Package ‘drifter’

May 31, 2019

Title  Concept Drift and Concept Shift Detection for Predictive Models

Version  0.2.1

Description  Concept drift refers to the change in the data distribution or
in the relationships between variables over time.
'drifter' calculates distances between variable distributions or
variable relations and identifies both types of drift.
Key functions are:
calculate_covariate_drift() checks distance between corresponding variables in two datasets,
calculate_residuals_drift() checks distance between residual distributions for two models,
calculate_model_drift() checks distance between partial dependency profiles for two models,
check_drift() executes all checks against drift.
'drifter' is a part of the 'DrWhy.AI' universe (Biecek 2018) <arXiv:1806.08915>.

Depends  R (>= 3.1)

License  GPL

Encoding  UTF-8

LazyData  true

Imports  DALEX, dplyr, tidyr, ingredients

Suggests  testthat, ranger

RoxygenNote  6.1.1

URL  https://ModelOriented.github.io/drifter/

BugReports  https://github.com/ModelOriented/drifter/issues

NeedsCompilation  no

Author  Przemyslaw Biecek [aut, cre]

Maintainer  Przemyslaw Biecek <przemyslaw.biecek@gmail.com>

Repository  CRAN

Date/Publication  2019-05-31 09:30:03 UTC
calculate_covariate_drift

Description

Here covariate drift is defined as Non-Intersection Distance between two distributions. More formally, \(d(P,Q) = 1 - \sum_i \min(P_i, Q_i)\). The larger the distance the more different are two distributions.

Usage

```r
calculate_covariate_drift(data_old, data_new, bins = 20)
```

Arguments

- `data_old`: data frame with ‘old’ data
- `data_new`: data frame with ‘new’ data
- `bins`: continuous variables are discretized to ‘bins’ intervals of equal sizes

Value

an object of a class ‘covariate_drift’ (data.frame) with Non-Intersection Distances

Examples

```r
library("DALEX")
# here we do not have any drift
d <- calculate_covariate_drift(apartments, apartments_test)
d
# here we do have drift
d <- calculate_covariate_drift(dragons, dragons_test)
d
```
**calculate_distance**  
*Calculate Non-Intersection Distance*

**Description**
Calculate Non-Intersection Distance

**Usage**
```r
calculate_distance(variable_old, variable_new, bins = 20)
```

**Arguments**
- `variable_old`: variable from `old` data  
- `variable_new`: variable from `new` data  
- `bins`: continuous variables are discretized to `bins` intervals of equal size

**Value**
Non-Intersection Distance

**Examples**
```r
calculate_distance(rnorm(1000), rnorm(1000))
calculate_distance(rnorm(1000), runif(1000))
```

---

**calculate_model_drift**  
*Calculate Model Drift for comparison of models trained on new/old data*

**Description**
This function calculates differences between PDP curves calculated for new/old models

**Usage**
```r
calculate_model_drift(model_old, model_new, data_new, y_new,
                      predict_function = predict, max_obs = 100, scale = sd(y_new, na.rm = TRUE))
```
calculate_model_drift

Arguments

- `model_old` model created on historical / ‘old’ data
- `model_new` model created on current / ‘new’ data
- `data_new` data frame with current / ‘new’ data
- `y_new` true values of target variable for current / ‘new’ data
- `predict_function` function that takes two arguments: model and new data and returns numeric vector with predictions, by default it’s ‘predict’
- `max_obs` if negative, then all observations are used for calculation of PDP, is positive, then only `max_obs` are used for calculation of PDP
- `scale` scale parameter for calculation of scaled drift

Value

An object of a class ‘model_drift’ (data.frame) with distances calculated based on Partial Dependency Plots

Examples

```r
library("DALEX")
model_old <- lm(m2.price ~ ., data = apartments)
model_new <- lm(m2.price ~ ., data = apartments_test[1:1000,])
calculate_model_drift(model_old, model_new,
                      apartments_test[1:1000,],
                      apartments_test[1:1000,]$m2.price)

library("ranger")
predict_function <- function(m, x,...) predict(m, x,...)$predictions
model_old <- ranger(m2.price ~ ., data = apartments)
model_new <- ranger(m2.price ~ ., data = apartments_test)
calculate_model_drift(model_old, model_new,
                      apartments_test, 
                      apartments_test$m2.price, 
                      predict_function = predict_function)

# here we compare model created on male data
# with model applied to female data
# there is interaction with age, and it is detected here
predict_function <- function(m,x,...) predict(m, x,..., probability=TRUE)$predictions[,1]
data_old = HR[HR$gender == "male", -1]
data_new = HR[HR$gender == "female", -1]
model_old <- ranger(status ~ ., data = data_old, probability=TRUE)
model_new <- ranger(status ~ ., data = data_new, probability=TRUE)
calculate_model_drift(model_old, model_new,
                      HR_test,
                      HR_test$status == "fired",
                      predict_function = predict_function)
```
# plot it
library("ingredients")
prof_old <- partial_dependency(model_old, 
    data = data_new[1:500,],
    label = "model_old",
    predict_function = predict_function,
    grid_points = 101,
    variable_splits = NULL)

prof_new <- partial_dependency(model_new, 
    data = data_new[1:500,],
    label = "model_new",
    predict_function = predict_function,
    grid_points = 101,
    variable_splits = NULL)

plot(prof_old, prof_new, color = "_label_")

calculate_residuals_drift

Calculate Residual Drift for old model and new vs. old data

Description

Calculate Residual Drift for old model and new vs. old data

Usage

calculate_residuals_drift(model_old, data_old, data_new, y_old, y_new, 
    predict_function = predict, bins = 20)

Arguments

- model_old: model created on historical / 'old' data
- data_old: data frame with historical / 'old' data
- data_new: data frame with current / 'new' data
- y_old: true values of target variable for historical / 'old' data
- y_new: true values of target variable for current / 'new' data
- predict_function: function that takes two arguments: model and new data and returns numeric vector with predictions, by default it’s 'predict'
- bins: continuous variables are discretized to 'bins' intervals of equal sizes

Value

an object of a class 'covariate_drift' (data.frame) with Non-Intersection Distances calculated for residuals
Examples

library("DALEX")
model_old <- lm(m2.price ~ ., data = apartments)
model_new <- lm(m2.price ~ ., data = apartments_test[1:1000,])
calculate_model_drift(model_old, model_new,
apartments_test[1:1000,],
apartments_test[1:1000,]$m2.price)

library("ranger")
predict_function <- function(m, x, ...) predict(m, x, ...)$predictions
model_old <- ranger(m2.price ~ ., data = apartments)
calculate_residuals_drift(model_old,
apartments_test[1:4000,], apartments_test[4001:8000,],
apartments_test$m2.price[1:4000], apartments_test$m2.price[4001:8000],
predict_function = predict_function)
calculate_residuals_drift(model_old,
apartments, apartments_test,
apartments$m2.price, apartments_test$m2.price,
predict_function = predict_function)

check_drift

This function executes all tests for drift between two datasets / models

Description

Currently three checks are implemented, covariate drift, residual drift and model drift.

Usage

check_drift(model_old, model_new, data_old, data_new, y_old, y_new,
predict_function = predict, max_obs = 100, bins = 20,
scale = sd(y_new, na.rm = TRUE))

Arguments

model_old model created on historical / 'old' data
model_new model created on current / 'new' data
data_old data frame with historical / 'old' data
data_new data frame with current / 'new' data
y_old true values of target variable for historical / 'old' data
y_new true values of target variable for current / 'new' data
predict_function function that takes two arguments: model and new data and returns numeric vector with predictions, by default it's 'predict'
**compare_two_profiles**

Calculates distance between two Ceteris Paribus Profiles

**Description**

This function calculates square root from mean square difference between Ceteris Paribus Profiles

**Usage**

```r
compare_two_profiles(cpprofile_old, cpprofile_new, variables, scale = 1)
```

**Arguments**

- `cpprofile_old`: Ceteris Paribus Profile for historical / 'old' model
- `cpprofile_new`: Ceteris Paribus Profile for current / 'new' model
- `variables`: variables for which drift should be calculated
- `scale`: scale parameter for calculation of scaled drift

**Value**

This function is executed for its side effects, all checks are being printed on the screen. Additionally it returns list with particular checks.

**Examples**

```r
library(“DALEX”)
model_old <- lm(m2.price ~ ., data = apartments)
model_new <- lm(m2.price ~ ., data = apartments_test[1:1000,])
check_drift(model_old, model_new,
            apartments, apartments_test,
            apartments$m2.price, apartments_test$m2.price)

library(“ranger”)
predict_function <- function(m, x, ...) predict(m, x, ...)$predictions
model_old <- ranger(m2.price ~ ., data = apartments)
model_new <- ranger(m2.price ~ ., data = apartments_test)
check_drift(model_old, model_new,
            apartments, apartments_test,
            apartments$m2.price, apartments_test$m2.price,
            predict_function = predict_function)
```
Value

data frame with distances between Ceteris Paribus Profiles

print.all_drifter_checks

Print All Drifter Checks

Description

Print All Drifter Checks

Usage

## S3 method for class 'all_drifter_checks'
print(x, ...)

Arguments

x  an object of the class 'all_drifter_checks'
... other arguments, currently ignored

Value

this function prints all drifter checks

Examples

library("DALEX")
model_old <- lm(m2.price ~ ., data = apartments)
model_new <- lm(m2.price ~ ., data = apartments_test[1:1000,])
check_drift(model_old, model_new,
            apartments, apartments_test,
apartments$m2.price, apartments_test$m2.price)

library("ranger")
predict_function <- function(m, x,...) predict(m, x,...)$predictions
model_old <- ranger(m2.price ~ ., data = apartments)
model_new <- ranger(m2.price ~ ., data = apartments_test)
check_drift(model_old, model_new,
            apartments, apartments_test,
apartments$m2.price, apartments_test$m2.price,
predict_function = predict_function)
print.covariate_drift  

Print Covariate Drift Data Frame

Description

Print Covariate Drift Data Frame

Usage

## S3 method for class 'covariate_drift'
print(x, max_length = 25, ...)

Arguments

x an object of the class 'covariate_drift'
max_length length of the first column, by default 25
... other arguments, currently ignored

Value

this function prints a data frame with a nicer format

Examples

library("DALEX")
# here we do not have any drift
d <- calculate_covariate_drift(apartments, apartments_test)
d
# here we do have drift
d <- calculate_covariate_drift(dragons, dragons_test)
d

print.model_drift  

Print Model Drift Data Frame

Description

Print Model Drift Data Frame

Usage

## S3 method for class 'model_drift'
print(x, max_length = 25, ...)


### Arguments

- **x**: an object of the class `model_drift`
- **max_length**: length of the first column, by default 25
- **...**: other arguments, currently ignored

### Value

This function prints a data frame with a nicer format.

### Examples

```r
library("DALEX")
model_old <- lm(m2.price ~ ., data = apartments)
model_new <- lm(m2.price ~ ., data = apartments_test[1:1000,])
calculate_model_drift(model_old, model_new,
    apartments_test[1:1000,],
    apartments_test[1:1000,]$m2.price)
```

```r
library("ranger")
predict_function <- function(m, x, ...) predict(m, x, ...)$predictions
model_old <- ranger(m2.price ~ ., data = apartments)
model_new <- ranger(m2.price ~ ., data = apartments_test)
calculate_model_drift(model_old, model_new,
    apartments_test,
    apartments_test$m2.price,
    predict_function = predict_function)
```

# here we compare model created on male data
# with model applied to female data
# there is interaction with age, and it is detected here
```r
data_old = HR[HR$gender == "male", -1]
data_new = HR[HR$gender == "female", -1]
model_old <- ranger(status ~ ., data = data_old, probability=TRUE)
model_new <- ranger(status ~ ., data = data_new, probability=TRUE)
calculate_model_drift(model_old, model_new,
    HR_test,
    HR_test$status == "fired",
    predict_function = predict_function)
```

# plot it
```r
library("ingredients")
prof_old <- partial_dependency(model_old,
    data = data_new[1:1000,],
    label = "model_old",
    predict_function = predict_function,
    grid_points = 101,
    variable_splits = NULL)
```

```r
prof_new <- partial_dependency(model_new,
    data = data_new[1:1000,],
    label = "model_new",
```
predict_function = predict_function,
grid_points = 101,
variable_splits = NULL)

plot(prof_old, prof_new, color = "_label_")
Index

calculate_covariate_drift, 2
calculate_distance, 3
calculate_model_drift, 3
calculate_residuals_drift, 5
check_drift, 6
compare_two_profiles, 7

print_all_drifter Checks, 8
print_covariate_drift, 9
print_model_drift, 9