Package ‘MarketMatching’

October 12, 2022

Type Package

Title Market Matching and Causal Impact Inference

Version 1.2.0

Date 2021-01-05

Description
For a given test market find the best control markets using time series matching and analyze the impact of an intervention. The intervention could be a marketing event or some other local business tactic that is being tested. The workflow implemented in the Market Matching package utilizes dynamic time warping (the 'dtw' package) to do the matching and the 'CausalImpact' package to analyze the causal impact. In fact, this package can be considered a ``workflow wrapper'' for those two packages. In addition, if you don't have a chosen set of test markets to match, the Market Matching package can provide suggested test/control market pairs and pseudo prospective power analysis (measuring causal impact at fake interventions).

Depends R (>= 3.5.0)

License GPL (>= 3)

Imports ggplot2, dplyr, utils, iterators, doParallel, parallel, foreach, reshape2, CausalImpact, tidyr, zoo, bsts, scales, Boom, utf8, dtw

LazyData true

VignetteBuilder knitr

Suggests knitr, rmarkdown

RoxygenNote 7.1.1

NeedsCompilation no

Author Larsen Kim [aut, cre]

Maintainer Larsen Kim <kblarsen4@gmail.com>

Repository CRAN

Date/Publication 2021-01-08 20:10:02 UTC

R topics documented:

`best_matches` *For each market, find the best matching control market*

Description

`best_matches` finds the best matching control markets for each market in the dataset using dynamic time warping (`dtw` package). The algorithm simply loops through all viable candidates for each market in a parallel fashion, and then ranks by distance and/or correlation.

Usage

```
best_matches(data=NULL,
             markets_to_be_matched=NULL,
             id_variable=NULL,
             date_variable=NULL,
             matching_variable=NULL,
             parallel=TRUE,
             warping_limit=1,
             start_match_period=NULL,
             end_match_period=NULL,
             matches=NULL,
             dtw_emphasis=1,
             suggest_market_splits=FALSE,
             splitbins=10,
             log_for_splitting=FALSE)
```

Arguments

`data` input data.frame for analysis. The dataset should be structured as "stacked" time series (i.e., a panel dataset). In other words, markets are rows and not columns – we have a unique row for each area/time combination.

`markets_to_be_matched`
Use this parameter if you only want to get control matches for a subset of markets or a single market The default is NULL which means that all markets will be paired with matching markets

`id_variable` the name of the variable that identifies the markets `date_variable` the time stamp variable

`matching_variable` the variable (metric) used to match the markets. For example, this could be sales or new customers

`parallel` set to TRUE for parallel processing. Default is TRUE

`warping_limit` the warping limit used for matching. Default is 1, which means that a single query value can be mapped to at most 2 reference values.

`start_match_period` the start date of the matching period (pre period). Must be a character of format "YYYY-MM-DD" – e.g., "2015-01-01"

`end_match_period` the end date of the matching period (pre period). Must be a character of format "YYYY-MM-DD" – e.g., "2015-10-01"

`matches` Number of matching markets to keep in the output (to use less markets for inference, use the control_matches parameter when calling inference). Default is to keep all matches.

`dtw_emphasis` Number from 0 to 1. The amount of emphasis placed on dtw distances, versus correlation, when ranking markets. Default is 1 (all emphasis on dtw). If emphasis is set to 0, all emphasis would be put on correlation, which is recommended when optimal splits are requested. An emphasis of 0.5 would yield equal weighting.

`suggest_market_splits` if set to TRUE, best_matches will return suggested test/control splits based on correlation and market sizes. Default is FALSE. For this option to be invoked, markets_to_be_matched must be NULL (i.e., you must run a full match). Note that the algorithm will force test and control to have the same number of markets. So if the total number of markets is odd, one market will be left out.

`splitbins` Number of size-based bins used to stratify when splitting markets into test and control. Only markets inside the same bin can be matched. More bins means more emphasis on market size when splitting. Less bins means more emphasis on correlation. Default is 10.

`log_for_splitting` This parameter determines if optimal splitting is based on correlations of the raw matching metric values or the correlations of log(matching metric). Only relevant if suggest_market_splits is TRUE. Default is FALSE.

Value

Returns an object of type `market_matching`. The object has the following elements:

`BestMatches` A data.frame that contains the best matches for each market. All stats reflect data after the market pairs have been joined by date. Thus SUMTEST and SUMCNTL can have smaller values than what you see in the Bins output table

`Data` The raw data used to do the matching

`MarketID` The name of the market identifier

`MatchingMetric` The name of the matching variable

DateVariable The name of the date variable

SuggestedTestControlSplits
Suggested test/control splits. SUMTEST and SUMCNTL are the total market volumes, not volume after joining with other markets. They're greater or equal to the values in the BestMatches file.

Bins Bins used for splitting and corresponding volumes

Examples

```
## Not run:
##-----------------------------------------------------------------------
## Find the best matches for the CPH airport time series
##-----------------------------------------------------------------------
library(MarketMatching)
data(weather, package="MarketMatching")
mm <- best_matches(data=weather,
                   id="Area",
                   markets_to_be_matched=c("CPH", "SFO"),
                   date_variable="Date",
                   matching_variable="Mean_TemperatureF",
                   parallel=FALSE,
                   start_match_period="2014-01-01",
                   end_match_period="2014-10-01")
head(mm$BestMatches)

## End(Not run)
```

---

inference *Given a test market, analyze the impact of an intervention*

Description

inference Analyzes the causal impact of an intervention using the CausalImpact package, given a test market and a matched_market object from the best_matches function. The function returns an object of type "market_inference" which contains the estimated impact of the intervention (absolute and relative).

Usage

```
inference(matched_markets=NULL,
          bsts_modelargs=NULL,
          test_market=NULL,
          end_post_period=NULL,
          alpha=0.05,
          prior_level_sd=0.01,
          control_matches=5,
          analyze_betas=FALSE,
          nseasons=NULL)
```

Arguments

`matched_markets`
: A matched_market object created by the market_matching function

`bsts_modelargs`
: A list() that passes model parameters directly to bsts – such as list(niter = 1000, nseasons = 52, prior.level.sd=0.1) This parameter will overwrite the values specified in prior_level_sd and nseasons. ONLY use this if you're using intricate bsts settings For most use-cases, using the prior_level_sd and nseasons parameters should be sufficient

`test_market`
: The name of the test market (character)

`end_post_period`
: The end date of the post period. Must be a character of format "YYYY-MM-DD" – e.g., "2015-11-01"

`alpha`
: Desired tail-area probability for posterior intervals. For example, 0.05 yields 0.95 intervals

`prior_level_sd`
: Prior SD for the local level term (Gaussian random walk). Default is 0.01. The bigger this number is, the more wiggliness is allowed for the local level term. Note that more wiggly local level terms also translate into larger posterior intervals This parameter will be overwritten if you're using the bsts_modelargs parameter

`control_matches`
: Number of matching control markets to use in the analysis (default is 5)

`analyze_betas`
: Controls whether to test the model under a variety of different values for prior_level_sd.

`nseasons`
: Seasonality for the bsts model – e.g., 52 for weekly seasonality

Value

Returns an object of type `inference`. The object has the following elements:

`AbsoluteEffect`
: The estimated absolute effect of the intervention

`AbsoluteEffectLower`
: The lower limit of the estimated absolute effect of the intervention. This is based on the posterior interval of the counterfactual predictions. The width of the interval is determined by the `alpha` parameter.

`AbsoluteEffectUpper`
: The upper limit of the estimated absolute effect of the intervention. This is based on the posterior interval of the counterfactual predictions. The width of the interval is determined by the `alpha` parameter.

`RelativeEffectLower`
: Same as the above, just for relative (percentage) effects

`RelativeEffectUpper`
: Same as the above, just for relative (percentage) effects

`TailProb`
: Posterior probability of a non-zero effect

`PrePeriodMAPE`
: Pre-intervention period MAPE

`DW`
: Durbin-Watson statistic. Should be close to 2.

`PlotActualVersusExpected`
: Plot of actual versus expected using `ggplot2`

`PlotCumulativeEffect`
: Plot of the cumulative effect using `ggplot2`

`PlotPointEffect`
: Plot of the pointwise effect using `ggplot2`

`PlotActuals`
: Plot of the actual values for the test and control markets using `ggplot2`

`PlotPriorLevelSdAnalysis`
: Plot of DW and MAPE for different values of the local level SE using `ggplot2`

`PlotLocalLevel`
: Plot of the local level term using `ggplot2`

`TestData`
: A `data.frame` with the test market data

`ControlData`
: A `data.frame` with the data for the control markets

`PlotResiduals`
: Plot of the residuals using `ggplot2`

`TestName`
: The name of the test market

`TestName`
: The name of the control market

`zooData`
: A `zoo` time series object with the test and control data

`Predictions`
: Actual versus predicted values

`CausalImpactObject`
: The CausalImpact object created

`Coefficients`
: The average posterior coefficients

Examples

```
## Not run:
library(MarketMatching)
##-----------------------------------------------------------------------
## Analyze causal impact of a made-up weather intervention in Copenhagen
## Since this is weather data it is a not a very meaningful example.
## This is merely to demonstrate the function.
##-----------------------------------------------------------------------
data(weather, package="MarketMatching")
mm <- best_matches(data=weather,
                   id="Area",
                   markets_to_be_matched=c("CPH", "SFO"),
                   date_variable="Date",
                   matching_variable="Mean_TemperatureF",
                   parallel=FALSE,
                   warping_limit=1, # warping limit=1
                   dtw_emphasis=1, # rely only on dtw for pre-screening
                   matches=5, # request 5 matches
                   start_match_period="2014-01-01",
                   end_match_period="2014-10-01")
library(CausalImpact)
results <- inference(matched_markets=mm,
                     test_market="CPH",
                     analyze_betas=FALSE,
                     control_matches=5, # use all 5 matches for inference
```

```
                     end_post_period="2015-12-15",
                     prior_level_sd=0.002)

## End(Not run)
```

MarketMatching *Market Matching and Causal Impact Inference*

Description

For a given test market find the best matching control markets using time series matching and analyze the impact of an intervention (prospective or historical). The intervention could be be a marketing event or some other local business tactic that is being tested. The package utilizes dynamic time warping to do the matching and the CausalImpact package to analyze the causal impact. In fact, MarketMatching is simply a wrapper and workflow for those two packages. MarketMatching does not provide any functionality that cannot be found in these packages but simplifies the workflow of using dtw and CausalImpact together. In addition, if you don't already have a set of test markets to match, 'MarketMatching' can provide suggested test/control market pairs using the 'suggest_market_splits' option in the 'best_matches()' function. Also, the 'test_fake_lift()' function provides pseudo prospective power analysis if you're using the 'MarketMatching' package to create your test design (i.e., not just doing the post inference).

Details

The MarketMatching package can be used to perform the following analyses:

- For all markets in the input dataset, find the best control markets using time series matching.

- Given a test market and a matching control market (from above), analyze the causal impact of an intervention.

- Create optimal test/control market splits and run pseudo prospective power analyses.

The package utilizes the dtw package in CRAN to do the time series matching, and the CausalImpact package to do the inference. (Created by Kay Brodersen at Google). For more information about the CausualImpact package, see the following reference:

CausalImpact version 1.0.3, Brodersen et al., Annals of Applied Statistics (2015). http://google.github.io/CausalImpact/

The MarketMatching has two separate functions to perform the tasks described above:

- best_matches(): This function finds the best matching control markets for all markets in the input dataset. If you don't know the test markets the function can also provide suggested optimized test/control pairs.

- inference(): Given an object from best_matches(), this function analyzes the causal impact of an intervention.

- test_fake_lift(): Calculate the probability of a causal impact for fake interventions (prospective pseudo power).

For more details, check out the vignette: browseVignettes("MarketMatching")

Author(s)

Kim Larsen (kblarsen4 at gmail.com)

Examples

```
## Not run:

##-------------------------------------------------------------------------
## Find best matches for CPH
## If we leave test_market as NULL, best matches are found for all markets
##-------------------------------------------------------------------------
library(MarketMatching)
data(weather, package="MarketMatching")
mm <- MarketMatching::best_matches(data=weather,
                   id="Area",
                   date_variable="Date",
                   matching_variable="Mean_TemperatureF",
                   parallel=FALSE,
                   markets_to_be_matched="CPH",
                   warping_limit=1, # warping limit=1
                   dtw_emphasis=1, # rely only on dtw for pre-screening
                   matches=5, # request 5 matches
                   start_match_period="2014-01-01",
                   end_match_period="2014-10-01")
head(mm$Distances)

##-------------------------------------------------------------------------
## Analyze causal impact of a made-up weather intervention in Copenhagen
## Since this is weather data it is a not a very meaningful example.
## This is merely to demonstrate the functionality.
##-------------------------------------------------------------------------
results <- MarketMatching::inference(matched_markets = mm,
                                     test_market = "CPH",
                                     analyze_betas=FALSE,
                                     end_post_period = "2015-10-01",
                                     prior_level_sd = 0.002)

## Plot the impact
results$PlotCumulativeEffect

## Plot actual observations for test market (CPH) versus the expectation (based on the control)
results$PlotActualVersusExpected

##-------------------------------------------------------------------------
## Power analysis for a fake intervention ending at 2015-10-01
## The maximum lift analyzed is 5 percent, the minimum is 0 (using 5 steps)
## Since this is weather data it is a not a very meaningful example.
## This is merely to demonstrate the functionality.
##-------------------------------------------------------------------------
power <- MarketMatching::test_fake_lift(matched_markets = mm,
                                     test_market = "CPH",
                                     end_fake_post_period = "2015-10-01",
```

```
                                      prior_level_sd = 0.002,
                                      steps=20,
                                      max_fake_lift=0.05)

## Plot the curve
power$ResultsGraph

##-----------------------------------------------------------------------
## Generate suggested test/control pairs
##-----------------------------------------------------------------------

data(weather, package="MarketMatching")
mm <- MarketMatching::best_matches(data=weather,
                                   id_variable="Area",
                                   date_variable="Date",
                                   matching_variable="Mean_TemperatureF",
                                   suggest_market_splits=TRUE,
                                   parallel=FALSE,
                              dtw_emphasis=0, # rely only on correlation for this analysis
                                   start_match_period="2014-01-01",
                                   end_match_period="2014-10-01")

##-----------------------------------------------------------------------
## The file that contains the suggested test/control splits
## The file is sorted from the strongest market pair to the weakest pair.
##-----------------------------------------------------------------------
head(mm$SuggestedTestControlSplits)

##-----------------------------------------------------------------------
## Pass the results to test_fake_lift to get pseudo power curves for the splits.
## This tells us how well the design can detect various lifts.
## Not a meaningful example for this data. Just to illustrate.
## Note that the rollup() function will aggregate the test and control markets.
## The new aggregated test markets will be labeled "TEST."
##-----------------------------------------------------------------------
rollup <- MarketMatching::roll_up_optimal_pairs(matched_markets = mm,
                                                synthetic=FALSE)

power <- MarketMatching::test_fake_lift(matched_markets = rollup,
                                        test_market = "TEST",
                                        end_fake_post_period = "2015-10-01",
                                        lift_pattern_type = "constant",
                                        max_fake_lift = 0.1)

## End(Not run)
```

`roll_up_optimal_pairs` *Roll up the suggested test/control optimal pairs for pseudo power analysis (testing fake lift)*

Description

`roll_up_optimal_pairs` Takes the suggested optimal pairs from best_matches() and aggregates the data for pseudo power analysis (test_fake_lift()). You run this function and then pass the result (a matched markets object) to test_fake_lift.

Usage

```
roll_up_optimal_pairs(matched_markets=NULL,
                      percent_cutoff=1,
                      synthetic=FALSE)
```

Arguments

`matched_markets`
: A matched market object from best_matches.

`percent_cutoff`
: The percent of data (by volume) to be included in the future study. Default is 1. 0.5 would be 50 percent.

`synthetic`
: If set to TRUE, the control markets are not aggregated so BSTS can determine weights for each market and create a synthetic control. If set to FALSE then the control markets are aggregated and each market will essentially get the same weight. If you have many control markets (say, more than 25) it is recommended to choose FALSE. Default is FALSE.

Value

Returns an object of type `market_matching`. The object has the following elements:

`BestMatches`
: A data.frame that contains the best matches for each market in the input dataset

`Data`
: The raw data used to do the matching

`MarketID`
: The name of the market identifier

`MatchingMetric`
: The name of the matching variable

`DateVariable`
: The name of the date variable

`SuggestedTestControlSplits`
: Always NULL

Examples

```
## Not run:
##-----------------------------------------------------------------------
## Generate the suggested test/control pairs
##-----------------------------------------------------------------------
library(MarketMatching)
data(weather, package="MarketMatching")
mm <- best_matches(data=weather,
                   id="Area",
                   date_variable="Date",
                   matching_variable="Mean_TemperatureF",
                   parallel=FALSE,
```

```
                   suggest_market_splits=TRUE,
                   start_match_period="2014-01-01",
                   end_match_period="2014-10-01")

##-------------------------------------------------------------------------
## Roll up the pairs to generate test and control markets
## Synthetic=FALSE means that the control markets will be aggregated
## -- i.e., equal weighhs in CausalImpact
##-------------------------------------------------------------------------

rollup <- roll_up_optimal_pairs(matched_markets=mm,
                                percent_cutoff=1,
                                synthetic=FALSE)

##-------------------------------------------------------------------------
## Pseudo power analysis (fake lift analysis)
##-------------------------------------------------------------------------

results <- test_fake_lift(matched_markets=rollup,
                     test_market="TEST",
                     lift_pattern_type="constant",
                     end_fake_post_period="2015-12-15",
                     prior_level_sd=0.002,
                     max_fake_lift=0.1)

## End(Not run)
```

`test_fake_lift` *Given a test market, analyze the impact of fake interventions (prospective power analysis)*

Description

`test_fake_lift` Given a matched_market object from the best_matches function, this function analyzes the causal impact of fake interventions using the CausalImpact package. The function returns an object of type "market_inference" which contains the estimated impact of the intervention (absolute and relative).

Usage

```
test_fake_lift(matched_markets=NULL,
          test_market=NULL,
          end_fake_post_period=NULL,
          alpha=0.05,
          prior_level_sd=0.01,
          control_matches=NULL,
          nseasons=NULL,
          max_fake_lift=NULL,
          steps=10,
          lift_pattern_type="constant")
```

Arguments

`matched_markets`
: A matched_market object created by the market_matching function This parameter will overwrite the values specified in prior_level_sd and nseasons. ONLY use this if you're using intricate bsts settings For most use-cases, using the prior_level_sd and nseasons parameters should be sufficient

`test_market`
: The name of the test market (character)

`end_fake_post_period`
: The end date of the post period. Must be a character of format "YYYY-MM-DD" – e.g., "2015-11-01"

`alpha`
: Desired tail-area probability for posterior intervals. For example, 0.05 yields 0.95 intervals

`prior_level_sd`
: Prior SD for the local level term (Gaussian random walk). Default is 0.01. The bigger this number is, the more wiggliness is allowed for the local level term. Note that more wiggly local level terms also translate into larger posterior intervals This parameter will be overwritten if you're using the bsts_modelargs parameter

`control_matches`
: Number of matching control markets to use in the analysis (default is 5)

`nseasons`
: Seasonality for the bsts model – e.g., 52 for weekly seasonality

`max_fake_lift`
: The maximum absolute fake lift – e.g., 0.1 means that the max lift evaluated is 10 percent and the min lift is -10 percent Note that randomization is injected into the lift, which means that the max lift will not be exactly as specified

`steps`
: The number of steps used to calculate the power curve (default is 10)

`lift_pattern_type`
: Lift pattern. Default is constant. The other choice is a random lift..

Value

Returns an object of type `matched_market_power`. The object has the following elements:

`ResultsData`
: The results stored in a data.frame

`ResultsGraph`
: The results stored in a ggplot graph

`LiftPattern`
: The random pattern applied to the lift

`FitCharts`
: The underlying actual versus fitted charts for each fake lift

`FitData`
: The underlying actual versus fitted data for each fake lift

Examples

```
## Not run:
library(MarketMatching)
##-----------------------------------------------------------------------
## Create a pseudo power curve for various levels of lift
## Since this is weather data it is a not a very meaningful example.
## This is merely to demonstrate the function.
##-----------------------------------------------------------------------
```

```
data(weather, package="MarketMatching")
mm <- best_matches(data=weather,
                   id="Area",
                   markets_to_be_matched=c("CPH", "SFO"),
                   date_variable="Date",
                   matching_variable="Mean_TemperatureF",
                   warping_limit=1, # warping limit=1
                   dtw_emphasis=1, # rely only on dtw for pre-screening
                   matches=5, # request 5 matches
                   start_match_period="2014-01-01",
                   end_match_period="2014-10-01")
library(CausalImpact)
results <- test_fake_lift(matched_markets=mm,
                     test_market="CPH",
                     lift_pattern_type="constant",
                     control_matches=5, # use all 5 matches for inference
                     end_fake_post_period="2015-12-15",
                     prior_level_sd=0.002,
                     max_fake_lift=0.1)

## End(Not run)
```

`weather` *Weather dataset*

Description

The data was extracted using the weatherData package It contains average temperature readings for 19 airports for 2014.

Usage

```
weather
```

Format

A time series dataset with 6,935 rows and 3 variables (19 airports and 365 days):

- Area: Airport code
- Date: Date
- Mean_TemperatureF: Average temperature

Index