The Unofficial Google Data Science Blog

Fitting Bayesian structural time series with the bsts R package

July 11, 2017

by STEVEN L. SCOTT

Time series data are everywhere, but time series modeling is a fairly specialized area within statistics and data science. This post describes the bsts software package, which makes it easy to fit some fairly sophisticated time series models with just a few lines of R code.

Introduction

Time series data appear in a surprising number of applications, ranging from business, to the physical and social sciences, to health, medicine, and engineering. Forecasting (e.g. next month's sales) is common in problems involving time series data, but explanatory models (e.g. finding drivers of sales) are also important. Time series data are having something of a moment in the tech blogs right now, with Facebook announcing their "Prophet" system for time series forecasting (Taylor and Letham 2017), and Google posting about its forecasting system in this blog (Tassone and Rohani 2017).

This post summarizes the `bsts` R package, a tool for fitting Bayesian structural time series models. These are a widely useful class of time series models, known in various literatures as "structural time series," "state space models," "Kalman filter models," and "dynamic linear models," among others. Though the models need not be fit using Bayesian methods, they have a Bayesian flavor and the `bsts` package was built to use Bayesian posterior sampling.

The `bsts` package is open source. You can download it from CRAN with the R command `install.packages("bsts")`. It shares some features with Facebook and Google systems, but it was written with different goals in mind. The other systems were written to do "forecasting at scale," a phrase that means something different in time series problems than in

other corners of data science. The Google and Facebook systems focus on forecasting daily data into the distant future. The "scale" in question comes from having many time series to forecast, not from any particular time series being extraordinarily long. The bottleneck in both cases is the lack of analyst attention, so the systems aim to automate analysis as much as possible. The Facebook system accomplishes this using regularized regression, while the Google system works by averaging a large ensemble of forecasts. Both systems focus on daily data, and derive much of their efficiency through the careful treatment of holidays.

There are aspects of `bsts` which can be similarly automated, and a specifically configured version of bsts is a powerful member of the Google ensemble. However, `bsts` can also be configured for specific tasks by an analyst who knows whether the goal is short term or long term forecasting, whether or not the data are likely to contain one or more seasonal effects, and whether the goal is actually to fit an explanatory model, and not primarily to do forecasting at all.

The workhorse behind `bsts` is the structural time series model. These models are briefly described in the section **Structural time series models**. Then the software is introduced through a series of extended examples that focus on a few of the more advanced features of `bsts`. **Example 1: Nowcasting** includes descriptions of the local linear trend and seasonal state models, as well as spike and slab priors for regressions with large numbers of predictors. **Example 2: Long term forecasting** describes a situation where the local level and local linear trend models would be inappropriate. It offers a semilocal linear trend model as an alternative. **Example 3: Recession modeling** describes an model where the response variable is non-Gaussian. The goal in Example 3 is not to predict the future, but to control for serial dependence in an explanatory model that seeks to identify relevant predictor variables. A final section concludes with a discussion of other features in the package which we won't have space (maybe "time" is a better word) to explore with fully fleshed out examples.

Structural time series models

A structural time series model is defined by two equations. The *observation equation* relates the observed data y_t to a vector of latent variables α_t known as the "state."

$$y_t = Z_t^T \alpha_t + \epsilon_t.$$

The *transition equation* describes how the latent state evolves through time.

$$\alpha_{t+1} = T_t \alpha_t + R_t \eta_t.$$

The error terms ϵ_t and η_t are Gaussian and independent of everything else. The arrays Z_t, T_t and R_t are *structural parameters.* They may contain parameters in the statistical sense, but often they simply contain strategically placed 0's and 1's indicating which bits of α_t are relevant for a particular computation. An example will hopefully make things clearer.

The simplest useful model is the "local level model," in which the vector α_t is just a scalar μ_t. The local level model is a random walk observed in noise.

$$y_t = \mu_t + \epsilon_t$$

$$\mu_{t+1} = \mu_t + \eta_t.$$

Here $\alpha_t = \mu_t$, and Z_t, T_t, and R_t all collapse to the scalar value 1. Similar to Bayesian hierarchical models for nested data, the local level model is a compromise between two extremes. The compromise is determined by variances of $\epsilon_t \sim N(0, \sigma^2)$ and $\eta_t \sim N(0, \tau^2)$. If $\tau^2 = 0$ then μ_t is a constant, so the data are IID Gaussian noise. In that case the best estimator of y_{t+1} is the mean of $y_1, \ldots, y_t$. Conversely, if $\sigma^2 = 0$ then the data follow a random walk, in which case the best estimator of y_{t+1} is y_t. Notice that in one case the estimator depends on all past data (weighted equally) while in the other it depends only on the most recent data point, giving past data zero weight. If both variances are positive then the optimal estimator of y_{t+1} winds up being "exponential smoothing," where past data are forgotten at an exponential rate determined by the ratio of the two variances. Also notice that while the state in this model is Markov (i.e. it only depends on the previous state), the dependence among the observed data extends to the beginning of the series.

Structural time series models are useful because they are flexible and modular. The analyst chooses the structure of α_t based on things like whether short or long term predictions are more important, whether the data contains seasonal effects, and whether and how regressors are to be included. Many of these models are standard, and can be fit using a variety of tools, such as the `StructTS` function distributed with base R or one of several R packages for fitting these models (with the `dlm` package (Petris 2010, Petris, Petrone, and Campagnoli 2009) deserving special mention). The `bsts` package handles all the standard cases, but it also includes several useful extensions, described in the next few sections through a series of examples. Each example includes a mathematical description of the model and example `bsts` code showing how to work with the model using the `bsts` software. To keep things short, details about prior assumptions are largely avoided.

Example 1: Nowcasting

Scott and Varian (2014, 2015) used structural time series models to show how Google search data can be used to improve short term forecasts ("nowcasts") of economic time series. Figure 1 shows the motivating data set from Scott and Varian (2014), which is also included with the `bsts` package. The data consist of the weekly initial claims for unemployment insurance in the US, as reported by the US Federal Reserve. Like many official statistics they are released with delay and subject to revision. At the end of the week, the economic activity determining these numbers has taken place, but the official numbers are not published until several days later. For economic decisions based on these and similar numbers, it would help to have an early forecast of the current week's number as of the close of the week. Thus the output of this analysis is truly a "nowcast" of data that has already happened rather than a "forecast" of data that will happen in the future.

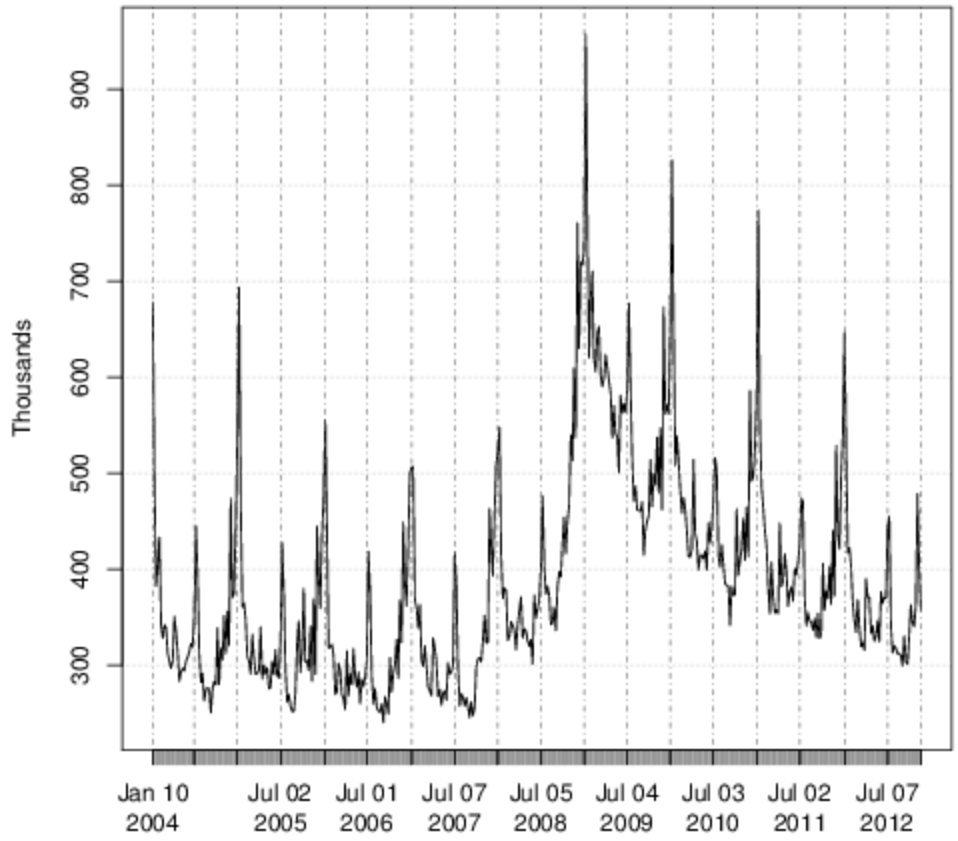


Figure 1: Weekly initial claims for unemployment in the US.

There are two sources of information about the current value y_t in the initial claims series: past values $y_{t-\tau}$ describing the time series behavior of the series, and contemporaneous

predictors $\mathbf{x}_t$ from a data source which is correlated with y_t, but which is available without the delay exhibited by y_t. The time series structure shows an obvious trend (in which the financial and housing crises in 2008 - 2009 are apparent) as well as a strong annual seasonal pattern. The external data source explored by Scott and Varian was search data from Google trends with search queries such as "how to file for unemployment" having obvious relevance.

State components

Scott and Varian modeled the data in Figure 1 using a structural time series with three state components: a trend μ_t, a seasonal pattern τ_t and a regression component $\beta^T \mathbf{x}_t$. The model is

$$y_t = \mu_t + \tau_t + \beta^T \mathbf{x}_t + \epsilon_t$$

$$\mu_{t+1} = \mu_t + \delta_t + \eta_{0t}$$

$$\delta_{t+1} = \delta_t + \eta_{1t}$$

$$\tau_{t+1} = -\sum_{s=1}^{S-1} \tau_t + \eta_{2t}\,.$$

The trend component looks similar to the local level model above, but it has an extra term δ_t. Notice that δ_t is the amount of extra μ you can expect as $t \to t+1$, so it can be interpreted as the slope of the local linear trend. Slopes normally multiply some x variable, but in this case $x = \Delta t$, which omitted from the equation because it is always 1. The slope evolves according to a random walk, which makes the trend an *integrated* random walk with an extra drift term. The local linear trend is a better model than the local level model if you think the time series is trending in a particular direction and you want future forecasts to reflect a continued increase (or decrease) seen in recent observations. Whereas the local level model bases forecasts around the average value of recent observations, the local linear trend model adds in recent upward or downward slopes as well. As with most statistical models, the extra flexibility comes at the price of extra volatility.

The best way to understand the seasonal component τ_t is in terms of a regression with seasonal dummy variables. Suppose you had quarterly data, so that $S = 4$. You might include the annual seasonal cycle using 3 dummy variables, with one left out as a baseline. Alternatively, you could include all four dummy variables but constrain their coefficients to sum to zero. The seasonal state model takes the latter approach, but the constraint is that the S most recent seasonal effects must sum to zero in expectation. This allows the seasonal pattern to slowly evolve. Scott and Varian described the annual cycle in the weekly initial claims data using a seasonal state component with $S = 52$. Of course weeks don't neatly divide years, but given the small number of years for which Google data are available the

occasional one-period seasonal discontinuity was deemed unimportant.

Let's ignore the regression component for now and fit a `bsts` model with just the trend and seasonal components.

```
library(bsts)     # load the bsts package
data(iclaims)     # bring the initial.claims data into scope

ss <- AddLocalLinearTrend(list(), initial.claims$iclaimsNSA)
ss <- AddSeasonal(ss, initial.claims$iclaimsNSA, nseasons = 52)
model1 <- bsts(initial.claims$iclaimsNSA,
               state.specification = ss,
               niter = 1000)
```

The first thing to do when fitting a `bsts` model is to specify the contents of the latent state vector α_t. The `bsts` package offers a library of state models, which are included by adding them to a state specification (which is just a list with a particular format). The call to `AddLocalLinearTrend` above adds a local linear trend state component to an empty state specification (the `list()` in its first argument). The call to `AddSeasonal` adds a seasonal state component with 52 seasons to the state specification created on the previous line. The state vector α_t is formed by concatenating the state from each state model. Similarly, the vector Z_t is formed by concatenating the Z vectors from the two state models, while the matrices T_t and R_t are combined in block-diagonal fashion.

The state specification is passed as an argument to `bsts`, along with the data and the desired number of MCMC iterations. The model is fit using an MCMC algorithm, which in this example takes about 20 seconds to produce 1000 MCMC iterations. The returned object is a list (with class attribute "`bsts`"). You can see its contents by typing `names(model1)`. The first few elements contain the MCMC draws of the model parameters. Most of the other elements are data structures needed by various S3 methods (`plot`, `print`, `predict`, etc.) that can be used with the returned object. MCMC output is stored in vectors (for scalar parameters) or arrays (for vector or matrix parameters) where the first index in the array corresponds to MCMC iteration number, and the remaining indices correspond to dimension of the deviate being drawn.

Most users won't need to look inside the returned `bsts` object because standard tasks like

plotting and prediction are available through familiar S3 methods. For example, there are several plot methods available.

```
plot(model)
plot(model1, "components")  # plot(model1, "comp") works too!
plot(model1, "help")
```

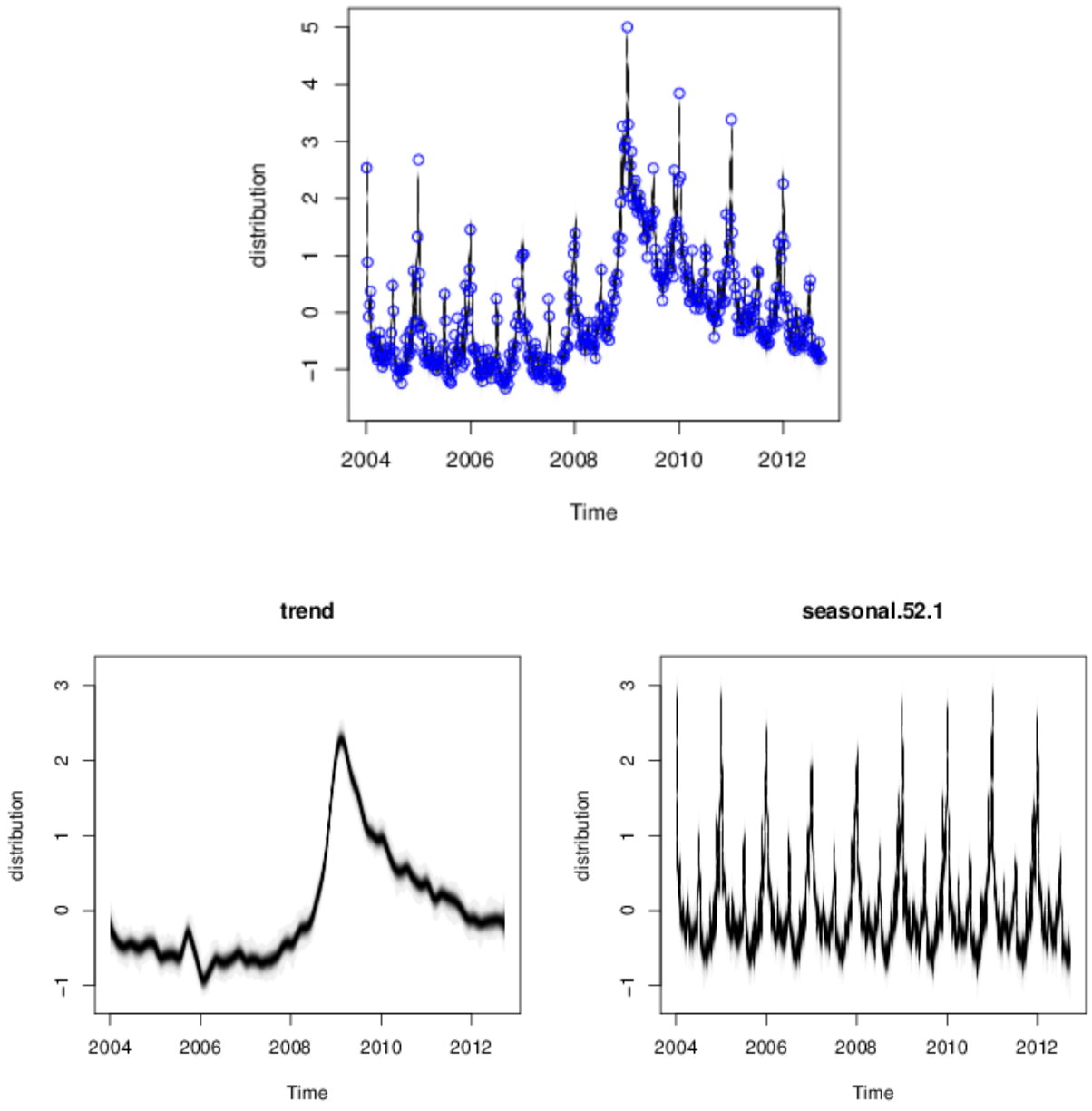


Figure 2: (top) Posterior distribution of model state. Blue circles are actual data points. (bottom) Individual state components. The plot looks fuzzy because it is showing the marginal posterior distribution at each time point.

The default `plot` method plots the posterior distribution of the conditional mean $Z_t^T \alpha_t$ given the full data $\mathbf{y} = y_1, \ldots, y_T$. Other plot methods can be accessed by passing a string to the plot function. For example, to see the contributions of the individual state components, pass the string `"components"` as a second argument, as shown above. Figure 2 shows the output of these two plotting functions. You can get a list of all available plots by passing the string `"help"` as the second argument.

To predict future values there is a `predict` method. For example, to predict the next 12 time points you would use the following commands.

```
pred1 <- predict(model1, horizon = 12)
plot(pred1, plot.original = 156)
```

The output of `predict` is an object of class `bsts.prediction`, which has its own `plot` method. The `plot.original = 156` argument says to plot the prediction along with the last 156 time points (3 years) of the original series. The results are shown in Figure 3.

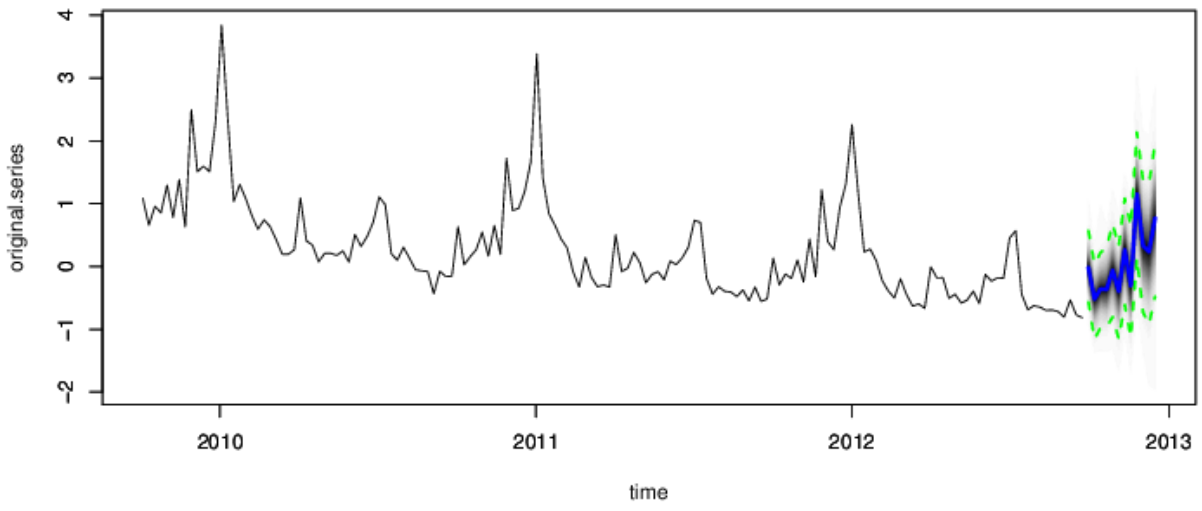


Figure 3: Posterior predictive distribution for the next 12 weeks of initial claims.

Regression with spike and slab priors

Now let's add a regression component to the model described above, so that we can use Google search data to improve the forecast. The `bsts` package only includes 10 search terms

with the initial claims data set, to keep the package size small, but Scott and Varian (2014) considered examples with several hundred predictor variables. When faced with large numbers of potential predictors it is important to have a prior distribution that induces sparsity. A spike and slab prior is a natural way to express a prior belief that most of the regression coefficients are exactly zero.

A spike and slab prior is a prior on a set of regression coefficients that assigns each coefficient a positive probability of being zero. Upon observing data, Bayes' theorem updates the inclusion probability of each coefficient. When sampling from the posterior distribution of a regression model under a spike and slab prior, many of the simulated regression coefficients will be exactly zero. This is unlike the "lasso" prior (the Laplace, or double-exponential distribution), which yields MAP estimates at zero but where posterior simulations will be all nonzero. You can read about the mathematical details of spike and slab priors in Scott and Varian (2014).

When fitting `bsts` models that contain a regression component, extra arguments captured by `...` are passed to the `SpikeSlabPrior` function from the `BoomSpikeSlab` package. This allows the analyst to adjust the default prior settings for the regression component from the `bsts` function call. To include a regression component in a `bsts` model, simply pass a model formula as the first argument.

```
# Fit a bsts model with expected model size 1, the default.
model2 <- bsts(iclaimsNSA ~ .,
               state.specification = ss,
               niter = 1000,
               data = initial.claims)


# Fit a bsts model with expected model size 5, to include more
coefficients.
model3 <- bsts(iclaimsNSA ~ .,
               state.specification = ss,
               niter = 1000,
               data = initial.claims,
               expected.model.size = 5)  # Passed to SpikeSlabPrior.
```

To examine the output you can use the same plotting functions as before. For example, to see the contribution of each state component you can type `plot(model2, "comp")`, producing the output in Figure 4. The regression component is explaining a substantial amount of variation in the initial claims series.

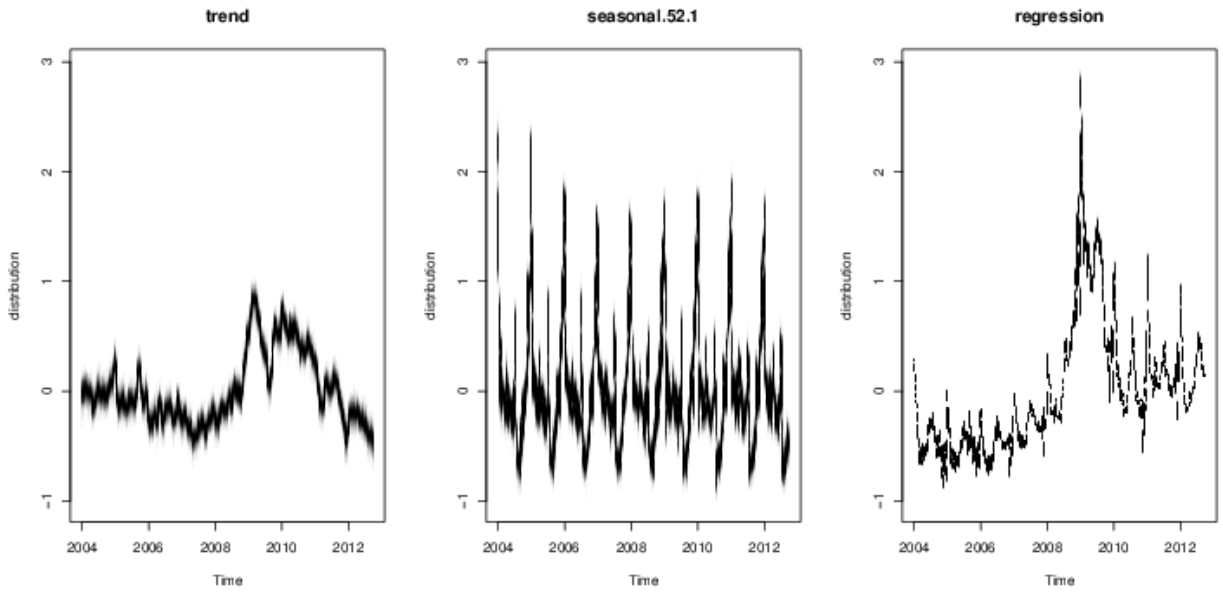


Figure 4: Contribution of each state component to the initial claims data, assuming a regression component with default prior. Compare to Figure 2.

There are also plotting functions that you can use to visualize the regression coefficients. The following commands

```
plot(model2, "coef")
plot(model3, "coef")
```

produce the summary plots in Figure 5. The search term "unemployment office" shows up with high probability in both models. Increasing the expected model size from 1 (the default) to 5 allows other variables into the model, though "Idaho unemployment" is the only one that shows up with high probability.

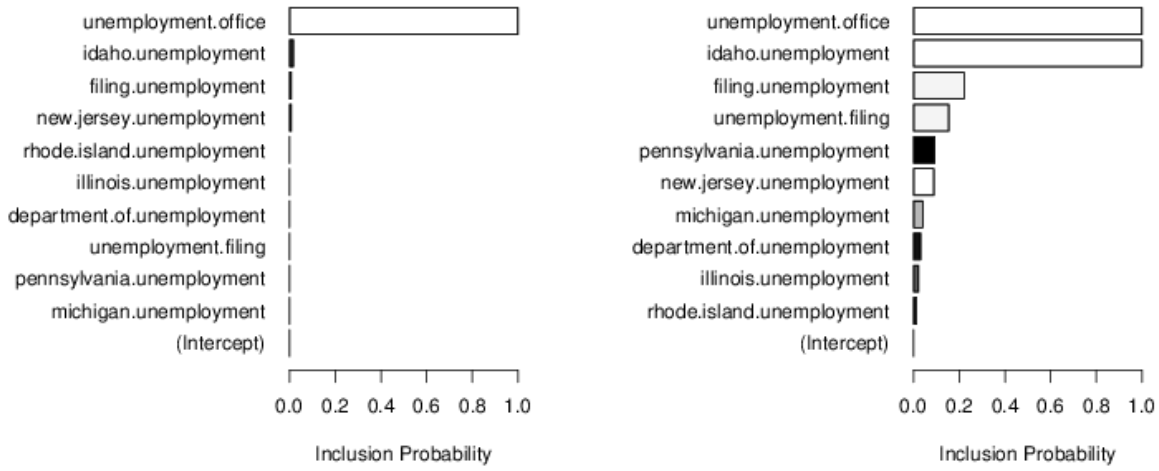


Figure 5: Posterior inclusion probabilities for predictors in the "initial claims" nowcasting example assuming an expected model size of (a) 1 and (b) 5.

Model diagnostics: Did the Google data help?

As part of the model fitting process, the algorithm generates the one-step-ahead prediction errors $y_t - E(y_t|Y_{t-1}, \theta)$, where $Y_{t-1} = y_1, \ldots, y_{t-1}$, and the vector of model parameters θ is fixed at its current value in the MCMC algorithm. The one-step-ahead prediction errors can be obtained from the `bsts` model by calling `bsts.prediction.errors(model1)`.

The one step prediction errors are a useful diagnostic for comparing several `bsts` models that have been fit to the same data. They are used to implement the function `CompareBstsModels`, which is called as shown below.

```
CompareBstsModels(list("Model 1" = model1,
                       "Model 2" = model2,
                       "Model 3" = model3),
                  colors = c("black", "red", "blue"))
```

The result of the call is the plot shown in Figure 6. The bottom panel shows the original series. The top panel shows the cumulative total of the mean absolute one step prediction errors for each model. The final time point in the top plot is proportional to the mean absolute prediction error for each model, but plotting the errors as a cumulative total lets you see particular spots

where each model encountered trouble, rather than just giving a single number describing each model's predictive accuracy. Figure 6 shows that the Google data help explain the large spike near 2009, where model 1 accumulates errors at an accelerated rate, but models 2 and 3 continue accumulating errors at about the same rate they had been before. The fact that the lines for models 2 and 3 overlap in Figure 6 means that the additional predictors allowed by the relaxed prior used to fit model 3 do not yield additional predictive accuracy.

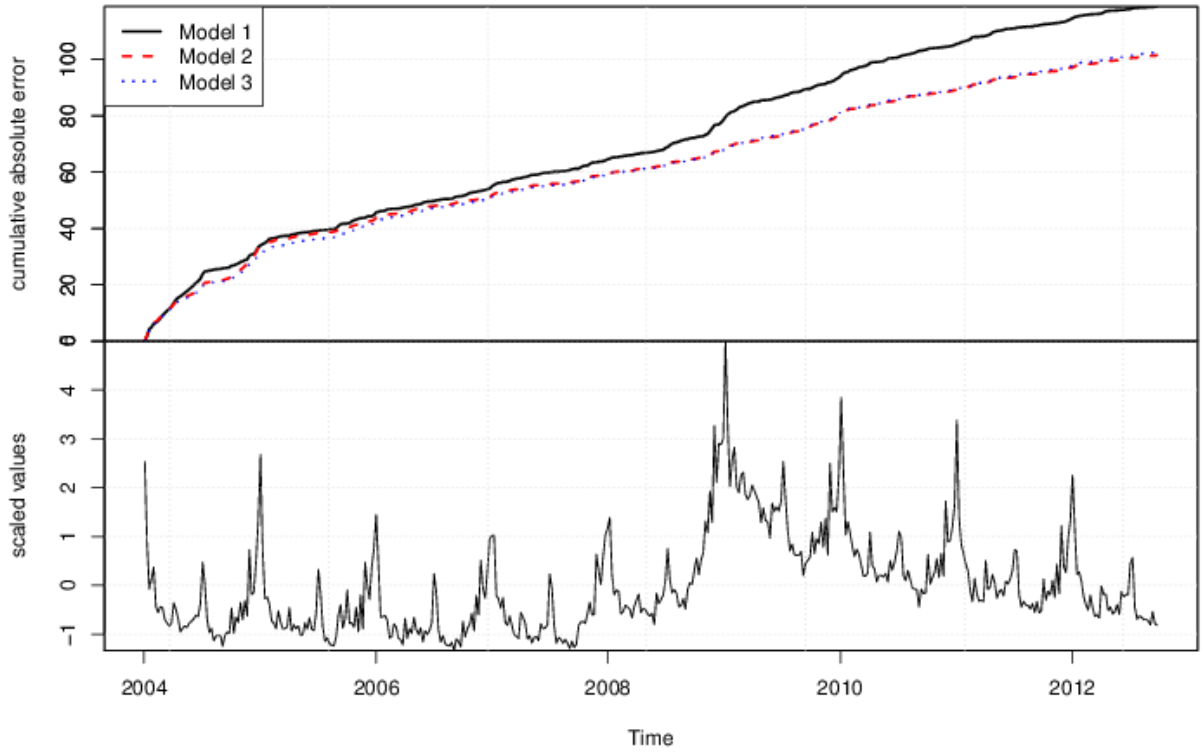


Figure 6: Comparing models 1 - 3 in terms of cumulative prediction error, for the nowcasting application.

Example 2: Long term forecasting

A common question about `bsts` is "which trend model should I use?" To answer that question it helps to know a bit about the different models that the `bsts` software package provides, and what each model implies.

In the local level model the state evolves according to a random walk:

$$\mu_{t+1} = \mu_t + \eta_t.$$

If you place your eye at time 0 and ask what happens at time t, you find that

$\mu_t \sim N(\mu_0, t\sigma_\eta^2)$. The variance continues to grow with t, all the way to $t = \infty$. The local linear trend is even more volatile. When forecasting far into the future the flexibility provided by these models becomes a double edged sword, as local flexibility in the near term translates into extreme variance in the long term.

An alternative is to replace the random walk with a stationary AR process. For example

$$\mu_{t+1} = \rho\mu_t + \eta_t,$$

with $\eta_t \sim N(0, \sigma_\eta^2)$ and $|\rho| < 1$. This model has stationary distribution

$$\mu_\infty \sim N\left(0, \frac{\sigma_\eta^2}{1-\rho^2}\right),$$

which means that uncertainty grows to a finite asymptote, rather than infinity, in the distant future. Bsts offers autoregressive state models through the functions `AddAr`, when you want to specify a certain number of lags, and `AddAutoAr` when you want the software to choose the important lags for you.

A hybrid model modifies the local linear trend model by replacing the random walk on the slope with a stationary AR(1) process, while keeping the random walk for the level of the process. The `bsts` package refers to this is the "semilocal linear trend" model.

$$\mu_{t+1} = \mu_t + \delta_t + \eta_{0t}$$

$$\delta_{t+1} = D + \rho(\delta_t - D) + \eta_{1t}$$

The D parameter is the long run slope of the trend component, to which δ_t will eventually revert. However δ_t can have short term autoregressive deviations from the long term trend, with memory determined by ρ. Values of ρ close to 1 will lead to long deviations from D. To see the impact this can have on long term forecasts, consider the time series of daily closing values for the S&P 500 stock market index over the last 5 years, shown in Figure 7.

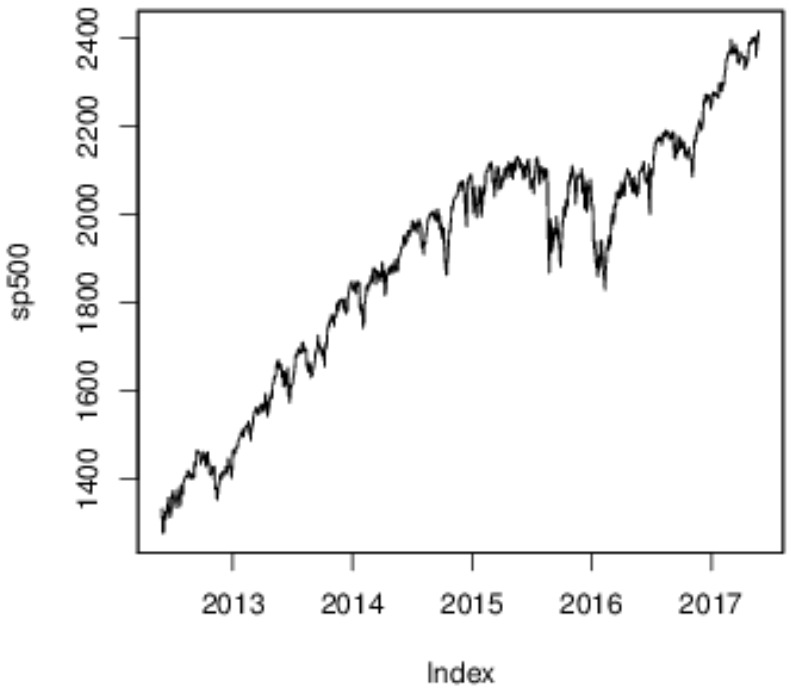


Figure 7: Daily closing values for the S&P 500 stock market index.

Consider two forecasts of the daily values of this series for the next 360 days. The first assumes the local linear trend model. The second assumes the semilocal linear trend.

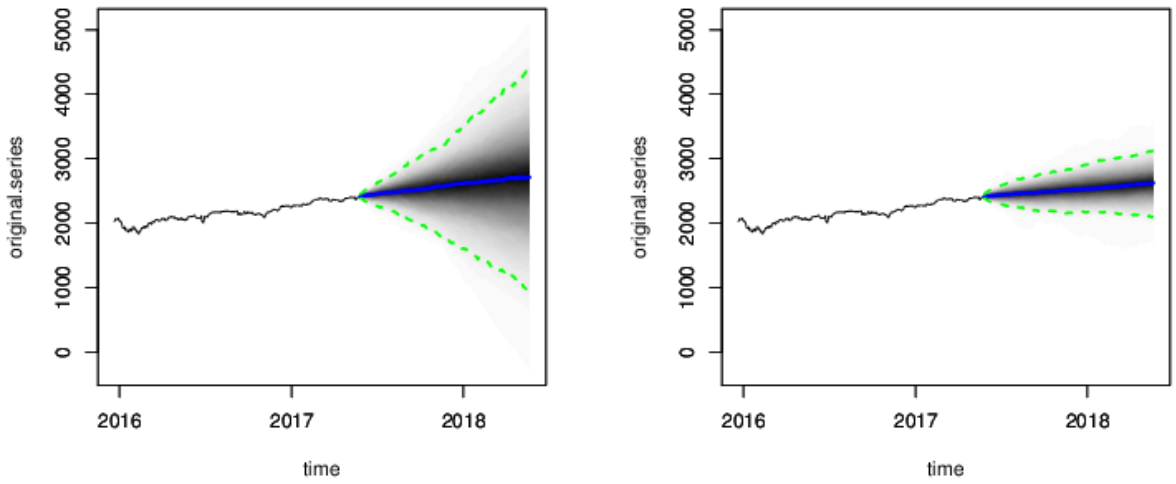
```
ss1 <- AddLocalLinearTrend(list(), sp500)
model1 <- bsts(sp500, state.specification = ss1, niter = 1000)
pred1 <- predict(model1, horizon = 360)
```

```
ss2 <- AddSemilocalLinearTrend(list(), sp500)
model2 <- bsts(sp500, state.specification = ss2, niter = 1000)
pred2 <- predict(model2, horizon = 360)
```

```
plot(pred2, plot.original = 360, ylim = range(pred1))
plot(pred1, plot.original = 360, ylim = range(pred1))
```

The resulting forecasts are plotted in Figure 8. The forecast expectations from the two models are quite similar, but the forecast errors from the local linear trend model are implausibly wide,

including a small but nonzero probability that the S&P 500 index could close near zero in the next 360 days. The error bars from the semilocal linear trend model are far more plausible, and more closely match the uncertainty observed over the life of the series thus far.



(a) (b)

Figure 8: Long term forecasts of the S&P 500 closing values under the (a) local linear trend and (b) semilocal linear trend state models.

Example 3: Recession modeling using non-Gaussian data

Although we have largely skipped details about how the `bsts` software fits models, the Gaussian error assumptions in the observation and transition equations are important for the model fitting process. Part of that process involves running data through the Kalman filter, which assumes Gaussian errors in both the state and transition equations. In many settings where Gaussian errors are obviously inappropriate, such as for binary or small count data, one can introduce latent variables that give the model a conditionally Gaussian representation. Well known "data augmentation" methods exist for probit regression (Albert and Chib, 1993) and models with student T errors (Gelman et al. 2014). Somewhat more complex methods exist for logistic regression (Frühwirth-Schnatter and Frühwirth 2005, Holmes and Held 2006, Gramacy and Polson 2012) and Poisson regression (Frühwirth-Schnatter et al 2008). Additional methods exist for quantile regression (Benoit and Van Den Poel 2012), support vector machines (Polson and Scott 2011), and multinomial logit regression (Frühwirth-Schnatter and Frühwirth 2010). These are not currently provided by the `bsts` package, but they

might be added in the future.

To see how non-Gaussian errors can be useful, consider the analysis done by Berge, Sinha, and Smolyansky (2016) who used Bayesian model averaging (BMA) to investigate which of several economic indicators would best predict the presence or absence of a recession. We will focus on their nowcasting example, which models the probability of a recession at the same time point as the predictor variables. Berge, Sinha, and Smolyansky (2016) also analyzed the data with the predictors at several lags.

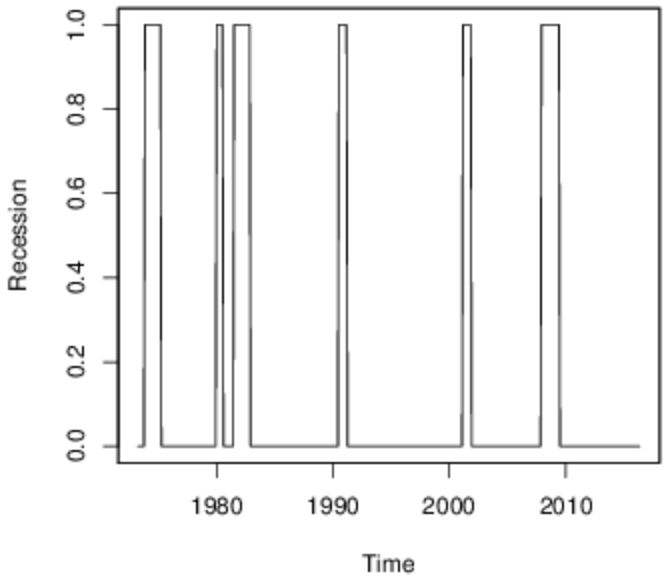


Figure 9: US recession indicators, as determined by NBER.

The model used in Berge, Sinha, and Smolyansky (2016) was a probit regression, with Bayesian model averaging used to determine which predictors should be included. The response variable was the the presence or absence of a recession (as determined by NBER), plotted in Figure 9. The BMA done by Berge, Sinha, and Smolyansky (2016) is essentially the same as fitting a logistic regression under a spike-and-slab prior with the prior inclusion probability of each predictor set to $1/2$. That analysis can be run using the `BoomSpikeSlab` R package (Scott 2010), which is similar to `bsts`, but with only a regression component and no time series. The marginal posterior inclusion probabilities produced by `BoomSpikeSlab` are shown in Figure 10(a). They largely replicate the findings of Berge, Sinha, and Smolyansky (2016), up to minor Monte Carlo error.

The logistic regression model is highly predictive, but it ignores serial dependence in the data.

To capture serial dependence, consider the following dynamic logistic regression model with a local level trend model.

$$\text{logit}(p_t) = \mu_t + \beta^T \mathbf{x}_t$$

$$\mu_{t+1} = \mu_t + \eta_t$$

Here p_t is the probability of a recession at time t, and $\mathbf{x}_t$ is the set of economic indicators used by Berge, Sinha, and Smolyansky (2016) in their analysis. To fit this model, we can issue the commands shown below.

```
## Because 'y' is 0/1 and the state is on the logit scale the default prior
## assumed by AddLocalLevel won't work here, so we need to explicitly set the
## priors for the variance of the state innovation errors and the initial value
## of the state at time 0.  The 'SdPrior' and 'NormalPrior' functions used to
## define these priors are part of the Boom package.  See R help for
## documentation.  Note the truncated support for the standard deviation of the
## random walk increments in the local level model.
ss <- AddLocalLevel(list(),
                    sigma.prior = SdPrior(sigma.guess = .1,
                                          sample.size = 1,
                                          upper.limit = 1),
                    initial.state.prior = NormalPrior(0, 5))


## Tell bsts that the observation equation should be a logistic regression by
## passing the 'family = "logit"' argument.
ts.model <- bsts(nber ~ ., ss, data = gdp, niter = 20000,a
                 family = "logit", expected.model.size = 10)
```

The marginal posterior inclusion probabilities under this model are shown in Figure 10(b). The top predictor is the same in both models, but posterior inclusion probabilities for the remaining predictors are smaller than in Figure 10(a). To understand why, consider the distribution of μ_t shown in Figure 11. The figure shows μ_t moving to very large values during a recession, and to very small values outside of a recession. This effect captures the strong serial dependence in the recession data. Recessions are rare, but once they occur they tend to persist. Assuming independent time points is therefore unrealistic, and it substantially overstates the amount of information available to identify logistic regression coefficients. The spike and slab priors that `bsts` uses to identify predictors naturally produce sparser models in the face of less information, which is why Figure 10(b) shows fewer included coefficients.

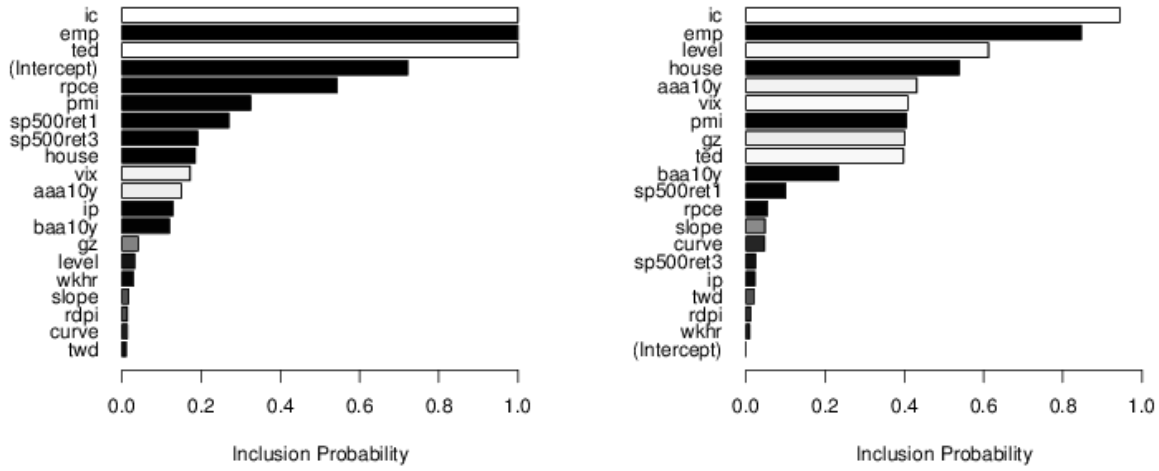


Figure 10: Regression coefficients for the (a) plain logistic regression model and (b) time series logistic regression model under equivalent spike and slab priors.

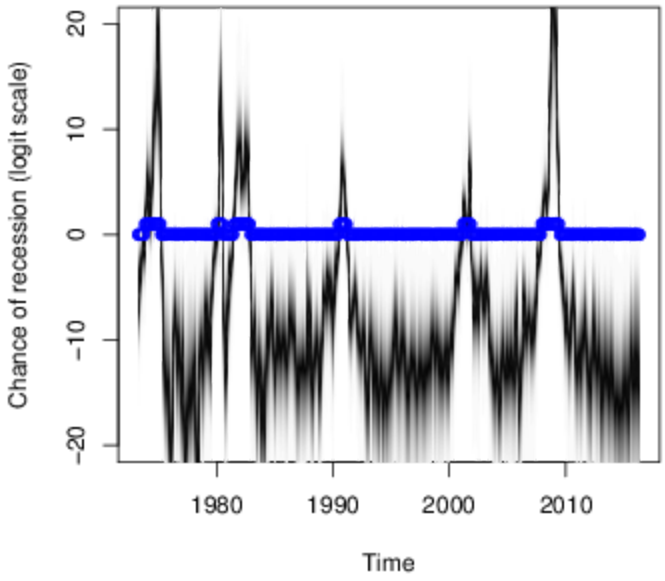


Figure 11: Distribution of state (on logit scale) for recession data. Blue dots show the true presence or absence of a recession, as determined by official statistics.

Conclusion

The preceding examples have shown that the `bsts` software package can handle several nonstandard, but useful, time series applications. These include the ability to handle large numbers of contemporaneous predictors with spike and slab priors, the presence of trend models suitable for long term forecasting, and the ability to handle non-Gaussian data. We have run out of space, but `bsts` can do much more.

For starters there are other state models you can use. Bsts has elementary support for holidays. It knows about 18 US holidays, and has capacity to add more, including holidays that occur on the same date each year, holidays that occur on a fixed weekday of a fixed month (e.g. 3rd Tuesday in February, or last Monday in November). The model for each holiday is a simple random walk, but look for future versions to have improved holiday support via Bayesian shrinkage.

Bsts offers support for multiple seasonalities. For example, if you have several weeks of hourly data then you will have an hour-of-day effect as well as a day-of-week effect. You can model these using a single seasonal effect with 168 seasons (which would allow for different hourly effects on weekends and weekdays), or you can assume additive seasonal patterns using the `season.duration` argument to `AddSeasonal`,

```
ss <- AddSeasonal(ss, y, nseasons = 24)
ss <- AddSeasonal(ss, y, nseasons = 7, season.duration = 24)
```

The latter specifies that each daily effect should remain constant for 24 hours. For modeling physical phenomena, `bsts` also offers trigonometric seasonal effects, which are sine and cosine waves with time varying coefficients. You obtain these by calling `AddTrig`. Time varying effects are available for arbitrary regressions with small numbers of predictor variables through a call to `AddDynamicRegression`.

In addition to the trend models discussed so far, the function `AddStudentLocalLinearTrend` gives a version of the local linear trend model that assumes student t errors instead of Gaussian errors. This is a useful state model for short term predictions when the mean of the time series exhibits occasional dramatic jumps. Student t errors can be introduced into the observation equation by passing the `family = "student"` argument to the `bsts` function call. Allowing for heavy tailed errors in the observation equation makes the model robust against individual outliers, while heavy tails in the state model provides robustness against sudden persistent shifts in level or slope. This can lead to tighter prediction limits than Gaussian models when modeling data that have been polluted by outliers. The observation equation can also be set to a Poisson model for small count data if desired.

Finally, the most recent update to `bsts` supports data with multiple observations at each time stamp. The Gaussian version of the model is

$$y_{it} = \beta^T \mathbf{x}_{it} + Z_t^T \alpha_t + \epsilon_{it}$$

$$\alpha_{t+1} = T_t \alpha_t + R_t \eta_t,$$

which is best understood as a regression model with a time varying intercept.

Bsts is a mature piece of software with a broad user base both inside and outside of Google. It is the product of several years of development, and I expect to continue improving it for the foreseeable future. I hope you find it useful.

References

Albert, J. H. and Chib, S. (1993). Bayesian analysis of binary and polychotomous response data. *Journal of the American Statistical Association* **88**, 669–679.

Benoit, D. F. and Van Den Poel, D. (2012). Binary quantile regression: A Bayesian approach based on the asymmetric Laplace distribution. *Journal of Applied Econometrics* **27**, 1174–1188.

Berge, T., Sinha, N., and Smolyansky, M. (2016). Which market indicators best forecast recessions? Tech. rep., US Federal Reserve.

Frühwirth-Schnatter, S. and Frühwirth, R. (2005). Auxiliary mixture sampling with applications to logistic models. Tech. rep., IFAS Research Paper Series, Department of Applied Statistics, Johannes Kepler University Linz.

Frühwirth-Schnatter, S. and Frühwirth, R. (2010). Data augmentation and MCMC for binary and multinomial logit models. In T. Kneib and G. Tutz, eds., *Statistical Modelling and Regression Structures – Festschrift in Honour of Ludwig Fahrmeir*, 111–132. Physica-Verlag, Heidelberg.

Frühwirth-Schnatter, S., Frühwirth, R., Held, L., and Rue, H. (2008). Improved auxiliary mixture sampling for hierarchical models of non-Gaussian data. *Statistics and Computing* **19**, 479.

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., and Rubin, D. B. (2014). *Bayesian Data Analysis*. Chapman & Hall, 3rd edn.

Gramacy, R. B. and Polson, N. G. (2012). Simulation-based regularized logistic regression. *Bayesian Analysis* **7**, 567–590.

Holmes, C. C. and Held, L. (2006). Bayesian auxiliary variable models for binary and multinomial regression. *Bayesian Analysis* **1**, 145–168.

Petris, G. (2010). An R package for dynamic linear models. *Journal of Statistical Software* **36**, 1–16.

Petris, G., Petrone, S., and Campagnoli, P. (2009). *Dynamic Linear Models with R.* useR! Springer-Verlag, New York.

Polson, N. G. and Scott, S. L. (2011). Data augmentation for support vector machines. *Bayesian Analysis* **6**, 1–24. (with discussion).

Scott, S. L. (2010). BoomSpikeSlab: MCMC for spike and slab regression. R package version 0.4.1.

Scott, S. L. and Varian, H. R. (2014). Predicting the present with Bayesian structural time series. *International Journal of Mathematical Modelling and Numerical Optimisation* **5**, 4–23.

Scott, S. L. and Varian, H. R. (2015). Bayesian variable selection for nowcasting economic time series. In A. Goldfarb, S. Greenstein, and C. Tucker, eds., *Economics of Digitization*, 119 –136. NBER Press, London.

Tassone, E. and Rohani, F. (2017). Our quest for robust time series forecasting at scale. http://www.unofficialgoogledatascience.com/2017/04/our-quest-for-robust-time-series.html.

Taylor, S. J. and Letham, B. (2017). Prophet: forecasting at scale. https://research.fb.com/prophet-forecasting-at-scale/

Rebecca *July 21, 2017 at 6:56 AM*

Very nice write up. I'm looking forward to trying out the support for multiple observations. Are there any plans to incorporate grouped or hierarchical structures into bsts? I'd love to see a cross between bsts/CausalImpact and hts.

REPLY

Ramesh Adavi *July 26, 2017 at 6:48 AM*

Wow. Finally a package that makes forecasting a well reasoned process!

REPLY

Richard W *August 14, 2017 at 5:20 AM*

Thank your this post and the work on the great package! For reproducibility: could you please tell me where to access the gdp data that you use in the logit-model? Thank you!

REPLY

Unknown *August 22, 2017 at 11:24 PM*

Great post - thanks for sharing the insights.

REPLY

Unknown *October 25, 2017 at 5:42 PM*

Great but can't produce reproducible runs even keeping the "seed" optionconstant within the bats function. ¿any idea why?

Rebecca *December 12, 2017 at 6:59 AM*

The predict function isn't setting a random seed. I've added that functionality in a forked version: https://github.com/rvessenes/bsts

Anonymous *December 12, 2017 at 7:41 AM*

Thanks. I'll make sure to set the seed in the next version pushed to CRAN.

REPLY

Rob *December 11, 2017 at 3:35 PM*

Thank you very much for this package. I have one question that I couldn't find the answer to anywhere - does this package take advantage of multi-core CPU setups?

REPLY

shushu *December 19, 2017 at 4:14 PM*

Brilliant!

REPLY

Anonymous *December 27, 2017 at 10:31 AM*

Thank you for sharing this package and knowledge. It is proving to be extremely useful in marketing data applications.

One challenging data set that I am working with has a mixture of steady state (advertising programs) and episodic (promotional sale days with both ads and email). If anyone has an approach to recommend for the type of model to handle the episodic data I would appreciate any advice.

REPLY

LE *December 29, 2017 at 9:03 AM*

Thanks for this! Is there a way to prevent the forecast from forecasting negative values? I don't think it's possible for unemployment claims to be negative

Unknown *January 11, 2018 at 12:17 PM*

Sure, you can apply a Box-Cox Transformation, e.g. lambda = 0 (Log-Transform), on the time series before fitting the model. After which you need the apply the inverse Box-Cox transformation on predictions. In that way you ensure that the forecasts and confidence intervals are positive

REPLY

Anonymous *January 14, 2018 at 2:29 PM*

I have been following the recent publication of large scale forecasting for google (on this blog), facebook prophet and others. The common technique appears to be separating out holiday, seasonal and trend effects. Including multi seasonal effects. BSTS appears to be a solution to this. I tried on some revenue data (daily for the last five years) and could not figure out how specify the dayofweek, weekly and quarterly seasons all with daily data. If I aggregate weekly data then weekly forecast works really well. Any thoughts as to why I can't do all of this in a single model and have the visibility into the components. It appears that the trend soaks up some of the weekly seasonal information when the data is daily. Would appreciate thoughts on how to to model with daily data.

REPLY

MM *April 6, 2018 at 6:39 AM*

Thank you for great package! I try to understand why the holiday handling in BSTS only for daily TS works! Or in other words what is the difficulty to implement it in a general form. Is it possible to give me a formal (mathematical) description of holiday state component!? Thank you so much in advanced!

REPLY

RonnieOneal *June 13, 2018 at 4:36 AM*

good post,thank you for this
REPLY

Unknown *July 24, 2018 at 11:20 AM*

Could you provides some examples for dynamic regression i.e. prediction? I have some trouble using it.

REPLY

To leave a comment, click the button below to sign in with Google.

SIGN IN WITH GOOGLE

Powered by Blogger

Archive