RBE 500 Homework #8

Arjan Gupta

Bayes Filter Report

I started my Bayes Filter implementation in a python file called `hw8_bayes_filter.py`. At the top of the file, I added a `VERBOSE` flag that helps me toggle on and off output that is helpful for debugging, but probably superfluous for the grader. I defined a list of states of the door, which are *open* and *closed*. I also defined the list of 'base' beliefs, which are given in the slides/textbook as the following.

$$\begin{aligned} bel(X_0 = \textbf{open}) &= 0.5 \\ bel(X_0 = \textbf{closed}) &= 0.5 \end{aligned}$$

These are the initial beliefs (as denoted by X_0), which means that these values will adjust as the script runs through the iteration cases. In my code, these are simply a python list, followed by two helper functions. One helper function helps us print the current belief values. The other helper function makes sure our states, which are strings, are 'mapped' to the belief values, and returns them as such.

```python
# List of base beliefs for [open, closed]
# These are the initial beliefs. As we run our
# iterations, these base beliefs change.
base_belief_list = [0.5, 0.5]

# Helper function for printing base beliefs
def print_base_beliefs():
    print(f'bel(open) = {base_belief_list[0]}')
    print(f'bel(closed) = {base_belief_list[1]}')

# Helper function to get "base" beliefs
def get_base_belief(state) -> float:
    if state == 'open':
        return base_belief_list[0]
    elif state == 'closed':
        return base_belief_list[1]
    else:
        return 0.0
```

Next, I started defining the helper function to get the belief values associated with the sensor, i.e. the measurement beliefs. In the slides, these are given as the following.

$$\begin{aligned} p(Z_t = \textbf{sense\_open} \mid X_t = \textbf{is\_open}) &= 0.6 \\ p(Z_t = \textbf{sense\_closed} \mid X_t = \textbf{is\_open}) &= 0.4 \\ p(Z_t = \textbf{sense\_open} \mid X_t = \textbf{is\_closed}) &= 0.2 \\ p(Z_t = \textbf{sense\_closed} \mid X_t = \textbf{is\_closed}) &= 0.8 \end{aligned}$$

Therfore, my helper function for the sensor is the following.

```python
# Helper function for retrieving measurement beliefs
def get_measurement_belief(sense, true_state) -> float:
    if true_state == 'open':
        return 0.6 if sense == 'open' else 0.4
    elif true_state == 'closed':
        return 0.2 if sense == 'open' else 0.8
    else:
        return 0.0
```

I used the condition on the right side of the probability statements to write the if-else logic branching in the helper function. For the sake of making sure that I only return a valid value from known strings, which are `'open'` and `'closed'`, I returned 0.0 when any other string is retrieved.

My next helper function is used for retrieving values associate beliefs with actions, which are given in the textbook/slides as the following.

$$
\begin{aligned}
p(X_t = \textbf{is\_open} \mid U_t = \textbf{push}, X_{t\text{-}1} = \textbf{is\_open}) &= 1 \\
p(X_t = \textbf{is\_closed} \mid U_t = \textbf{push}, X_{t\text{-}1} = \textbf{is\_open}) &= 0 \\
p(X_t = \textbf{is\_open} \mid U_t = \textbf{push}, X_{t\text{-}1} = \textbf{is\_closed}) &= 0.8 \\
p(X_t = \textbf{is\_closed} \mid U_t = \textbf{push}, X_{t\text{-}1} = \textbf{is\_closed}) &= 0.2 \\
p(X_t = \textbf{is\_open} \mid U_t = \textbf{do\_nothing}, X_{t\text{-}1} = \textbf{is\_open}) &= 1 \\
p(X_t = \textbf{is\_closed} \mid U_t = \textbf{do\_nothing}, X_{t\text{-}1} = \textbf{is\_open}) &= 0 \\
p(X_t = \textbf{is\_open} \mid U_t = \textbf{do\_nothing}, X_{t\text{-}1} = \textbf{is\_closed}) &= 0 \\
p(X_t = \textbf{is\_closed} \mid U_t = \textbf{do\_nothing}, X_{t\text{-}1} = \textbf{is\_closed}) &= 1
\end{aligned}
$$

These are represented in my code as the following.

```python
# Helper function for retrieving action beliefs
def get_action_belief(prediction, action, last_known) -> float:
    if action == 'open':
        if last_known == 'open':
            return 1.0 if prediction == 'open' else 0.0
        elif last_known == 'closed':
            return 0.8 if prediction == 'open' else 0.2
        else:
            return 0.0
    elif action == 'do_nothing':
        if last_known == 'open':
            return 1.0 if prediction == 'open' else 0.0
```

```
        elif last_known == 'closed':
            return 0.0 if prediction == 'open' else 1.0
        else:
            return 0.0
    else:
        return 0.0
```

Next, I defined a data-only class called `FilterIteration` to represent the filter iterations in a coherent manner. The data members of this class are simply `self.action` and `self.measurement`. In my main function, I defined a list of instances of this `FilterIteration` class, as given in the prompt of this assignment. The list is shown below here.

```
    filter_iteration_list = [
        FilterIteration('do_nothing', 'closed'),
        FilterIteration('open', 'closed'),
        FilterIteration('do_nothing', 'closed'),
        FilterIteration('open','open'),
        FilterIteration('do_nothing', 'open')
    ]
```

I now began implementating my Bayes Filter algorithm. The algorithm is contained in the `bayes_filter_algorithm()` function in my code. The code loops over all states of the door and performs the prediction step and correction step for each step. After these steps are done, the algorithm computes the normalizer, η, and updates the `base_belief_list` for usage in the next iteration.

The prediction and correction step in my algorithm make use of helper functions, named `bf_predict_step()` and `bf_correct_step()` respectively. The prediction step helper function loops over all states and sums the products of the action-belief and base beliefs, as given in the pseudo-code algorithm in the lecture slides. The correction step helper function simply returns the product of the measurement belief and the prediction step result. This is also implemented as seen in the slides. The code snippet for my implementation of the algorithm is shown below here.

```
# Helper function for the prediction stage of bayes filter
def bf_predict_step(state, action):
    sum = 0
    for s in all_states:
        sum += get_action_belief(prediction=state, action=action, ...
            last_known=s)*get_base_belief(state=s)
    return sum

# Helper function for the correction stage of bayes filter
```

```
def bf_correct_step(state, measurement, prediction):
    return get_measurement_belief(sense=measurement, ...
        true_state=state)*prediction

# Implementation of the Bayes Filter algorithm
def bayes_filter_algorithm(iteration: FilterIteration):
    # Display information about current iteration case
    print(f'The given action is {iteration.action} and given mesurement ...
        is {iteration.measurement}')
    # Declare lists for each step
    predictions_list = []
    correction_list = []
    # Loop over [open, closed] to perform prediction and correction steps
    for s in all_states:
        # Step 1 - prediction stage
        prediction = bf_predict_step(s, iteration.action)
        if VERBOSE:
            print(f'Prediction for {s} is {prediction}')
        predictions_list.append(prediction)
        # Step 2 - correction stage
        correction = bf_correct_step(s, iteration.measurement, ...
            prediction=prediction)
        if VERBOSE:
            print(f'Correction for {s} is {correction}')
        correction_list.append(correction)
    # Use correction list to calculate normalizer
    normalizer = 1/sum(correction_list)
    if VERBOSE:
        print(f'The normalizer is {normalizer}')
    # Update beliefs
    global base_belief_list
    base_belief_list = [b * normalizer for b in correction_list]
```

Finally, after we run our iterations through this algorithm, we get the following output.

```
---- Starting HW8 Bayes Filter ----

The initial beliefs are:
bel(open) = 0.5
bel(closed) = 0.5
------------------------
Iteration 1:
The given action is do_nothing and given mesurement is closed
The updated beliefs are:
bel(open) = 0.3333333333333333
bel(closed) = 0.6666666666666666
------------------------
```

```
Iteration 2:
The given action is open and given mesurement is closed
The updated beliefs are:
bel(open) = 0.7647058823529411
bel(closed) = 0.23529411764705882
------------------------
Iteration 3:
The given action is do_nothing and given mesurement is closed
The updated beliefs are:
bel(open) = 0.6190476190476191
bel(closed) = 0.38095238095238093
------------------------
Iteration 4:
The given action is open and given mesurement is open
The updated beliefs are:
bel(open) = 0.9732441471571905
bel(closed) = 0.026755852842809368
------------------------
Iteration 5:
The given action is do_nothing and given mesurement is open
The updated beliefs are:
bel(open) = 0.9909194097616345
bel(closed) = 0.009080590238365497
```