Ian MacDonald

ESE 650

Summary 1 (Lectures 4-10, Chapters 2-4)

State Estimation

Markov Chains and Hidden Markov Models Overview

Markov chains represent a series of states where the probability distribution of each state is only dependent on the state that came directly before it. The probability of going to a certain state given the previous state is modeled using a transition matrix, which records the probability of going to each other possible state from the given previous state. Often associated with Markov Chains are Hidden Markov Models, a sequence of observations of the state the Markov Chain is in at given points. The observation at any given point in the chain is only dependent on the state that point in the chain is in. The observation recorded is determined by the observation matrix, which contains the probability of each possible observation given the actual state of the Markov Chain.

Different Types of State Estimation

Given a Markov Chain and associated Hidden Markov Model, there are various types of state estimations we may be interested in. Below are the common ones, with a brief description of each one.

Filtering

The filtering problem is used to determine the probability distribution of the state of the Markov Chain at any point, given all of the observations leading up to that point. It answers the question: What is the chance the Markov Chain is in a certain state given all of the states we have observed up to this point?

Mathematically:

$$\mathbf{P}(X_k \mid Y_1, \ldots, Y_k).$$

Smoothing

The smoothing problem is used to determine the probability distribution of all the previous states, given observations up to some current state. It answers the question: Given all the observations up to this point, what are the chances the Markov Chain was in a certain state for each point before the current one?

Mathematically:

$$\mathbf{P}(X_j \mid Y_1, \ldots, Y_k) \quad \text{for } j < k.$$

Prediction

The prediction problem is used to predict future states. It answers the question: Given all of our observations up to this point, what is the probability distribution of the next unobserved state? Similar to the filtering problem, but filtering reasons about a state where we have an observation, while prediction deals with future states which we have not observed.

Mathematically:

$$\mathrm{P}(X_j \mid Y_1, \ldots, Y_k) \quad \text{for } j > k.$$

Decoding

The decoding problem finds the most likely series of states given a series of observations. It answers the same question as the smoothing problem, but differs by modeling the probability of seeing all observations jointly rather than each one independently.

Mathematically:

$$\mathrm{P}(X_1, \ldots, X_k \mid Y_1, \ldots, Y_k)$$

Likelihood of Observations

Given our observations, what was the likelihood of observing the exact sequence we did?

Mathematically:

$$\mathrm{P}(Y_1, \ldots, Y_k).$$

Algorithms for Calculating State Estimation

Both the forward algorithm and the backward algorithm are extremely useful building blocks for calculating the various different types of state estimations mentioned above. These algorithms are then used within other algorithms in order to calculate the above.

The Forward Algorithm

The forward algorithm is used to calculate the forward variable, which models the joint probability of seeing all of our given observations up until a certain point, as well as seeing a given state at that point. The algorithm is calculated recursively, with each point in the algorithm using the value computed at the previous point. For any point, we will have a probability distribution over all of the possible states with our given observations, and a summation of these probabilities would give us the probability of seeing all the given observations up until that point.

Mathematically:

$$\alpha_k(x) = \mathrm{P}(Y_1, \ldots, Y_k, X_k = x)$$

The Backward Algorithm

The backward algorithm can be thought of as the reverse of the forward algorithm. Here, we are interested in calculating the backward variable, which models the probability of seeing all of the future observations given that we are at a certain state at a past point. It differs slightly from the forward algorithm, as the forward algorithm models a joint probability distribution while the backward algorithm models a conditional probability distribution.

Mathematically:

$$\beta_k(x) = \mathrm{P}(Y_{k+1}, Y_{k+2}, \ldots, Y_t \mid X_k = x).$$

Bayes Filter

The Bayes Filter uses the above mentioned forward algorithm to solve the filtering problem. The result is the probability of seeing a certain state given all observations up to and including the one at the point we are estimating.

Mathematically:

$$\mathrm{P}(X_k = x \mid Y_1, \ldots, Y_k) = \frac{\mathrm{P}(X_k = x, Y_1, \ldots, Y_k)}{\mathrm{P}(Y_1, \ldots, Y_k)} = \eta\, \alpha_k(x)$$

Given that:

$$\eta = \left(\sum_x \alpha_k(x) \right)^{-1}.$$

Smoothing Calculation

Our smoothing calculation is done by using both the forward and backward algorithm, and allows us to calculate the probability of seeing certain states at each previous point, given the observations up to the current point. However, as stated before, these probability distributions are calculated independently, not jointly. For the joint probability, see Viterbi's Algorithm for decoding below.

Mathematically:

$$\mathrm{P}(X_k = x \mid Y_1, \ldots, Y_t) = \frac{\mathrm{P}(X_k = x, Y_1, \ldots, Y_t)}{\mathrm{P}(Y_1, \ldots, Y_t)}$$

$$= \frac{\beta_k(x)\, \alpha_k(x)}{\mathrm{P}(Y_1, \ldots, Y_t)}$$

Prediction Algorithm

The prediction algorithm can be done by using the calculations from the forward algorithm to give us a filtering estimate, which can then be propagated forward using the Markov Chain's transition matrix.

Mathematically:

$$\pi_t^f = \mathrm{P}(X_k = x \mid Y_1, \ldots, Y_t)$$

$$\pi_{i+1} = T' \pi_i.$$

Viterbi's Algorithm

Viterbi's Algorithm is used to solve the decoding problem. It gives us the probability of a given state trajectory given all of the observations we have. This differs from the smoothing calculations, as this models the joint probability of all of the states together, allowing us to estimate the most likely sequence of states given all of our observations.

Mathematically:

$$\delta_k(x) = \max_{(X_1,\ldots,X_{k-1})} \mathrm{P}(X_1 = X_1, \ldots, X_{k-1} = X_{k-1}, X_k = x, Y_1, \ldots, Y_k); \tag{2.13}$$

Learning HMM Through Observations

Given a HMM with our best guesses at the true transition matrix, observation matrix, and initial state probabilities, we can calculate our best state estimations given the observations we have seen. However, after making these calculations, we can go back and update our HMM, with the goal of maximizing the likelihood of seeing our given observations. The updated model learns a new transition matrix, a new observation matrix, and new initial state probabilities such that the probability of seeing our observations under this new model is greater than that of our original model.

Kalman Filters

Linear State Estimation

In a basic estimation problem, we have a random variable that is attempting to estimate some true state. This estimator can often be improved by adding a second estimator which improves the overall error covariance. Additionally, the estimators are often weighted, with more accurate estimators (smaller variance) receiving higher weights.

Kalman Gain

Expanding on the concept of linear state estimation, consider a sensor that gives observations:

$\mathbb{R}^p \ni Y = CX + \nu$, which is a linear function of the true state X with zero-mean Gaussian noise. Now, if we want to combine our estimate here with a previous estimator, we have linear combination:

$\hat{X} = K'\hat{X}' + KY.$. Like before, we want the estimator with minimal variance. The value of K that achieves this is known as the Kalman gain.

Kalman Filter

The Kalman filter is one of the most important algorithms in robotics. Just as with HMM, the goal of filtering here is to calculate the best state estimate at a point given a series of observations. Consider a linear dynamical system with linear observations:

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k + \epsilon_k \\ y_k &= Cx_k + \nu_k. \end{aligned}$$

$$\begin{aligned} \epsilon_k &\sim N(0, R) \\ \nu_k &\sim N(0, Q) \end{aligned}$$

Our goal is to compute:

$$\mathrm{P}(x_k \mid y_1, \ldots, y_k).$$.

Given that there are no longer a finite amount of states, we will model the possible states as a Gaussian random variable, and calculate the Kalman Filter from there.

Extended Kalman Filter

The Kalman Filter is optimal when dynamics/measurements are linear, which is often not the case. To handle these nonlinear problems, we have the Extended Kalman Filter, a version of the Kalman Filter that has been modified for this purpose. It works by taking a nonlinear system and linearizing it about a moving point. This moving point is given by the filter's last state estimate, and the linear system that results from this can be used to calculate the next state estimation.

Unscented Kalman Filter

While the EKF operates by assuming the transformed variable is normally distributed, this is not always the case. The Unscented Kalman Filter is a better way of dealing with nonlinear problems. It works by sampling a few points from the original distribution, known as sigma points, and then transforming each of those points using the nonlinear dynamics. It then computes the mean and covariance of these sigma points to try and get a better approximation of the distribution surrounding the transformed variable. The above process is known as the Unscented Transform, which can then be used to modify our EKF, giving us our new UKF algorithm.

Particle Filters

Particle Filters are a generalization of the UKF to filtering distributions that are non-normal. A core concept for particle filters is that of importance sampling. In importance sampling, we try and approximate a probability distribution by taking samples from a different probability distribution, and accounting for any differences by adding a weight to each sample, or particle, that we take. With enough samples, we can generate a distribution that closely approximates our target distribution. These particles are then resampled so that the weights are all even, with the goal of removing unused particles and splitting large particles into smaller, equally likely particles. These equally weighted particles are then fed into the Particle Filtering algorithm, with the same goal as the previous algorithms – approximating the probability of seeing different states at a point, given observations up to and including the preceding point.

Localization and Mapping

Transformations

In order to model real-world robots, we need to know about transformations. Transformations fall under two main categories – translations and rotations. Translations can be thought of as moving the robot while keeping it's orientation the same – for example, a car going from one end of the road to the other. It faces the same way, it's just moved it's location. Rotations are when you change the orientation of the robot, for example, if your Roomba runs into a wall and decides to turn itself around to keep going. The robot is still next to the wall, it just goes from facing the wall to facing the opposite of the wall. When moved to a 3D space, rotations get a little bit more complicated, since you can now rotate along each axis. The rotations about the x,y, and z axis are known as the roll, pitch, and yaw respectively, and any rotation in a 3D space can be thought of as a sequential application of each rotation. The rotation matrices for this are shown below.

$$R_x(\gamma) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix}$$

$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}$$

$$R_z(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbb{R}^{3\times 3} = R(\alpha, \beta, \gamma) = R_z(\alpha) R_y(\beta) R_x(\gamma)$$

The rotation angles are known as Euler angles, and can be back calculated from any given rotation matrix.

Mathematically:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix},$$

$$\alpha = \tan^{-1}(r_{21}/r_{11})$$
$$\beta = \tan^{-1}\left(-r_{31}/\sqrt{r_{32}^2 + r_{33}^2}\right)$$
$$\gamma = \tan^{-1}(r_{32}/r_{33}).$$

Quaternions

Instead of thinking of 3D rotations as three sequential rotations, we can think of it as one rotation done about an axis. This gives us the quaternion, a 4 dimensional vector that can be used to represent 3D rotations. Quaternions can be derived from rotation matrices and vice versa, they represent the same rotations, but quaternions are generally easier to work with in practice.