EE 8581 Detection and Estimation Theory
Spring Semester 2016

Tuesday March 1, 2016

Miniproject 1 — Due Friday April 1, 2016 by 11:55pm (central time)

Stochastic Filtering

Background

In class we discussed a general framework for *Stochastic Filtering*. For our purposes, we were interested in performing Bayesian inference to estimate (the posterior density of) a discrete-time, generally vector-valued, random process given random observations of the process at each time step. Under a few simplifying assumptions on the state dynamics (that describe the how the process to be estimated evolves over time), and the observation model (which quantifies the dependence of the observations on the current and past states and observations), we were able to derive a simple sequential update procedure.

More formally, our setting was as follows. Suppose that the process that we aim to estimate is denoted by $\{x_k\}_{k\geq 1}$, and that the initial state x_1 is drawn from a known prior, so that

$$x_1 \sim p_1(x_1).$$

We assume that the states evolve randomly in time, and while the dependence of the distribution of the current state can generally be on all previous states, we assume that the dependence here to satisfy a *Markovian* assumption, so that the dependence is only on the immediately previous state. Formally, for $k \geq 2$, we assume that

$$x_k \sim p(x_k|x_{k-1}, x_{k-2}, \ldots, x_1) = p(x_k|x_{k-1}) \triangleq S_k(x_k|x_{k-1}).$$

Our observations are denoted collectively by $\{y_k\}_{k\geq 1}$, and while each y_k generally could depend on the current state and all past states and observations, were assumed in our model that the dependence was only on the current state; specifically, we assumed that for $k \geq 1$,

$$y_k \sim p(y_k|y_{k-1}, \ldots, y_1, x_k, x_{k-1}, \ldots, x_1) = p(y_k|x_k) \triangleq L_k(y_k|x_k).$$

With this, we showed that the overall inference procedure may be interpreted as a sequence of "prediction" and "correction" steps. Qualitatively, the "prediction" step aims to estimate the posterior density of x_k using observations only up to and including step $k-1$. This results in a prediction density, given by

$$p(x_k|y_1, \ldots, y_{y-1}) \triangleq P_k(x_k|Y_{k-1}),$$

where $Y_k \triangleq \{y_1, y_2, \ldots, y_{k-1}\}$ for $k \geq 2$, and where the prediction density in the first step is simply the prior on x_1, $P_1(x_1) = p_1(x_1)$, since no "earlier" observations are available.

Then, the "correction" step makes use of y_k, the observations at the k-th step, to update the posterior. The result is the posterior density of x_k given observations at times up to and including step k, given by

$$p(x_k|y_1, \ldots, y_k) \triangleq F_k(x_k|Y_k)$$

for $k \geq 1$.

The explicit calculations that produce the quantities above are as follows:

First Step ($k = 1$): For the initial step, the prediction step is availed no observations at all, so we take

$$P_1(x_1) = p_1(x_1),$$

the prior on x_1. Then, the posterior is given by

$$F_1(x_1|Y_1) \propto L_1(y_1|x_1)P_1(x_1),$$

where the constant of proportionality is such that the posterior is a proper density.

Subsequent Steps ($k > 1$): For each other step, we take the prediction density to be

$$P_k(x_k|Y_{k-1}) = \int S_k(x_k|x_{k-1})F_{k-1}(x_{k-1}|Y_{k-1})dx_{k-1}.$$

This was derived in the class notes[1]. Then, analogously to above, we form the posterior as

$$F_k(x_k|Y_k) \propto L_k(y_k|x_k)P_k(x_k|Y_{k-1}),$$

where the constant of proportionality is again chosen so that the posterior is a proper density.

Interpretation

In addition to the prediction/correction interpretation discussed above, the update steps also admit an alternative, more intuitive interpretation.

Namely, the "prediction" step may be interpreted as describing what happens when random state variables drawn according to the posterior from the previous step are subjected to the state evolution process. The result is a distribution over "evolved" states that does not utilize any information from the *actual* evolved state. The "correction" step then updates the predictions using *specific* (noisy, uncertain) observations of the dynamic evolution of the actual state.

A specific and well-known example of this framework is the *Kalman Filter*. There, one assumes *(i)* the initial state vector is drawn according to a (multivariate) Gaussian prior, *(ii)* the state transition model describes each "new" state vector as a fixed linear transformation of the previous state vector, plus additive Gaussian noise, and *(iii)* the observation vector is a fixed linear transformation of the current state plus additive Gaussian noise. When the additive noises in the state transition are mutually independent and independent of the initial state, and the observation noises are mutually independent, all of the predicted densities and posteriors are (conditionally) Gaussian. Thus, the update equations effectively just propagate the corresponding mean vectors and covariance matrices – this, essentially, gives the Kalman Filter update equations! (See, in particular, pp. 448-449 of [1]).

From Densities to "Particles"

One of the most powerful aspects of the Stochastic Filtering approach outlined above is its flexibility – the same set of update equations (or analogous equations, when densities are replaced by probability mass functions) hold in general, regardless of the specific choices of the prior, likelihood, and the density (pmf) describing the state transition dynamics. (The Kalman Filter is but one example where the update equations have a particularly simple form.) In more general settings, such as when the state transition dynamics or observation likelihood models may be *nonlinear*, or when the prior is non-Gaussian, the simple updates given by the Kalman Filter equations are no longer valid. While many analytical alternatives have been developed for these more general settings (e.g., Extended Kalman Filter), a promising class of alternatives entails propagating a set of *samples* through the update equations (e.g., Unscented Kalman Filter, Point-Mass Filters,

[1] See also Rob Nowak's Notes linked here and here

and Particle Filters).

Here, we'll be using a particular particle-based approach based on the well-known *Condensation* algorithm from Computer Vision [2]. The essential idea may be summarized as follows:

Initialization: We begin by postulating a prior density over the initial state x_1, specifying a "forward" dynamics model that maps states x_k to states x_{k+1} (in general, randomly) for $k \geq 1$, and quantifying a function $L_k(y_k|x_k)$ that quantifies the likelihoods of the state x_k given observations y_k, for $k \geq 1$.

In practice, the dynamics model can be essentially described by a collection of functional mappings of the form

$$x_{k+1} = \mathcal{S}_k(x_k), \quad k \geq 1$$

that take an input x_k and produce a (random) output x_{k+1} corresponding to the "evolved" state. The likelihood functions can be specified as functional mappings of the form

$$\ell_k(x) = \mathcal{L}_k(x; y_k), \quad k \geq 1$$

that are parameterized by observations y_k, and produce a (non-negative) likelihood value for each input x. The prior need only be conceptual – to initialize, we draw some pre-specified number, say n, of realizations from this prior. These realizations, known as "particles" in what follows, are denoted by $\{x_1^{(i)}\}_{i=1}^n$.

Recall from the discussion above that the prior corresponds to a prediction density for the first step; our notation for the particles reflects this interpretation, in that the subscript denotes from which prediction density the particles are drawn, while the superscript indexes the individual particles.

First Step: In the first iteration of the algorithm, we make the first observation y_1 and compute the corresponding likelihoods $\ell_1(x_1^{(i)})$ for each particle $x_1^{(i)}$, for $i = 1, 2, \ldots, n$. The weights of the n particles may be interpreted as samples of the *posterior* of x_1 given y_1.

Subsequent Steps: Subsequent steps proceed somewhat analogously to the first step, but with one major difference owing to the fact that we don't process the *same* particles as above (or even the "weighted" particles that arise from the posterior from the previous step). Instead, at each step we must obtain particles (samples) drawn from the *prediction density*. In the first step the prediction density was just the prior (since no earlier observations were available). In subsequent steps, we need to obtain a *new* set of particles that are drawn from the prediction density at step k (given observations Y_{k-1}).

To this end, we must *resample* the collection of particles that we started with in such a way that the sampled particles occur with relative frequencies proportional to their likelihoods $\{\ell_{k-1}(x_{k-1}^{(i)})\}_{i=1}^n$ from the previous step[2]. Each "resampled" state is then passed through the dynamics model, yielding a new set of particles $\{x_k^{(i)}\}_{i=1}^n$ that can be interpreted as samples drawn from the prediction density (for x_k given Y_{k-1}).

Now, the process proceeds as in the first step – we make the observation y_k and compute the corresponding likelihoods $\ell_k(x_k^{(i)})$ for each particle $x_k^{(i)}$. At any step, the MMSE estimate (the mean of the posterior) may be approximated – for example, by renormalizing the likelihoods so they sum to one, and forming the empirical average of the states weighted by these renormalized likelihood values.

[2]This is easily accomplished in this scenario, as follows. First, renormalize the likelihoods for the n particles so that they sum to one (i.e., so that they form a proper pmf over the n particles). Next, form the cumulative distribution function of the particles; this will be a function of the index of the particles – 1 to n – and will take values between 0 to 1. Now, draw $U \sim$ uniform$[0, 1]$, and find the first index for which the cdf computed above is greater than or equal to U. This index will correspond to a draw from the (weighted, generally non-uniform) pmf over the particles.

Task 1

For this part of the miniproject, your aim will be to apply these ideas to perform tracking in a video sequence.

Specifically, you will apply the approach outlined above to track a 'Coke' can in a sequence of 2D video frames. The data we will use is from the benchmark data sets available at the 'Tracker Benchmark' website `https://sites.google.com/site/trackerbenchmark/benchmarks/v10`. The specific data set that we will consider for this part is available at `http://cvlab.hanyang.ac.kr/tracker_benchmark/seq/Coke.zip`

Here, your approach should adopt a state transition model where the next state is related to the previous state via a noisy version of simple kinematic relationships. More specifically, at each state $k \geq 1$, let

$$x_k = \begin{bmatrix} r_k \\ v_k \end{bmatrix}$$

be a 4-dimensional state vector where r_k is a two-dimensional vector whose coordinates correspond to the current position (here, corresponding to integer-valued 'spatial' coordinates in the domain of the image), and v_k is a two-dimensional vector whose entries correspond to the horizontal and vertical components of the current velocity (which may be positive or negative). Then, for $k \geq 1$ the state transition dynamics are described by the model

$$\begin{bmatrix} r_{k+1} \\ v_{k+1} \end{bmatrix} = \begin{bmatrix} I & \delta I \\ 0 & I \end{bmatrix} \begin{bmatrix} r_k \\ v_k \end{bmatrix} + \begin{bmatrix} \sigma_1 I & 0 \\ 0 & \sigma_2 I \end{bmatrix} u_k$$

where $u_k \sim \mathcal{N}(0, I)$, I denotes the identity matrix (of the appropriate dimensions), and $\delta, \sigma_1, \sigma_2 > 0$ are user-specified parameters[3]. The state x_1 is drawn from some conveniently-chosen prior (as alluded above).

You'll also need to devise a reasonable likelihood function to quantify (even heuristically) the extent to which each pixel in the image exhibits some degree of "Coke can-ness." One of the virtues of the particle-based approach is that this likelihood function need not correspond to any particular density...it only needs to assign likelihood values to pixels, and for this, can use any clever features you can dream up[4].

Overall, you are to do the following:

1. Implement the "particle-based" tracking method outlined above in `MATLAB`, using likelihoods computed from the `'redlikelihood.m'` function available on the course website, and the noisy kinematic model outlined above. Recall that when computing the particles drawn from the prediction density at each step, you'll need to also implement a script that performs the resampling (proportional to likelihood values) described above; each of these resampled particles is passed through the state evolution model, and then becomes a particle drawn from the prediction density. You should use the function `'plotstep.m'` to display the output at each step of your algorithm; here, the output consists of particles from the prediction density (the position particles r_k only), the prediction density particles weighted by the likelihood (position coordinates only), the current observations (image), and the observed image with a tracking square superimposed (corresponding to an estimate of the posterior mean obtained from the estimated posterior). Also, you should EDIT the `'plotstep.m'` file so that the title of the bottom right subfigure includes your own last name.

2. Create a *video* of the output of your tracker. This is relatively easy to do in `MATLAB`, as follows. First, before plotting the results of the first iteration of your procedure, issue the commands

```
v = VideoWriter('YOURNAME_Coke_video.avi');
open(v);
```

[3] In my code, I used $\delta = 2$, $\sigma_1 = 0.01$, and $\sigma_2 = 2$ with some success...

[4] For the example I showed in class, I used a function that assigned higher likelihoods to portions of the image that showed higher degrees of "redness," motivated by the idea that the red lettering on the can was a reasonable feature to track. Specifically, I used a heuristic notion of redness based on differences between the Red, Green, and Blue color panels of the observed image, and then "smeared" the resulting quantity with a Gaussian-like "blob" to allow for some spatial inaccuracies in the likelihood.

to create and open a new video object `'v'` (or whatever you care to name the object). You should name the output file as above, replacing 'YOURNAME' with your own last name.

You'll write a new frame to this video object after each step of your procedure using the following code:

```
plotstep(*particles*, *obs*, *likelihoods*, *stp*, *sidelen*);
fig = figure(1);
f = getframe(fig);
writeVideo(v,f);
```

Here, `*particles*, *obs*, *likelihoods*, *stp*, *sidelen*` are whatever the input variables to `'plotstep.m'` are called in your code for each step.

Finally, after plotting the output of the last step of your procedure, issue the command

```
close(v)
```

to close the video object and create the video file.

3. **Submit** your code to the Moodle site to the link titled '**MP1 Task 1 Code**'.

   You should submit your (working!) code compressed as a single `.zip` file. Further, your code should be "plug-and-play" for the Coke can data linked above, in the following sense: I should be able to decompress your scripts into a folder that contains a subdirectory `img` with the individual image files, and it should run! You shouldn't upload the image files themselves!

4. **Submit** your video to the Moodle site to the link titled '**MP1 Task 1 Video**'.

   You should use the code above, so your video file should be an `.avi` file. My code produces a video that is $\sim$ 7.6MB, so yours should likely be comparable, especially since you'll be using the same plotting script and video creation commands that I used.

Task 2

In this task you'll extend your method to perform tracking in one of the other data sets available at the 'Tracker Benchmark' website `https://sites.google.com/site/trackerbenchmark/benchmarks/v10`.

Specifically, choose *any* of the other video sequences, download it, and modify your approach from above to track the object of interest. You'll have to be creative in devising an appropriate likelihood function that assigns high values to the features of interest. If they're not based strictly on color (e.g., as in the example I provided for the approach above), then you might consider shape, contrast, etc.

You might also consider likelihood functions that compare with a known pattern, e.g., those that assign high values of likelihoods to image regions that correlate strongly with some pre-specified template, such as a specific person's face. In this case, you can assume that you start with some initial video frame from which you can extract this template; that is, take the first frame as "training data" from which you'll extract your template, and apply your tracker starting with the second frame.

For this task, you are to:

1. Implement the "particle-based" tracking method for a different data set from the 'Tracker Benchmark' website. You should use the function `'plotstep.m'` to display the output at each step of your algorithm; here, the output consists of particles from the prediction density (the position particles r_k only), the prediction density particles weighted by the likelihood (position coordinates only), the current observations (image), and the observed image with a tracking square superimposed (corresponding to an estimate of the posterior mean obtained from the estimated posterior). Also, you should again EDIT the `'plotstep.m'` file so that the title of the bottom right subfigure includes your own last name.

2. Create a *video* of the output of your tracker, as above.

3. **Submit** your code to the Moodle site to the link titled '**MP1 Task 2 Code**'.

   You should submit your (working!) code compressed as a single `.zip` file. Further, your code should again be "plug-and-play," in that I should be able to decompress your scripts into a folder that contains a subdirectory with the individual image files, and it should run! You shouldn't upload the image files themselves, but DO let me know which data set you used!

4. **Submit** your video to the Moodle site to the link titled '**MP1 Task 2 Video**'.

5. **Submit** a short (1-2 page) discussion of the approach that you applied to this problem to the Moodle site to the link titled '**MP1 Task 2 Discussion**'.

   Your discussion should describe, in particular, the likelihood function you implemented, and why you believe this is sensible for the tracking task you chose.

IMPORTANT NOTES

Your tracking code is to be developed by you individually. You may *discuss* your methodology with classmates, but the codes and videos you submit must be developed by each student individually! More to the point, we will be looking for codes that exhibit a high degree of overlap with another's code, and will give these submissions a grade of zero. Be on the safe side - write your code YOURSELF.

Also, on a related point, the website linked above contains code for many trackers from the literature. You are NOT to use these in your project – instead, you are to implement the approach outlined above, with your own clever likelihood function!

References

[1] S. M. Kay, "Fundamentals of statistical signal processing, volume i: Estimation theory," 1993.

[2] M. Isard and A. Blake, "Condensation – conditional density propagation for visual tracking," *International Journal of Computer Vision*, vol. 29, no. 1, pp. 5–28, 1998.