environment, therefore, you need to install it the first time before using it.

Alternative: if you are familiar with YAML environments, you can install the provided YAML environment by either importing it in Anaconda Navigator or using `conda env create -f skfuzzy_env.yml` in Anaconda Prompt.

3 Belief network in Python

Consider the problem domain reviewed during the lecture when I go home, and I want to know if someone from my family is home before I go in. Let's say I know the following information:

1. When my wife leaves the house, she often (but not always) turns on the outside light.
2. When nobody is home, the dog is often left outside.
3. If the dog has bowel troubles, it is also often left outside.
4. If the dog is outside, I will probably hear it barking (though it might not bark, or I might hear a different dog barking and think it's my dog).

Given the previous information, we can consider the following five Boolean random variables:

1. family-out (fo): everyone is out of the house.
2. light-on (lo): the light is on.
3. dog-out (do): the dog is outside.
4. bowel-problem (bp): the dog has bowel troubles.
5. hear-bark (hb): I can hear the dog barking.

From this information, the following direct causal influences are appropriate:

1. hb is only directly influenced by do. Hence hb is conditionally independent of lo, fo and bp given do.

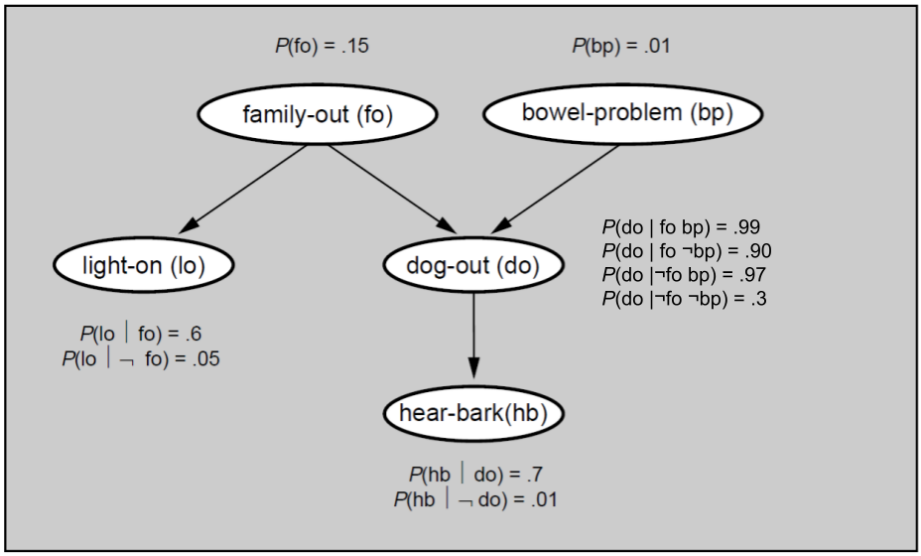

Figure 3: Belief network for the family out problem.

2. do is only directly influenced by fo and bp. Hence do is conditionally independent of lo given fo and bp.

3. lo is only directly influenced by fo. Hence lo is conditionally independent of do, hb and bp given fo.

4. fo and bp are independent.

The belief network representing these direct causal relationships (though these causal connections are not absolute, i.e., they are not implications) is shown in Fig. 3. The network includes the prior probability of the random variable for root nodes fo and bp as well as the conditional probabilities of the node's variable given all possible combinations of its immediate parent nodes need to be determined for non-root nodes.

3.1 Exercise:

1. Using `pomegranate` create the root nodes fo and bp and assign them the respective prior probability.

2. Create the non-root nodes lo, do, and hb. For lo and hb consider four combinations taking into account one immediate parent. For do consider eight combinations as this node has two immediate parents.

3. Create the Bayesian network model.

4. Using the previous nodes, create five states and add them to the Bayesian network model.

5. Add to the model four edges as shown in Fig. 3.

6. Finalise the model to use it for joint and conditional probabilities.

7. Test the Bayesian network model for different situations.

    - Test 1: Compute the joint probability P(~FO,BP,~LO,DO,HB).
    - Test 2: Compute a causal inference (top-down) using the conditional probability P(DO|BP).
    - Test 3: Compute a diagnostic inference (bottom-up) using the conditional probability P(BP| DO).

4 Fuzzy logic controller in Python

In the second experience, we aim at building a fuzzy logic controller to regulate the fuel injection to an engine from two variables used as antecedents: speed and temperature. The domain of each variable is as follows:

- **Speed:** from 0 to 150.
- **Temperature:** from 0 to 150.
- **Injection:** from 0 to 100.

Both input variables use fuzzy values for three ranges, as shown next (see Fig. 4):

- **Low speed:** A trapezoid membership function equal to one between 0 and 25, and decreasing to zero between 25 and 60.
- **Medium speed:** A triangular membership function increasing from zero to one between 25 and 75, and decreasing from one to zero between 75 and 125.
- **High speed:** A trapezoid membership function increasing from zero to one between 60 and 100, and equal to one between 100 and 150.

The temperature variable has the same values for its membership function, i.e., low temperature, medium temperature, and high temperature. The output variable uses fuzzy values for three ranges as well, this is shown in Fig. 5 and described next:

- **Low injection:** A triangular membership function decreasing from one to zero between 0 and 30.

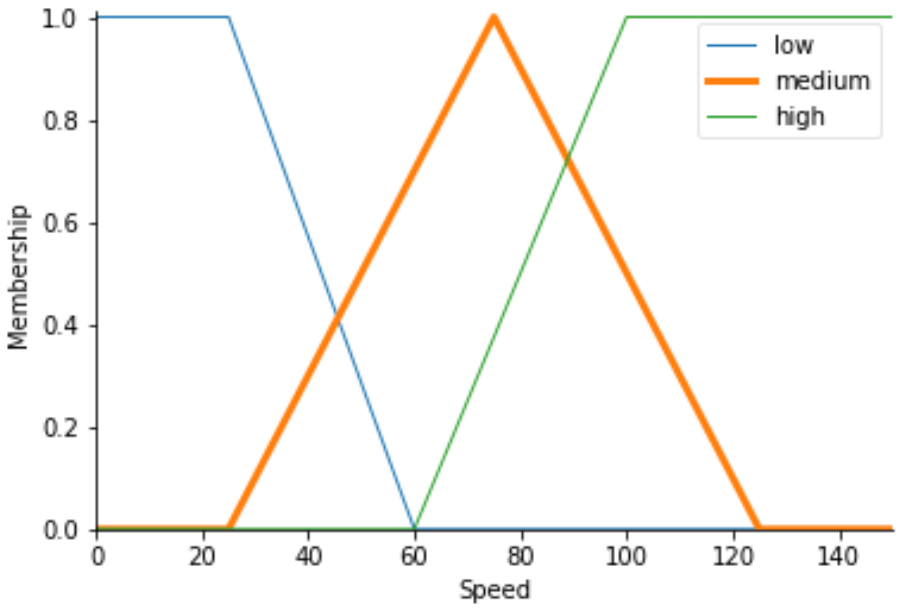

Figure 4: Membership function for speed (temperature variable uses the same membership function).

- **Medium injection:** A triangular membership function increasing from zero to one between 10 and 50, and decreasing from one to zero between 50 and 90.
- **High injection:** A triangular membership function increasing from zero to one between 70 and 100.

The fuzzy logic controller will implement the following rules:

- IF speed = *medium* AND temperature = *high* THEN injection = *low*.
- IF speed = *low* AND temperature = *low* THEN injection = *high*.
- IF speed = *high* AND temperature = *medium* THEN injection = *low*.

4.1 Exercise:

1. Using `skfuzzy` choose and create the antecedents and consequent for the fuzzy logic controller. For the consequent, use the defuzzification method centroid.
2. Build the membership function for each variable, i.e., speed, temperature, and injection.

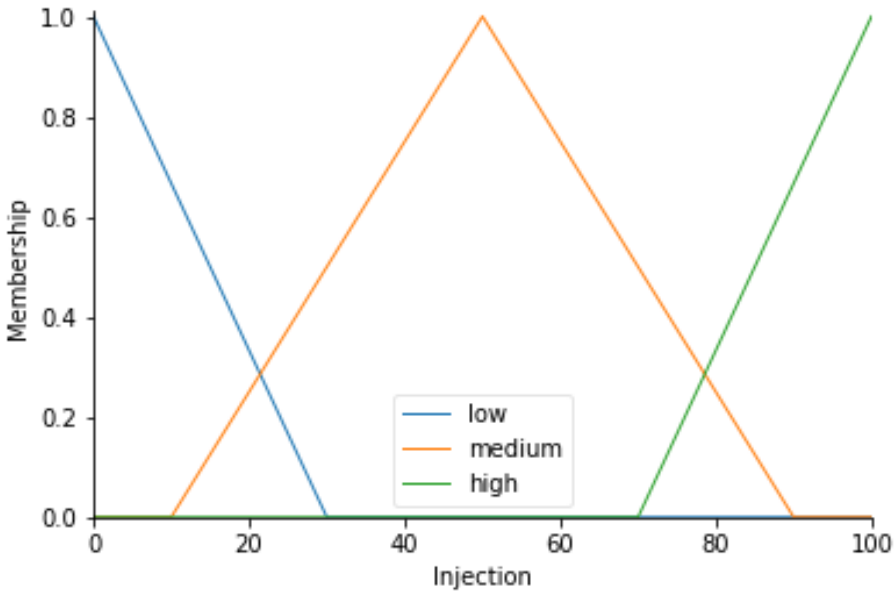

Figure 5: Membership function for output variable injection.

3. Visualise the fuzzy sets for each variable.
4. Create the rules implemented by the controller.
5. Using the rules, create the controller and a control simulator.
6. Test the controller with a couple of speeds and temperatures and visualise the output.
    - Test 1: speed = 50, temperature = 10.
    - Test 2: speed = 70, temperature = 100.
7. Add the following rules to have a full definition of the controller:
    - IF speed = *low* AND temperature = *medium* THEN injection = *high*.
    - IF speed = *low* AND temperature = *high* THEN injection = *medium*.
    - IF speed = *medium* AND temperature = *low* THEN injection = *high*.
    - IF speed = *medium* AND temperature = *medium* THEN injection = *medium*.
    - IF speed = *high* AND temperature = *low* THEN injection = *medium*.
    - IF speed = *high* AND temperature = *high* THEN injection = *low*.
8. Plot the surface of the controller for both input variables: speed and temperature.