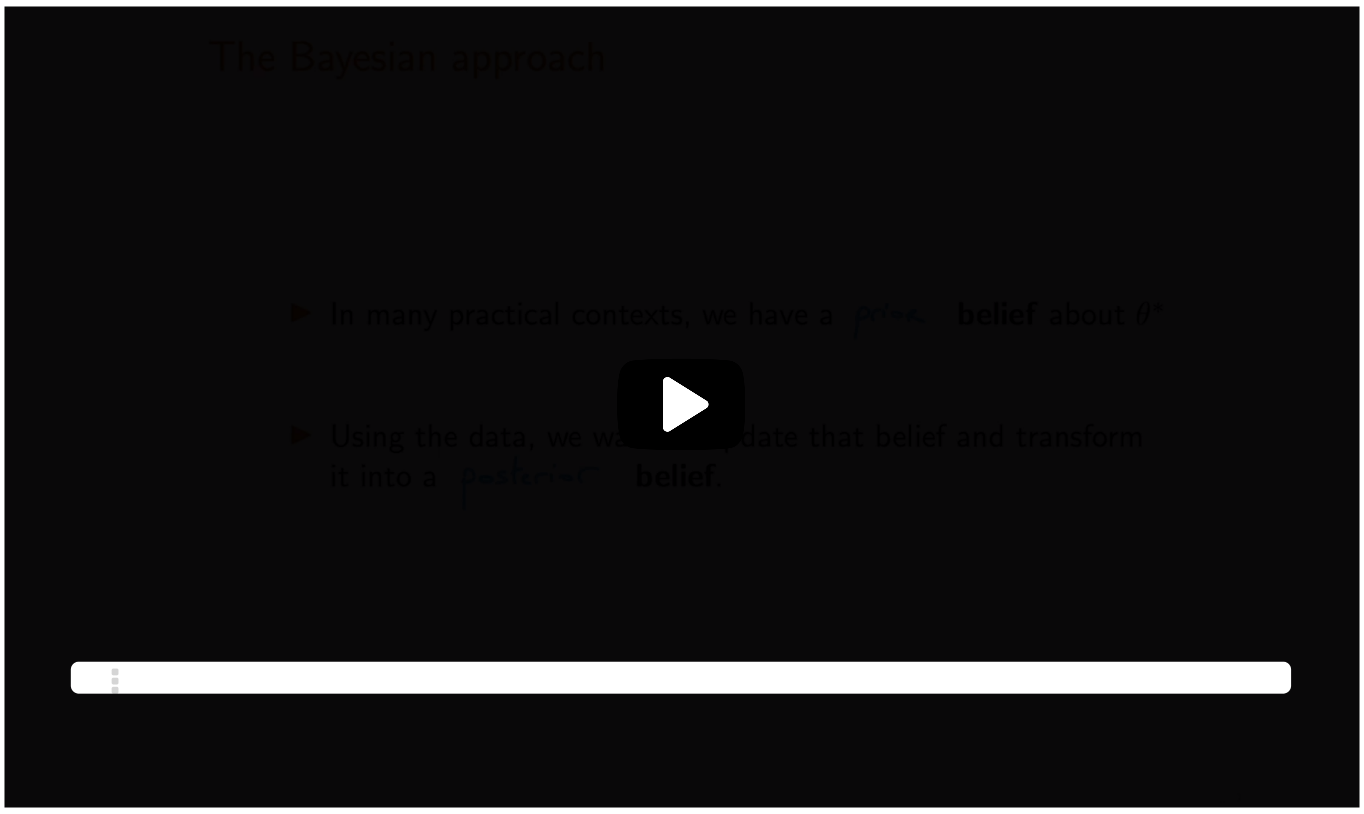
3. Introduction to the Bayesian Framework
Frequentist vs Bayesian Approaches



8:49 / 8:49 1.0x

I will never be able to move this.

But if you say, it's very likely to be close to one half,

and as I go away from one half, I

find it less and less likely to have this particular p,

then this will actually be updated from seeing data.

OK so using the data, what we want to do

is update that belief and transform it

into a posterior belief.

End of transcript. Skip to the start.

Video
Download video file

Transcripts
Download SubRip (.srt) file
Download Text (.txt) file

Frequentist vs. Bayesian I

1/1 point (graded)
Which of the following are aspects of **Bayesian** modeling approach, as opposed to the **frequentist** modeling approach? (Choose all that apply. Refer to the slides.)

- [x] In Bayesian statistics, the true parameter is modeled as a random variable, or at the very least, the uncertainty regarding the true parameter is modelled as such. ✔
- [ ] In most practical applications of Bayesian statistics, we are trying to estimate the true parameter only from the observation data and our chosen model.
- [x] In Bayesian statistics, we use the data to update our prior belief about a parameter and transform it into a posterior belief, which is reflected by a posterior distribution. ✔

✔

Solution:

We examine the choices in order.

- The first choice "**In Bayesian statistics, the true parameter is modeled as a random variable, or at the very least, the uncertainty regarding the true parameter is modelled as such**" is correct. In the Bayesian set-up, we model the true parameter as a random variable and update its distribution as we receive more data.

- The second choice "**In most practical applications of Bayesian statistics, we are trying to estimate the true parameter as accurately as possible as possible only from the observation data and our chosen model.**" is incorrect. In the Bayesian set-up, we do not even assume that there exists a true parameter, or at least we model it as a random variable to represent our uncertainty. This is rather the approach of frequentist statistics.

- The third choice "**In Bayesian statistics, we use the data to update our prior belief about a parameter and transform it into a posterior belief, which is reflected by a posterior distribution.**" is correct. Our prior belief is captured by the prior distribution on the parameter, and we can use Bayes' formula to update the prior distribution as we receive more data.

Submit You have used 2 of 2 attempts

Answers are displayed within the problem

Factors that Can be Specified in the Frequentist View

0/1 point (graded)
Suppose that we have some background information about our statistical problem, say from intuition or existing literature. We want to stick to a frequentist approach, but wish to specify our problem so that the outcome would be more in line with what we know so far. Which of the following components are we allowed to specify? (Choose all that apply.)

- [x] The set Θ possible parameters ✔
- [x] The probability model $\mathbb{P}_\theta$ ✔
- [x] A distribution $\pi(\theta)$ by which we weight the likelihood
- [ ] The procedure by which we would infer or estimate the true parameter based on the observations and model. (MLE, method of moments, M-estimator, etc.) ✔

✖

Solution:

- The first choice "**The set Θ possible parameters**" is correct, as it is part of the statistical model that can be specified regardless if we are in the frequentist or the Bayesian setting.

- The second choice "**The probability model $\mathbb{P}_\theta$**" is correct, as it is also part of the statistical model for the observations.

- The third choice "**A distribution $\pi(\theta)$ by which we weight the likelihood function**" is incorrect. Having a prior belief independent of the model or the observations, to be used as weights, is the core of the Bayesian approach.

- The fourth choice "**The procedure by which we would infer or estimate the true parameterbased on the observations and model**" is correct as frequentist inference or estimation procedures treat the parameter as a fixed, true value, not as a random variable as is done in the Bayesian approach.

Submit You have used 2 of 2 attempts

Answers are displayed within the problem

Discussion

Hide Discussion

Topic: Unit 5 Bayesian statistics:Lecture 17: Introduction to Bayesian Statistics / 3. Introduction to the Bayesian Framework

Add a Post

‹ All Posts

A computational reason for having a prior

discussion posted 4 days ago by ptressel

There's another reason for having a prior that comes out of machine learning, and is based not on the desire to include prior knowledge, but rather for computational tractability. That is the desire to be able to *update the model* as new data comes in, rather than have to rebuild it entirely from scratch if we get new data. A model that can be updated is, itself, a prior. Updating it yields a posterior.

Why is this desirable? It's because of the increasing amounts of data we have to deal with. If we can update a model, we can account for new data as it becomes available -- we don't have rebuild the model using all the data we got in the past.

And we may also be able to split up the data, send it out to be processed on separate computers, then take the results of each and combine them -- that combining process is also updating. But this generally requires the prior to have the same mathematical form as the posterior. For instance, for the kissing data, if I wanted to start with a prior belief that the probability of turning right was about a half, I would not make the prior just some continuous distribution with a peak at 0.5. Rather, I'd take the same model as I'd use to aggregate counts of right and left, and start with one that was "pre-loaded" with an equal number of right and left counts. The more of these prior counts I had, the more data counts I'd need to change the belief. If I have a prior like this, where the form remains the same after data is added, then I can do the trick of farming out work on batches of data to different computers. The problem with having a prior in a different form is that when I run batches of data separately, the effect of that different prior has been included in the results, and can't easily be backed out. When I combine the partial models, I get the effect of that prior included multiple times. If I have the same form of prior, I can either back out that prior before combining, or I can use an "empty" (non-informative) prior. For the kiss example, that would mean a count model with zero counts.

If a machine learning algorithm is capable of being updated, we refer to it as an *online* algorithm. If it has to be constructed (trained) all at once using the entire data, then it's called an *offline* or *batch* algorithm. Being able to split the training task actually goes beyond this, as it needs the prior to be in the same form as the posterior. If I only wanted an online algorithm, I could use (for the kiss example) any prior that was just a hump peaked at 0.5. But if I want a *distributed* algorithm, I'd represent the prior as counts.

This post is visible to everyone.

Add a Response

2 responses

ptressel
4 days ago

Hah! I just watched the next video. The beta distribution is exactly what I was referring to as a model for (binary) counts. You start with a beta distribution as the prior, and you end up with a beta distribution as the posterior. To farm the work out to many computers (e.g. to do it on a mapreduce framework), you just have each count up the number of right and left samples in its part of the data.

Note if you did happen to start with a prior that preferred some p by pre-loading some number of counts for right and left, you could easily back that out -- just subtract off those counts from the current model. That way, if someone else had a trained model with their own prior, and you had yours, you could combine them -- you'd each remove the other folks' prior before combining into your model.

Another reason why being able to combine models is useful is that it can allow the model to "forget" old data, and thus change with changing situations. E.g. for the counts (beta) model, one could checkpoint the model periodically -- save its state at each time, or rather, the difference between the previous checkpoint and the current state. Then, to "forget" the oldest batch of data, just subtract that checkpoint off.

Do you split the data to train it and model it?

posted 3 days ago by **risanchez**

risanchez --

Two answers, depending on interpretation of the question... 😉

If you just mean, which part of the machine learning pipeline was I talking about, yes, that's for training / model construction. Once the model is trained, it can be run on multiple servers -- usage of the model is a lot simpler to distribute, as there are no mathematical / algorithmic issues in the way -- just availability of resources.

If you mean splitting into training and test and validation sets, that's a tricky point, for multiple reasons.

For the initial model training, before one puts the model into production, one would have to split off the validation set up front, and set it aside. It would be used exactly once to get a final performance measure, after all the (distributed or otherwise) training was complete. After that, if the model is going to be updated online as new data comes in, then one would still want to sequester some data for checking, to make sure the model isn't going haywire.

Another issue during the initial training is that cross-validation is used to select the algorithm's settings. Splits for cross-validation during model training -- for picking the algorithm's settings, for instance -- could be done within each chunk of data sent off to be processed separately, or handled centrally, and just distribute the actual data processing for construction of a single cross-validation split. This raises a tricky point -- if the algorithm's settings are chosen differently per each distributed chunk of the data, is there any issue with combining them? It may be that any algorithm that even has tunable settings is not truly an online algorithm -- I'd have to go through lists of ML algorithms to is if they fall into camps that way.

And that brings up another possibility... We want to watch for problems after the ML model is deployed. One way to do that would be to train multiple models on different splits of the data...and then *not* combine them. Instead, send samples to be classified to the separate models, and if the results aren't close enough, call the production engineer's pager...

posted a day ago by **ptressel**

Add a comment