JSM talk

Marcin Jurek

August 5, 2020

Intro

Hello everyone, thanks for selecting my talk from among so many good JSM presentations. I've recently graduated from Texas A&M and this work would be impossible without the support of my advisor, Dr. Matthias Katzfuss, so many thanks to him

Intro filtering plots

Without further ado, let us dive right into the main topic. Since, as they say, a picture is worth a thousand words, let me briefly summarize what we are going to talk about in one animation. This short video, which you can see being played in a loop shows a realization of a stochastic process proposed by Lorenz in 2004 on 960 grid points distributed evenly on along a circle. Lorenz postulated that this process has many properties that atmospheric variables might have as measured along a latitudinal band. At each moment in time we select 10% of points and simulate data from a Poisson distribution with the intensity parameter determined by the value of the underlying blue process. They are represented by the yellow circles and their size corresponds to the value drawn.

To make this simulation more tangible let's imagine, without worrying too much about accuracy, that the Lorenz model captures value and changes in lightning potential and that observations denote the number of lightnings at each point. Within this framework, we would like to infer the lightning potential field given our knowledge of how it changes over time and the number of lightnings we observe. Moreover, we would like to be able to do it quickly, so that this inference can be done, ideally, in real time.

Pure space

Let us first assume we know nothing about how the process changes over time and let's treat each time point separately. For simplicity let's consider the case where our process of interest, x, is Gaussian and for starters let observations also be Gaussian. In this setup the most straightforward solution would be to use the analytic expressions for conditional mean and variance. However, as indicated by the red color, there is one term in these expressions that is a problem when the amount of data is very large. The covariance matrix of the vector of observatinons, y, will be very large and since matrix inversion has cubic complexity, this operation is prohibitvely expensive.

HV approximation

Obviously, this problem has been tackled in statistics before and a slew of solutions have been proposed. However, very few of them is suitable for filtering applications. So let me present one which, as we will see, is very suitable. So let's say that the vector x is composed of elements, each of which corresponds to one dot on the left-msot picture (solid or empty). Let us group the points and use a different color for each group.

First we select a few points which we mark in black. Second, we split the rest of our domain into several pieces, two pieces in this case as indicated by the black line in the middle picture, and then select another set of points from each half, red on the left, green on the right. In general we might iterate this process multiple times, but here let's just look at one mroe partition, along the horizontal direction, and select points from each quarter of the original domain: orange , yellow, and in two shades of green. In this way we build a hierarchy of points, with poins of the same color belonging to the same level of the hierarchy. Let's now see how we can use it to build an aproximation.

First, let us put all the elements of the vector in the order in which they were coloured. So the black nodes, labelled X, come first, then the red nodes, called X1 and so forth.

Now comes the approximation part. Let use a graphical model to illustrate it. In this graph, each vertex corresponds to a set of points (marked with the same colors). In graphical models language, a directed edge between two vertices means that the we can decompose the joint density such that it will have the term with the conditional density of the child variable given the parent. What a graph really encodes are independece relations. In this example the red variables are independent of the green variables given the black variables. Now this is where the approximation is made. In general, this might not be true. In terms of the upper picture, the graph says, that given the values of the process at the black dots, the values at the red dots are independent of the values at the green dots.

The key feature of this graph, and the key property of the HV approximation, is that each descendant node is a direct child of all its ancestors. What does it mean? Let's look at node X11. If we look only at the thick lines, X11 is a child of X1 and a grand child of X. However, like in human families, being someone's grandchild means that one is their descendant. Since in our graph all descendants are children, we also have an arrow from X to X11 meaning that X11 is also a child of X.

IC0

Okay this is the structure of the HV approximation. Let's switch gears now and talk about another concept and we will see how they come together later on. This other concept comes from the realm of numerical methods and is called the incomplete Cholesky decomposition. Now, most of us have probably heard of the Cholesky decomposition. Given a covariance matrix, it calculates a factor, such that its outer product with itself gives the original matrix. In statistics, it is often easier to work with Cholesky factor of a covariance matrix than with the matrix itself. The Cholesky factorization in general produces a dense lower-triangular matrix. Here we change the algorithm a little bit by introducing a matrix S, with elements being zeros or ones. S is lower triangular and if we use it in our algorithm, we would calculate an element of the Cholesky factor if the corresponding element of the S matrix is 1. Let's slowly go through the algoirhtm step by step and see how it works. So in general, we can run incomplete cholesky on some positive definite matrix A and a sparsity matrix S. We proceed over each row and each column under the diagonal, this is what lines 1 and 2 indicate. The actual calculation happens in the fourth line and the third line just tells us if we eun the fourth line or not. The fourth line is the standard formula for the element in the Cholesky factor. We end with the diagonal element in row 7.

HV + IC0

Okay, keeping this in mind let's see how we can combine both of these concepts. So let's see what the S matrix corresponding to the graph we had before would look like. In this case S will have it's i,j th element equal to zero, if i and j element of vector x are independent given the previous elements. So let us pick something in the first few columns. It's black, indicating 1, because the black nodes are direct parents of all other points. But as we go further more and more points are independent of each other given the preceding points, which are members of the ancestral groups. Finally, notice that the matrix S is very sparse (and gets sparse as the problem increases) which means that the number of elements in the incomplete cholesky factor that we need to calculate will be small. This allows us to calculate it quickly.

sparsity

Now here is where the key property of the HV is on full display. It turns out that the sparsity of S is preserved in several important matrices. First, in the top right picture we see the sparsity of the incomplete cholesky factor of the covariance matrix of Sigma. This is not surprising, given how we defined the incomplete cholesky factor. However, moving on the bottom left, its inverse has the same sparsity. This is a very unique property of this particular sparsity pattern but we can prove it holds. The key upshot of this property is fast computation. We already mentioned that calculating L is fact. But if it's inverse has few nonzero elements, and if we know where these elements are, we only need to calculate those

Another property worth mentioning, which we can also prove is that while L is an incomplete Cholesky factor of Sigma, it is an exact Cholesky factor of the covariance matrix of X, given our conditional indpendence assumptions specified before. Finally, even though it's less apparent and requires some technical assumptions, the matrix L tilde, which is a reverse cholesky factor of Lambda also has the same sparsity. We'll talk about it in more detail on the next slide.

matrix algebra

With all these pieces in mind, let's go through some matrix algebra and see how they help us solve the original problem. Let us recall that the original problem we encountered look as follows. We wanted to compute the moments of the posterior distribution of x and the key difficulty lied in inverting Lambda which, we said, could be a very large matrix. It turns out that after we perform the incomplete cholesky factorization and denote the factor as L, we can approximate the inverse of Lambda as an inner product of L inverse plus a product of H and R matrices, which are sparse. Using what we know about L and its iverse we conclude that our main bottleneck, Lambda inverse, can be rapidly calculated.

Now we proceed to a less obvious part and, for brevity, there will be a lot of detail that will remain hidden. Let us first recall that We defined L tilde to be a reverse cholesky factor of Lambda. But what is it? Well, if we define P to be an order reversing permutation matrix then L tilde can be defined as we see on the left side of the curly bracket. Now, once we have L tilde we can work through the matrix algebra and see that the outer product of L's is approximately the posterior covariance and that using Ls we can also calculate the posterior mean. However we were trying to accelerate the calculations and it might not be yet obiovus that we succeeded.

First, notice that if we have L tilde, then due to the sparsity of all the terms involved, posterior mean is not a problem. How about posterior covariance. Well, we don't necessarily have to calculate it at all. If we want to use it for confidence intervals, we can just calculate the norm of each of the columns of L, but we rarely need the entire matrix. Thus the only question that remains is how do we quickly compute L tilde. Well, I'm not going to discuss it in its whole depth, but let's just say that in order to calculate the reverse Cholesky factor of Lambda, which as it turns out is also sparse. Thus its computation and factorization doesn't take too long.

HV algo

Okay, so let's bring all the pieces together and see how we can do approximate posterior inference using HV. So let's start with what we are given: data y, sparsity pattern or approximation, encoded in S, the moments of the prior distirbution and H and R matrices, relating y to x. What we want is the approximate posterior distribution, which, as we said, can be represented by approximate mean and a sparse factor of the covariance matrix.

First we obtain the incomplete cholesky factor of prior covairance and in line 2 calculate its inverse. In the third line we compute the approximate matrix Lambda and in line 4 its reverse cholesky factor. This allows us to quickly compute the approximate mean in line 5. As a side note, let me just mention, that even though we discussed only the Gaussian case here, this method can be easily extended to non-Gaussian data. The way to do it was developed in Zilber and Katzfuss which shows how to do posterior inference when the likelihood is from an exponential family.

adding time

Okay, now that we know how to do inference at a single time point, let's see how we can incorporate our knowledge about the temporal dynamics. Let's say that using the domain knowledge we can define the funciton script E and, since we want everything to be Gaussian, we assume that whatever our model does not capture is normal with some known matrix Q. Using this framework, we are now interested in the filtering distribution of x t, i.e. the distribution of x conditional on all the observations up to that point. For completion, let's just observe that if our initial distirbution at time 0 was gaussian, every filtering distribution will be Gaussian as well.

EKF intro

The standard way to address this problem is the extended kalman filter based on the idea of recursion. Let's assume that before we get to time t, we already know the filtering distribution from time t-1. How can we use it? We'll proceed in two steps. First, using the knowledge about how the process changes, let us make our best guess about its current state. That's how we come up with the forecast mean. Then we linearize the temporal evolution operator around the previously derived mean, and use it to propagate the uncertainty captured by the covariance matrix. It is obviously an approximation, but for models without too much non-linearity, it works fine. And if the evolution function is linear, it is equal to the E matrix and the results are exact. One way or the other, Using our best guess as a prior distribution we then can carry our posterior inference. It's probably not hard to see how we can combine our previous findings with this filtering algorithm. Since exact posterior inference is costly, we will use an approximation.

EKVF algo

This brings us to our final algorithm, the extended Kalman Vecchia filter. Let's go through it step by step. We start with the sparsity matrix S, moments of the initial distribution at time zero, and then for each time point a vector of observations, an evolution function, covariance of the model errors, and the H and R matrices, relating observations and the latent state. The algorithm produces an approximate represenation of the filtering distribution at each time point, using a mean vector and a sparse factor of the covariance matrix. We start by decomposing the initial covariance and obtaining its sparse factor L00. then at each time point we linearize the evolution operator and then calculate the forecast mean and a forecast factor Ltt, simply by multiplying the latter by E. Next we calculate the elements of the forecast covariance matrix that will be used in posterior inference. As a final step, we use the hierarchical vecchia method discussed before to obtain the representation of the filtering distribution.

Alright, so that's it, that's how it all works. Now in order to see how accurate all these approximations we made are, we ran a number of simulations and looked at the difference in log scores between our method and some of the most prominent competitors.

simulations

These eight plots show the results of filtering using three different methods evaluated using functions of mean squared error and log score. We simulated the data from four exponential families with parameters determined by a realization from a Lorenz model with additive Gaussian noise. The lower value indicates better accuracy. The red line represents the score of an HV-based filter which outperforms its chief competitor, a low-rank filter.

thanks

Alright, this is it. I wanted to thank everyone for their attention and encourage to check out the full version of the paper available on arXiv.