HMM Write Up

1 - In this assignment we implemented a generic HMM to help George filter his noisy touchscreen data, in order to successfully order the shipments. We are able to do this by primarily using the **filter_noisy_data** function in our touchscreenHMM class (a class that we exclusively created to implement HMM for our noisy touchscreen analysis). This function would take in the touchscreens noisy reading (i.e. **frame**), which would be a **2d numpy array** containing all 0s except one 1, corresponding to the **observed** location of George's finger; these frames will be our observations. The filter_noisy_data function would then output a probability distribution over the true location of George's finger as another 2d array, considering the current observations (noisy reading frames) and the past observations.

Computationally, the function works by including a call to the tell() function in our HMM class, which takes in the observation (noisy touchscreen reading frame) as a 2d numpy array, records it and processes it. Then the function includes a call to ask() passing in the current timestep and outputting the probability distribution over **hidden states (that we represent as floats**) at that time informed by the observations so far. Ask() and Tell() perform their computation relative to the following equation:

$$\Pr(S_{t+1} \mid o_{1,2,\ldots,t+1}) = \alpha \cdot \Pr(o_{t+1} \mid S_{t+1}) \sum_{s_t} \Pr(S_{t+1} \mid S_t = s_t) \cdot \Pr(S_t = s_t \mid o_1, o_2, \ldots, o_t)$$

Our sensor model (corresponding to $\Pr(o_{t+1} \mid S_{t+1})$) that gives us the probability of observing a certain observation o given that we are in state s, and our transition model (corresponding to $\Pr(S_{t+1} \mid S_t = s_t)$) that gives us the probability of transitioning to state S_{t+1} from state S_t, helps us in the implementation of the equation.

We represent a **state as a float** which corresponds to the position of the finger on the touchscreen; we could hypothetically model the touchscreen as a 2d numpy array, and accordingly the state float corresponds to the index of the position of the finger in the flattened touchscreen, e.g. if a finger is at to the [1][1] cell of a 3x3 screen, the float representation for the state would be 3.

The sensor model takes in an observation as a 2d numpy array and a state as a float computes the probability of getting the input observation from the input state, by calculating the euclidean distance between the location of the finger specified by the observation and the location of the finger specified by the state. If the positions are the same in both the observation and the state the probability is 0.5; else if the euclidean distance is 1 the probability is 0.5/(the neighboring cells of the observed position); else its 0.

The transition model takes in an old_state and a new_state, both as floats and computes the probability of getting to the new_state from the old state. The calculations for the probability are governed by the following factors:

- In a single timestep, George's finger will always either stay in the same place or move to one of the 8 adjacent locations (including diagonally adjacent locations).
- George's finger is more likely to continue moving in the same direction as it moved in the last timestep than it is to move in any other direction - the concept of directional velocity. Otherwise, his finger moves in a uniformly random direction.
- If George's finger is about to move off the edge of the touchscreen, he will pause for one timestep, and then be more likely to move in the opposite direction at the next timestep, just as if he had been moving in the opposite direction before.

In the end, the filter_noisy_data function uses np.asanarray().reshape() function to reshape the output of ask() - which was a list - into a 2d numpy array, thereby outputting the probability distribution over the hidden states.

2 - The relative frequency for the finger moving to the cell in the direction of the previous cell was more when there were a greater number of past movements in that direction as compared to just one past movement in that direction, giving the concept of directional velocity. I then tried to incorporate it by recording the position of the fingers in the past states and if those positions were in the same direction then I computed a greater probability for the next position in that direction. I also noticed that there was a greater relative frequency for the finger staying in the same position as compared to moving somewhere. I tried incorporating it by computing a greater probability for the state which has a finger in the same position as the previous state.

3 -
For the final Sensor model, I mainly calculated the euclidean distance between the position of the finger reflected by the float representation (o_int) of our 2d-numpy array observation (the connection between the 2d arrays and their float representations is explained in q1) and the position of the finger reflected by the state. If the state and o_int are the same then the probability is 0.5, else if the euclidean distance is 1 the probability is 0.5/(the neighboring cells of the observed position). The number of cells vary relative to the position of the finger in the observation, if it is in one of the 4 corners then the neighboring cells are 3, else if it is one of the other border cells the neighboring cells are 5, else there are 8. If the euclidean distance is more than 1 then probability is 0.

For the transition model, I used the self.prev_states[] to incorporate the effect of directional velocity. I populated the self.prev_states[] in the filter_noisy_data function by storing the index of the max value value of the output list from ask, as that index would correspond to our float representation for the most probable position of the finger at an instance.

Moreover, I tried to compute the probabilities according to the following pointers:

- In a single timestep, George's finger will always either stay in the same place or move to one of the 8 adjacent locations (including diagonally adjacent locations).
    I did this by computing the euclidean between the positions reflected by both the states. If the euclidean distance was greater than 1, the function would return zero; otherwise it will continue.
- George's finger is more likely to continue moving in the same direction as it moved in the last timestep than it is to move in any other direction - the concept of directional velocity. Otherwise, his finger moves in a uniformly random direction.
    I did this by adding weights to the probability p. If the previous state, the current state and the next state are in the same direction, p would be incremented by a certain value, there were 8 different situations in this regard - left, right, top, bottom, NE-SW, SW-NE, NW-SE, SE-NW. In order to account for the directional velocity I used the self.prev_states[] list and nested nested if statements to look at the position of the finger at the previous to previous position and did the math above once again, incrementing p by a certain weight when condition met.
- If George's finger is about to move off the edge of the touchscreen, he will pause for one timestep, and then be more likely to move in the opposite direction at the next timestep, just as if he had been moving in the opposite direction before.
    I did this by computing the probability to 1 if the next state and the current state are the same, and the current state is a border state, and the previous state is not a border state. Moreover, once again I used self.prev_states[] list and incremented p with a certain weight if the new state was in the opposite direction from the previous to previous state (not the current one!), there were also 8 different cases to consider relative to the directions - left, right, top, bottom, NE-SW, SW-NE, NW-SE, SE-NW.

This was my planned strategy but it would always return an error that the frame is not a probability distribution :(
So, rather than this tried using simpler versions and those also did not work :((
4 - I could've made the assumption that a specific location of the finger relative to a state could account for its neighboring 8 cells as well thereby compromising accuracy

for time efficiency. Moreover, my proposed model in q 3 / q 5 compromises assumptions 2 and 3, as the directional velocity that we are considering in transition function takes past states and observations in to account as evident by our self.prev_states[] list.

5 - I tried considering the approach from q3 but it would not work because of the “frame is not a probability distribution” error.
(Snippet from q3)
I tried to compute the probabilities according to the following pointers:

- In a single timestep, George’s finger will always either stay in the same place or move to one of the 8 adjacent locations (including diagonally adjacent locations).
    I did this by computing the euclidean between the positions reflected by both the states. If the euclidean distance was greater than 1, the function would return zero; otherwise it will continue.
- George’s finger is more likely to continue moving in the same direction as it moved in the last timestep than it is to move in any other direction - the concept of directional velocity. Otherwise, his finger moves in a uniformly random direction.
    I did this by adding weights to the probability p. If the previous state, the current state and the next state are in the same direction, p would be incremented by a certain value, there were 8 different situations in this regard - left, right, top, bottom, NE-SW, SW-NE, NW-SE, SE-NW. In order to account for the directional velocity I used the self.prev_states[] list and nested nested if statements to look at the position of the finger at the previous to previous position and did the math above once again, incrementing p by a certain weight when condition met.
- If George’s finger is about to move off the edge of the touchscreen, he will pause for one timestep, and then be more likely to move in the opposite direction at the next timestep, just as if he had been moving in the opposite direction before.
    I did this by computing the probability to 1 if the next state and the current state are the same, and the current state is a border state, and the previous state is not a border state. Moreover, once again I used self.prev_states[] list and incremented p with a certain weight if the new state was in the opposite direction from the previous to previous state (not the current one!), there were also 8 different cases to consider relative to the directions - left, right, top, bottom, NE-SW, SW-NE, NW-SE, SE-NW.

6 -

(a) Smoothing mainly makes us go backwards in observations, where when we have k observations we want to go back to S_t where t is smaller than k. In order to do that, the following proportionality relation by bayes rule comes in handy:

$$P(S_t \mid O_0 \dots O_k) \propto P(O_t \dots O_k \mid S_t)\, P(S_t \mid O_0 \dots O_t)$$

$P(S_t \mid O_0, \dots, O_K)$ corresponds to the stuff that happened before t and the later corresponds to the stuff that happened after. We can get $P(S_t \mid O_0, \dots, O_t)$ by the forward algorithm, which can be summed up in the following lines of code:

Let $F(k, 0) = P(S_0 = s_k)P(O_0 \mid S_0 = s_k)$.

For t = 1, …, T:
  For k in possible states:

$$F(k,t) = P(O_t|S_t = s_k) \sum_i P(s_k|s_i)F(i, t-1)$$

$F(k,T)$ is $P(S_T = s_k, O_0 \dots O_T)$
(normalize to get $P(S_T \mid O_0 \dots O_T)$)

(credits - Lecture on HMM)

And for the $P(O_t, \dots, O_k \mid S_t)$ we use the backward pass algorithm, which follows the same principle at the forward algorithm but with the edit that the recursive part of the code goes backward in time. So we compute the forward algorithm till the end and then we compute the backward algorithm from the last piece of observation till t.

(b) In prediction, if we need to find the probability of being at a specific state ‘a’ at time t+1 we would multiply the probability of being a specific state at time t and the probability of transitioning to the state ‘a’ from that specific old state, we would then sum these products relative to all probable states at time t, e.g. if there were only two states ‘a’ and ‘b’ we would have the following:

$$P(S_1 = a) = P(S_0 = a)P(a \mid a) + P(S_0 = b)P(a \mid b)$$

Accordingly, for prediction we could recursively calculate the probability of being at a specific state at time t by the probability at time t-1, going all the way back to 0.

SRC Questions

1 -

I personally think that AI usually augments professions rather than completely replacing them. It is true that “machines don’t fall ill, they don’t need to isolate to protect peers, they don’t need to take time off from work,” (Article) but all of these things make their use more effective rather than adding to the vulnerability of certain professions. The augmentation nature of AI should strictly be enforced where the question of human life is involved and one such situation is the use of AI in law making. As laws govern the people of a country and could account for cases where too much is at stake, e.g. whether someone should be convicted for a lifetime in jail for a particular crime. Similarly, another profession is a truck driver, a truck could easily jeopardize someone’s safety if there is an error in the truck driving system. Thus, we could use AI to augment human beings in such situations e.g. recommendation engines for lawmakers, and GPS viable/feasible routes for truck driver, rather than completely replacing them; in cases where the situation is not that critical (e.g. a relatively controlled environment, or where humans are not endangered e.g. cashier) we could use AI do things which would rather be done by human beings, however.

2-

Which party do you think is responsible for accounting for the displacement of jobs from AI? The developers of the AI? The government? The employer? Or another party?

I think almost all of them are responsible for the displacement of jobs from AI. This is because everyone wants maximum utility, and AI creates value for almost everyone - the developer of AI, the government, the employer. The AI developers get a sense of achievement and money when they create a new AI machines and software. The government could also benefit from AI through Society 5.0, for example, where everything is digitally connected through the internet of things - traffic lights (less need for traffic sergeants), train stations (less need for train conductors), cars (less need for travel planning assistants), safe-city cameras (less need for security guards) etc. - which could not only reduce crime rate, but also reduce energy consumption and generally raise the standards of living. AI for employers could mean reduced long term costs as they wouldn’t have to hire labor for many things, enriched user experience for their customers, valuable

business insights through the use of business intelligence and much more. Thus the popularity of AI and the subsequent displacement of jobs is inevitable.