CausalImpact

An R package for causal inference using Bayesian structural time-series models

What does the package do?

This R package implements an approach to estimating the causal effect of a designed intervention on a time series. For example, how many additional daily clicks were generated by an advertising campaign? Answering a question like this can be difficult when a randomized experiment is not available.

How does it work?

Given a response time series (e.g., clicks) and a set of control time series (e.g., clicks in non-affected markets or clicks on other sites), the package constructs a Bayesian structural time-series model. This model is then used to try and predict the counterfactual, i.e., how the response metric would have evolved after the intervention if the intervention had never occurred. For a quick overview, watch the tutorial video. For details, see: Brodersen et al., Annals of Applied Statistics (2015).

What assumptions does the model make?

As with all non-experimental approaches to causal inference, valid conclusions require strong assumptions. In the case of CausalImpact, we assume that there is a set control time series that were *themselves not affected by the intervention.* If they were, we might falsely under- or overestimate the true effect. Or we might falsely conclude that there was an effect even though in reality there wasn't. The model also assumes that the relationship between covariates and treated time series, as established during the pre-period, remains stable throughout the post-period (see `model.args$dynamic.regression` for a way of relaxing this assumption). Finally, it's important to be aware of the *priors* that are part of the model (see `model.args$prior.level.sd` in particular).

How is the package structured?

The package is designed to make counterfactual inference as easy as fitting a regression model, but much more powerful, provided the assumptions above are met. The package has a single entry point, the function `CausalImpact()`. Given a response time series and a set of control time series, the function constructs a time-series model, performs posterior inference on the counterfactual, and returns a `CausalImpact` object. The results can be summarized in terms of a table, a verbal description, or a plot.

1. Installing the package

`CausalImpact` is available on CRAN and can be installed as follows in an R session:

```
install.packages("CausalImpact")
```

Once installed, the package can be loaded in a given R session using:

```
library(CausalImpact)
```

2. Creating an example dataset

To illustrate how the package works, we create a simple toy dataset. It consists of a response variable `y` and a predictor `x1`. Note that in practice, we'd strive for including many more predictor variables and let the model choose an appropriate subset. The example data has 100 observations. We create an *intervention effect* by lifting the response variable by 10 units after timepoint 71.

```
set.seed(1)
x1 <- 100 + arima.sim(model = list(ar = 0.999), n = 100)
y <- 1.2 * x1 + rnorm(100)
y[71:100] <- y[71:100] + 10
data <- cbind(y, x1)
```

We now have a simple matrix with 100 rows and two columns:

```
dim(data)
```

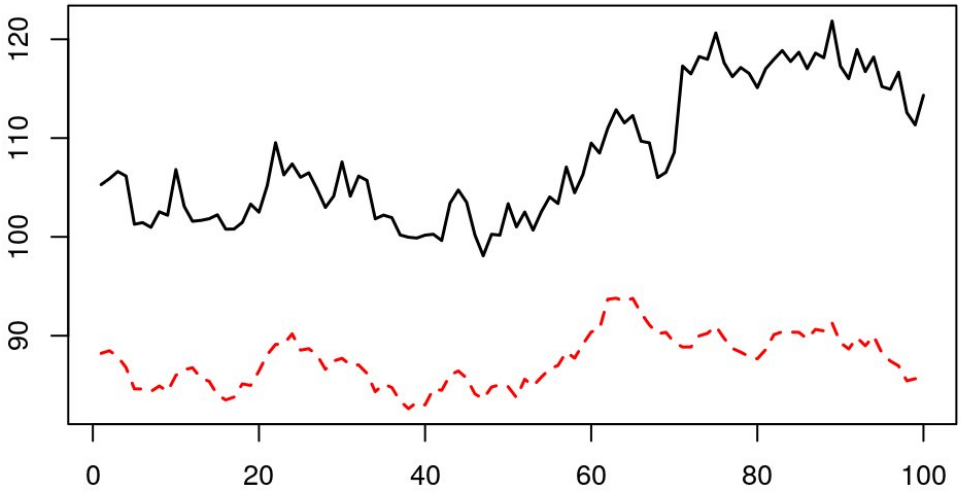
```
## [1] 100   2
```

```
head(data)
```

```
##             y       x1
## [1,] 105.2950 88.21513
## [2,] 105.8943 88.48415
## [3,] 106.6209 87.87684
## [4,] 106.1572 86.77954
## [5,] 101.2812 84.62243
## [6,] 101.4484 84.60650
```

We can visualize the generated data using:

```
matplot(data, type = "l")
```



3. Running an analysis

To estimate a causal effect, we begin by specifying which period in the data should be used for training the model (*pre-intervention period*) and which period for computing a counterfactual prediction (*post-intervention period*).

```
pre.period <- c(1, 70)
post.period <- c(71, 100)
```

This says that time points 1 … 70 will be used for training, and time points 71 … 100 will be used for computing predictions. Alternatively, we could specify the periods in terms of dates or time points; see Section 5 for an example.

To perform inference, we run the analysis using:

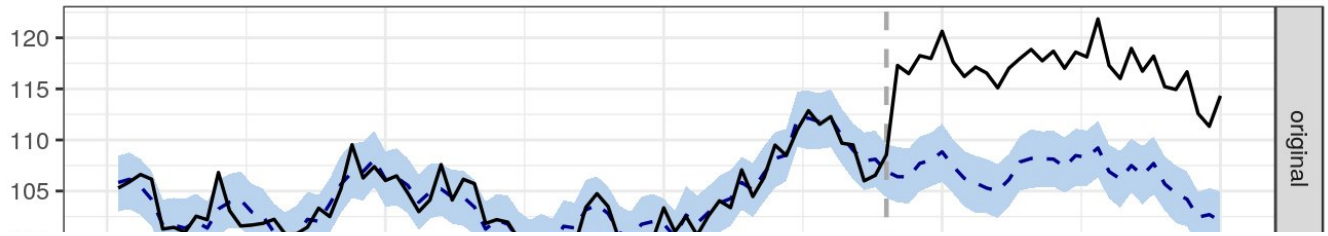
```
impact <- CausalImpact(data, pre.period, post.period)
```

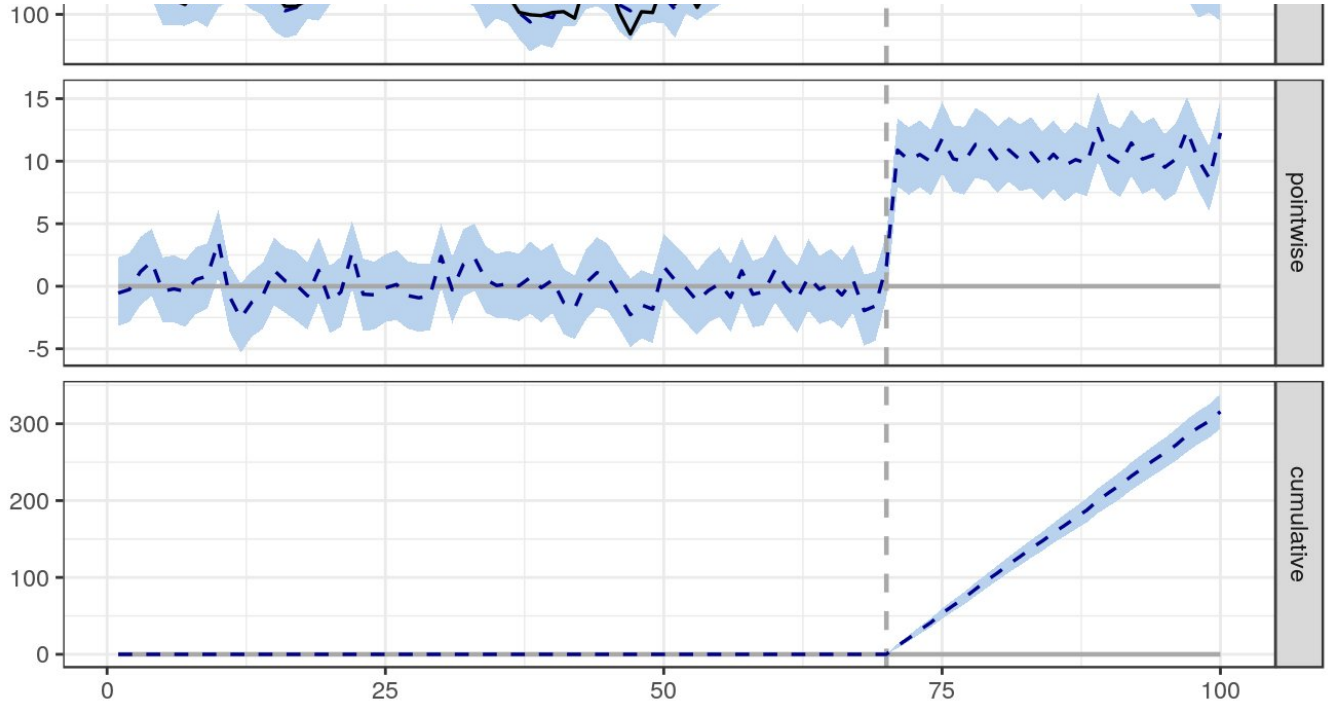
This instructs the package to assemble a structural time-series model, perform posterior inference, and compute estimates of the causal effect. The return value is a `CausalImpact` object.

4. Plotting the results

The easiest way of visualizing the results is to use the `plot()` function that is part of the package:

```
plot(impact)
```





By default, the plot contains three panels. The first panel shows the data and a counterfactual prediction for the post-treatment period. The second panel shows the difference between observed data and counterfactual predictions. This is the *pointwise* causal effect, as estimated by the model. The third panel adds up the pointwise contributions from the second panel, resulting in a plot of the *cumulative* effect of the intervention.

Remember, once again, that all of the above inferences depend critically on the assumption that the covariates were not themselves affected by the intervention. The model also assumes that the relationship between covariates and treated time series, as established during the pre-period, remains stable throughout the post-period.

5. Working with dates and times

It is often more natural to feed a time-series object into `CausalImpact()` rather than a data frame. For example, we might create a `data` variable as follows:

```
time.points <- seq.Date(as.Date("2014-01-01"), by = 1, length.out = 100)
data <- zoo(cbind(y, x1), time.points)
head(data)
```

```
##                   y       x1
## 2014-01-01 105.2950 88.21513
## 2014-01-02 105.8943 88.48415
## 2014-01-03 106.6209 87.87684
## 2014-01-04 106.1572 86.77954
## 2014-01-05 101.2812 84.62243
## 2014-01-06 101.4484 84.60650
```

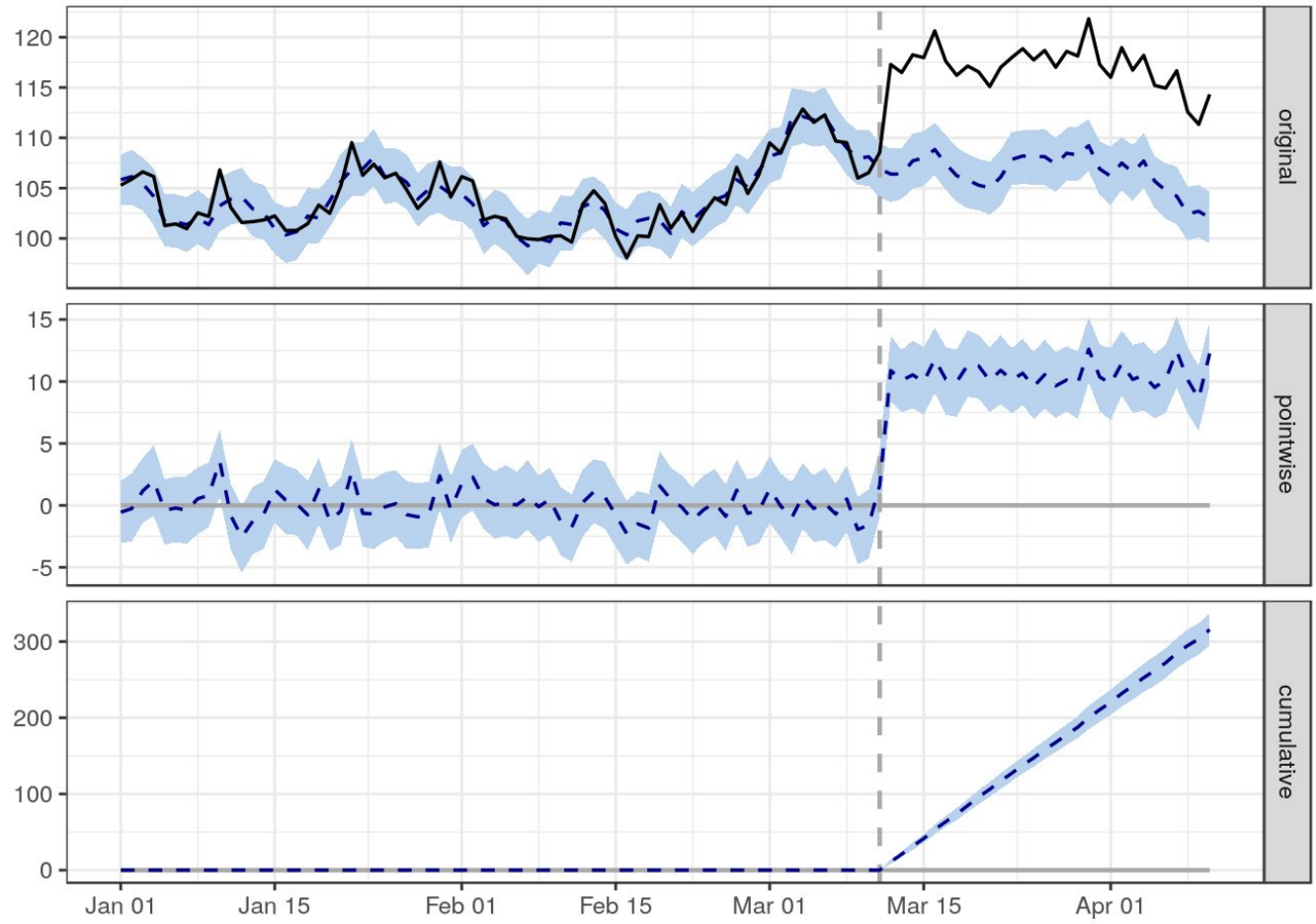
We can now specify the pre-period and the post-period in terms of time points rather than indices:

```
pre.period <- as.Date(c("2014-01-01", "2014-03-11"))
```

```
post.period <- as.Date(c("2014-03-12", "2014-04-10"))
```

As a result, the x-axis of the plot shows time points instead of indices:

```
impact <- CausalImpact(data, pre.period, post.period)
plot(impact)
```



6. Printing a summary table

To obtain a numerical summary of the analysis, we use:

```
summary(impact)
```

```
## Posterior inference {CausalImpact}
##
##                          Average        Cumulative
## Actual                   117            3511
## Prediction (s.d.)        107 (0.37)     3196 (11.03)
## 95% CI                   [106, 107]     [3174, 3217]
##
## Absolute effect (s.d.)   11 (0.37)      316 (11.03)
## 95% CI                   [9.8, 11]      [294.9, 337]
```

```
##
## Relative effect (s.d.)   9.9% (0.35%)   9.9% (0.35%)
## 95% CI                   [9.2%, 11%]    [9.2%, 11%]
##
## Posterior tail-area probability p:   0.001
## Posterior prob. of a causal effect:  99.9%
##
## For more details, type: summary(impact, "report")
```

The **Average** column talks about the average (across time) during the post-intervention period (in the example: time points 71 through 100). The **Cumulative** column sums up individual time points, which is a useful perspective if the response variable represents a flow quantity (such as queries, clicks, visits, installs, sales, or revenue) rather than a stock quantity (such as number of users or stock price).

In the example, the estimated average causal effect of treatment was 11 (rounded to a whole number; for full precision see `impact$summary`). This is because we observed an average value of 99 but would have expected an average value of only 89. The 95% posterior interval of the average effect is [9.8, 11]. Since this excludes 0, we (correctly) conclude that the intervention had a causal effect on the response variable. Since we generated the data ourselves, we know that we injected a true effect of 10, and so the model accurately recovered ground truth. One reason for this is that we ensured, by design, that the covariate `x1` was not itself affected by the intervention. In practice, we must always reason whether this assumption is justified.

For additional guidance about the correct interpretation of the summary table, the package provides a verbal interpretation, which we can print using:

```
summary(impact, "report")
```

The individual numbers in the table, at full precision, can be accessed using:

```
impact$summary
```

See below for tips on how to use these commands with *knitr* / *R Markdown*.

7. Adjusting the model

So far, we've simply let the package decide how to construct a time-series model for the available data. However, there are several options that allow us to gain a little more control over this process. These options are passed into `model.args` as individual list elements, for example:

```
impact <- CausalImpact(..., model.args = list(niter = 5000, nseasons = 7))
```

Available options

- `niter` Number of MCMC samples to draw. More samples lead to more accurate inferences. Defaults to **1000**.
- `standardize.data` Whether to standardize all columns of the data before fitting the model. This is

equivalent to an empirical Bayes approach to setting the priors. It ensures that results are invariant to linear transformations of the data. Defaults to **TRUE**.

- `prior.level.sd` Prior standard deviation of the Gaussian random walk of the local level. Expressed in terms of data standard deviations. Defaults to **0.01**, a typical choice for well-behaved and stable datasets with low residual volatility after regressing out known predictors (e.g., web searches or sales in high quantities). When in doubt, a safer option is to use **0.1**, as validated on synthetic data, although this may sometimes give rise to unrealistically wide prediction intervals.
- `nseasons` Period of the seasonal components. In order to include a seasonal component, set this to a whole number greater than 1. For example, if the data represent daily observations, use 7 for a day-of-week component. This interface currently only supports up to one seasonal component. To specify multiple seasonal components, use bsts to specify the model directly, then pass the fitted model in as bsts.model. Defaults to **1**, which means no seasonal component is used.
- `season.duration` Duration of each season, i.e., number of data points each season spans. Defaults to **1**. For example, to add a day-of-week component to data with daily granularity, use `model.args = list(nseasons = 7, season.duration = 1)`. To add a day-of-week component to data with hourly granularity, set `model.args = list(nseasons = 7, season.duration = 24)`.
- `dynamic.regression` Whether to include time-varying regression coefficients. In combination with a time-varying local trend or even a time-varying local level, this often leads to overspecification, in which case a static regression is safer. Defaults to **FALSE**.

8. Using a custom model

Instead of using the default model constructed by the CausalImpact package, we can use the bsts package to specify our own model. This provides the greatest degree of flexibility.

Before constructing a custom model, we set the observed data in the post-treatment period to NA, reflecting the fact that the counterfactual response is unobserved after the intervention. We keep a copy of the actual observed response in the variable `post.period.response`.

```
post.period <- c(71, 100)
post.period.response <- y[post.period[1] : post.period[2]]
y[post.period[1] : post.period[2]] <- NA
```

We next set up and estimate a time-series model using the bsts package. Here is a simple example:

```
ss <- AddLocalLevel(list(), y)
bsts.model <- bsts(y ~ x1, ss, niter = 1000)
```

Finally, we call `CausalImpact()`. Instead of providing input data, we simply pass in the fitted model object (`bsts.model`). We also need to provide the actual observed response. This is needed so that the package can compute the difference between predicted response (stored in `bsts.model`) and actual observed response (stored in `post.period.response`).

```
impact <- CausalImpact(bsts.model = bsts.model,
                       post.period.response = post.period.response)
```

The results can be inspected in the usual way:

```
plot(impact)
summary(impact)
summary(impact, "report")
```

9. FAQ

How do I cite the package in my work?

We recommend referencing the use of the CausalImpact R package as shown in the example below:

“CausalImpact 1.2.1, Brodersen et al., Annals of Applied Statistics (2015). http://google.github.io/CausalImpact/”

To find out which package version you are using, type `packageVersion("CausalImpact")`. See the bottom of this page for full bibliographic details.

How can I check whether the model assumptions are fulfilled?

It’s the elephant in the room with any causal analysis on observational data: how can we verify the assumptions that go into the model? Here are a few ways of getting started. First of all, it is critical to reason why the covariates that are included in the model (this was *x1* in the example) *were not themselves affected* by the intervention. Sometimes it helps to plot all covariates and do a visual sanity check. Next, it is a good idea to examine how well the outcome data *y* can be predicted *before the beginning of the intervention*. This can be done by running `CausalImpact()` on an imaginary intervention. Then check how well the model predicted the data following this imaginary intervention. We would expect not to find a significant effect, i.e., counterfactual estimates and actual data should agree reasonably closely. Finally, when presenting or writing up results, be sure to list the above assumptions explicitly, including the priors in `model.args`, and discuss them with your audience.

May the data contain missing values?

The response variable (i.e., the first column in data) may contain missing values (`NA`), but covariates (all other columns in data) may not. If one of your covariates contains missing values, consider imputing (i.e., estimating) the missing values; if this is not feasible, leave the regressor out.

How can I customize the default plot?

By default, `plot()` creates three panels, showing the counterfactual, pointwise, and cumulative impact estimates. One way of customizing the plot is to specify which panels should be included:

```
plot(impact, c("original", "pointwise"))
```

This creates a plot without cumulative impact estimates. This is sensible whenever the response variable

represents a stock quantity that cannot be meaningfully summed up across time (e.g., number of current subscribers), rather than a flow quantity (e.g., number of clicks).

How can I change the font size in the plot?

The `plot()` function for `CausalImpact` objects returns a ggplot2 object. This means we can customize the plot using standard ggplot2 functions. For example, to increase the font size, we can do:

```
library(ggplot2)
impact.plot <- plot(impact) + theme_bw(base_size = 20)
plot(impact.plot)
```

How can I obtain 90% intervals instead of 95% intervals?

The size of the intervals is specified by the argument `alpha`, which defaults to 0.05. To obtain 90% intervals instead, we would use:

```
impact <- CausalImpact(data, pre.period, post.period, alpha = 0.1)
```

Which predictor variables were used in the model?

Analyses may easily contain tens or hundreds of potential predictors (i.e., columns in the data function argument). Which of these were informative? We can plot the posterior probability of each predictor being included in the model using:

```
plot(impact$model$bsts.model, "coefficients")
```

10. Further resources

Literature

Brodersen KH, Gallusser F, Koehler J, Remy N, Scott SL. Inferring causal impact using Bayesian structural time-series models. *Annals of Applied Statistics*, 2015, Vol. 9, No. 1, 247-274.
http://research.google.com/pubs/pub41854.html

Discussion forum

http://stats.stackexchange.com/

Version 1.2.1

Authors: Kay H. Brodersen, Alain Hauser