Package 'ADLP'

July 21, 2025

Type Package

Title Accident and Development Period Adjusted Linear Pools for Actuarial Stochastic Reserving

Version 0.1.0

Author Benjamin Avanzi [aut],
William Ho [aut],
Yanfeng Li [aut, cre],
Bernard Wong [aut],
Alan Xian [aut]

Maintainer Yanfeng Li <yanfeng.li@student.unsw.edu.au>

Description Loss reserving generally focuses on identifying a single model that can generate superior predictive performance. However, different loss reserving models specialise in capturing different aspects of loss data.
This is recognised in practice in the sense that results from different models are often considered, and sometimes combined. For instance, actuaries may take a weighted average of the prediction outcomes from
various loss reserving models, often based on subjective assessments.

This package allows for the use of a systematic framework to objectively combine (i.e. ensemble) multiple stochastic loss reserving models such that the strengths offered by different models can be utilised effectively. Our
framework is developed in Avanzi et al. (2023). Firstly, our criteria model combination considers the full distributional properties of the ensemble and not just the central
estimate - which is of particular importance in the reserving context. Secondly, our framework is that it is tailored for the features inherent to reserving data. These include, for instance, accident, development,
calendar, and claim maturity effects. Crucially, the relative importance and scarcity of data across accident periods renders the problem distinct from the traditional ensemble techniques in statistical learning.

Our framework is illustrated with a complex synthetic dataset. In the results, the optimised ensemble outperforms both (i) traditional model selection strategies, and (ii) an equally weighted ensemble. In
particular, the improvement occurs not only with central estimates but also relevant quantiles, such as the 75th percentile of reserves (typically of interest to both insurers and regulators).
Reference: Avanzi B, Li Y, Wong B, Xian A (2023) ``Ensemble distributional forecasting for insurance loss reserving" <doi:10.48550/arXiv.2206.08541>.

License GPL-3

Encoding UTF-8

URL https://github.com/agi-lab/ADLP

BugReports https://github.com/agi-lab/ADLP/issues

LazyData true

RoxygenNote 7.3.1

Imports methods

Suggests knitr, tidyverse, SynthETIC (>= 1.0.0)

VignetteBuilder knitr

Depends R (>= 2.10)

NeedsCompilation no

Repository CRAN

Date/Publication 2024-04-18 19:23:01 UTC

Contents

`adlp_partition` *Accident and Development period Adjusted Linear Pools partition function*

Description

General framework for any user-defined function for partitions of claims triangle data.

Usage

```
adlp_partition(df, ...)

adlp_partition_none(df)

adlp_partition_ap(df, tri.size, size = 1, weights = rep(1, size))
```

Arguments

df	data.frame format of claims and related information for each cell. Dataframe will have columns origin and dev as columns 1 and 2 respectively.
...	Other parameters used to calculate ADLP partitions
tri.size	Triangle size in claims
size	Number of partitions
weights	a vector of weights for the size of each partition.

Details

adlp_partition_none is the default functionality with no partitions. This is equivalent to the standard linear pooling.

adlp_partition_ap will partition the claims triangle by accident period, Where the choice of accident period to partition will be determined to most closely resemble the desired weights.

The choice of accident period relies on a greedy algorithm that aims to find the accident period that provides the amount of cells that is larger or equal to the desired split.

Value

List containing the df as a result of the partitions.

See Also

adlp_partition_none, adlp_partition_ap

Examples

```
data("test_claims_dataset")
adlp_partition_none(test_claims_dataset)

data("test_claims_dataset")
adlp_partition_ap(test_claims_dataset, tri.size = 40, size = 3)
```

`calc_adlp_component` *Accident and Development period Adjusted Linear Pools Component Models*

Description

Accident and Development period Adjusted Linear Pools Component Models

Usage

```
calc_adlp_component(
  component,
  newdata,
  y = NULL,
  model = c("train", "full"),
  calc = c("pdf", "cdf", "mu", "sim")
)

calc_adlp_component_lst(
  components_lst,
  newdata,
  model = c("train", "full"),
  calc = c("pdf", "cdf", "mu", "sim"),
  ...
)
```

Arguments

`component`	Object of class `adlp_component`
`newdata`	Claims Triangle and other information. `data.frame` format of claims and related information for each cell. Dataframe will have columns `origin` and `dev` as columns 1 and 2 respectively.
`y`	Optional vector of `y` to be used in pdf or cdf calculations. Will default to the response fetched by `model.frame`.
`model`	Whether the training component model or the full component model should be used
`calc`	Type of calculation to perform
`components_lst`	List of objects of class `adlp_component`
`...`	Other parameters to be passed into `calc_adlp_component`

Details

Calls the specified function for an object of class `adlp_component`.

`calc_adlp_component_lst` is a wrapper for `calc_adlp_component` for each component in the list `components_lst`. This wrapper also contains functionality to signal the component that causes an error if it is occuring downstream.

Value

The result of the evaluated function on the `adlp_component`. This would be a vector with the same length as rows on `newdata` with the calculations.

Examples

```
data(test_adlp_component)

newdata <-  test_adlp_component$model_train$data
pdf_data = calc_adlp_component(test_adlp_component, newdata = newdata,
                          model = "train", calc = "pdf")

data(test_adlp_component)
test_component1 <- test_adlp_component
test_component2 <- test_adlp_component
test_components <- adlp_components(
    component1 = test_component1,
    component2 = test_component2
)

newdata <-  test_adlp_component$model_train$data
pdf_data = calc_adlp_component_lst(test_components, newdata = newdata,
                          model = "train", calc = "pdf")
```

`custom_model` *Custom Model Wrapper*

Description

Function to define basic functionality needed for a custom model that does not fit the general framework of models that align with adlp_component

Usage

```
custom_model(formula, data, ...)

## S3 method for class 'custom_model'
update(object, data, ...)
```

Arguments

`formula`	Formula needed that defines all variables required for the model
`data`	data to update custom model
`...`	Additional variables for update
`object`	Object of type `custom model`

Details

Custom model should support the S3 method `formula` and `update`.

Value

An object of class `custom_model`. `custom_model` is a list that stores the required formula to update the model and the data used to update the model.

Examples

```
data("test_claims_dataset")
custom_model <- custom_model(claims~., data=test_claims_dataset)
```

MM_optim *Minorization-Maximisation Algorithm performed to fit the ADLPs*

Description

The Minorization-Maximization algorithm aims to optimize a surrogate objective function that approximates the Log Score. This approach typically results in fast and stable convergence, while ensuring that combination weights adhere to the constraints of being non-negative and summing to one. For detailed description of the algorithm, one might refer to: Conflitti, De Mol, and Giannone (2015)

Usage

```
MM_optim(w_init, dat, niter = 500)
```

Arguments

`w_init`	initial weights for each ADLP
`dat`	matrix of densities for each ADLP
`niter`	maximum number of iterations. Defaults to 500

Value

An object of class `mm_optim`. `mm_optim` is a list that stores the results of the MM algorithm performed, including the final parameters, the final loss and numer of iterations.

References

Conflitti, Cristina, Christine De Mol, and Domenico Giannone. "Optimal combination of survey forecasts." International Journal of Forecasting 31.4 (2015): 1096-1103.

Examples

```
w_init <- rep(1/3, 3)
set.seed(1)
density_data <- matrix(runif(9), nrow = 3, ncol = 3)
MM_optim(w_init, density_data, niter = 500)
```

`predict.adlp` *Accident and Development period Adjusted Linear Pools (ADLP) Functions*

Description

Family of functions used to support ADLP inference and prediction.

Usage

```
## S3 method for class 'adlp'
predict(object, newdata = NULL, ...)

adlp_dens(adlp, newdata, model = c("train", "full"))

adlp_logS(adlp, newdata, model = c("train", "full"), epsilon = 1e-06)

adlp_CRPS(
  adlp,
  newdata,
  response_name,
  model = c("train", "full"),
  lower = 1,
  upper = NULL,
  sample_n = 2000
)

adlp_simulate(n, adlp, newdata = NULL)
```

Arguments

`object`	Object of class `adlp`
`newdata`	Data to perform the function on
`...`	Other parameters to pass onto predict
`adlp`	Object of class `adlp`
`model`	Whether the `train` or `full` model should be used in function
`epsilon`	Offset added to the density before calculating the log
`response_name`	The column name of the response variable; in string format

lower	The lower limit to calculate CRPS; the default value is set to be 1
upper	The upper limit to calculate CRPS; the default value is set to be twice the maximum value of the response variable in the dataset
sample_n	The number of evenly spaced values to sample between lower and upper range of numeric integration used to calculate CRPS. This sample function is designed to constrain memory usage during the computation of CRPS, particularly when dealing with large response variables.
n	number of simulations

Details

Predicts the central estimates based on the ADLP component models and weights.

Calculates the probability density ad each point, given `newdata`.

Calculates the log score, which is the log of the probability density, with an offset `epsilon` to handle zero densities. Log Score is a strictly proper scoring rule. For full discussion of the mathematical details and advantages of Log Score, one might refer to Gneiting and Raftery (2007)

Continuously Ranked Probability Score (CRPS) is calculated for each data point. `lower` and `upper` are used as limits when approximating the integral. CRPS is a strictly proper scoring rule. For full discussion of the mathematical details and advantages of CRPS, one might refer to Gneiting and Raftery (2007). The CRPS function has been discretized in this context to ensure adaptability to various distributions. For details, one might refer to Gneiting and Ranjan (2011)

Simulations of ADLP predictions, given component models and ADLP weights.

Value

`data.frame` of results, where the first and second columns correspond to the `$origin` and `$dev` columns from the triangles. An index column for `simulation #` is also included when simulating ADLP.

References

Gneiting, T., Raftery, A. E., 2007. Strictly proper scoring rules, prediction, and estimation. Journal of the American Statistical Association 102 (477), 359–378.

Gneiting, T., Ranjan, R., 2011. Comparing density forecasts using threshold-and quantile-weighted scoring rules. Journal of Business & Economic Statistics 29 (3), 411–422.

Examples

```
data(test_adlp_component)
test_component1 <- test_adlp_component
test_component2 <- test_adlp_component
test_components <- adlp_components(
    component1 = test_component1,
    component2 = test_component2
)

newdata <- test_component1$model_train$data
```

```
test_adlp <- adlp(test_components, newdata = newdata,
    partition_func = adlp_partition_ap, tri.size = 40, size = 3)

test_adlp_dens <- adlp_dens(test_adlp, newdata, "full")

data(test_adlp_component)
test_component1 <- test_adlp_component
test_component2 <- test_adlp_component
test_components <- adlp_components(
    component1 = test_component1,
    component2 = test_component2
)

newdata <- test_component1$model_train$data

test_adlp <- adlp(test_components, newdata = newdata,
    partition_func = adlp_partition_ap, tri.size = 40, size = 3)

test_adlp_logs <- adlp_logS(test_adlp, newdata, "full")

data(test_adlp_component)
test_component1 <- test_adlp_component
test_component2 <- test_adlp_component
test_components <- adlp_components(
    component1 = test_component1,
    component2 = test_component2
)

newdata <- test_component1$model_train$data

test_adlp <- adlp(test_components, newdata = newdata,
    partition_func = adlp_partition_ap, tri.size = 40, size = 3)

test_adlp_crps <- adlp_CRPS(test_adlp, newdata, "full", response_name = "claims", sample_n = 100)

data(test_adlp_component)
test_component1 <- test_adlp_component
test_component2 <- test_adlp_component
test_components <- adlp_components(
    component1 = test_component1,
    component2 = test_component2
)

newdata <- test_component1$model_train$data

test_adlp <- adlp(test_components, newdata = newdata,
    partition_func = adlp_partition_ap, tri.size = 40, size = 3)

test_adlp_sim <- adlp_simulate(10, test_adlp, newdata=newdata)
```

`print.adlp` *Accident and Development period Adjusted Linear Pools (ADLP) Models*

Description

Class to estimate an ADLP model fitted by Minorization-Maximisation.

Usage

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

adlp(components_lst, newdata, partition_func, param_tol = 1e-16, ...)
```

Arguments

`x`	Object of class `adlp`
`...`	Other named parameters passed onto further functions
`components_lst`	List of `adlp_components`
`newdata`	Validation data to fit the ADLP partitions on
`partition_func`	Partition function used to subset the data. ADLP weights will be generated for each partition. To specify partition preferences, set the parameter to `adlp_partition_none` if no partitioning is required. For partitioning the claims triangle by accident periods with predetermined weights, use `adlp_partition_ap`. Alternatively, users can create a custom partition function by defining the cut-off accident period for each subset manually.
`param_tol`	Tolerance for weights. Any value less than tolerance in magnitude is assumed zero.

Details

See adlp_component and adlp_components objects for more information on required format for inputs.

See adlp_partition for information on valid partition functions.

For an understanding of how partitions affect the performance of the ADLP ensemble, one might refer to Avanzi, Li, Wong and Xian (2022)

Value

Object of class `adlp`. This object has the following components:

components_lst adlp_components; List of adlp_components, see also `adlp_components`

model_weights vector; vector of model weights fitted for each component

partition_func function; Partition function used to fit the components

optim_MM mm_optim; Details related to the MM algorithm see also `MM_optim()`

newdata data.frame; Data.frame used to fit the ADLP

References

Avanzi, B., Li, Y., Wong, B., & Xian, A. (2022). Ensemble distributional forecasting for insurance loss reserving. arXiv preprint arXiv:2206.08541.

Examples

```
data(test_adlp_component)
test_component1 <- test_adlp_component
test_component2 <- test_adlp_component
test_components <- adlp_components(
    component1 = test_component1,
    component2 = test_component2
)

newdata <- test_component1$model_train$data

test_adlp <- adlp(test_components, newdata = newdata, response_name = "claims",
    partition_func = adlp_partition_ap, tri.size = 40, size = 3)
```

---

print.adlp_component *Accident and Development period Adjusted Linear Pools Component Models*

Description

Class to store component models and related functions required for ADLP estimation, prediction and goodness of fit.

Usage

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

adlp_component(
  model_train,
  model_full,
  calc_dens,
  calc_mu,
  calc_cdf,
  sim_fun,
  ...
)
```

Arguments

x	Object of class adlp_component
...	Other named parameters required for the model or any of its related functions to run.
model_train	Model trained on training data
model_full	Model trained on all in-sample data
calc_dens	function to calculate the pdf of each point
calc_mu	function to calculate the estimated mean of each point
calc_cdf	function to calculate the cdf of each point
sim_fun	function to simulate new from

Details

Component models model_train and model_full are designed to be objects of class glm, lm, or similar. The models would desirably have a S3 method for 'formula. For models that do not fit under this umbrella, see custom_model. For a potential list of candidate models, one might refer to Avanzi, Li, Wong and Xian (2022).

Functions as assumed to have the following parameter naming convention:

- y as the response variable
- model as the modeling object model_train or model_full
- newdata to designate new data

Other inputs not in this list will need to be intialised with the adlp_component

Value

Object of class adlp_component

References

Avanzi, B., Li, Y., Wong, B., & Xian, A. (2022). Ensemble distributional forecasting for insurance loss reserving. arXiv preprint arXiv:2206.08541.

Examples

```
data("test_claims_dataset")

train_val <- train_val_split_method1(
    df = test_claims_dataset,
    tri.size = 40,
    val_ratio = 0.3,
    test = TRUE
)
train_data <- train_val$train
valid_data <- train_val$valid
insample_data <- rbind(train_data, valid_data)
```

```
base_model1 <- glm(formula = claims~factor(dev),
                   family=gaussian(link = "identity"), data=train_data)

base_model1_full <- update(base_model1, data = insample_data)

dens_normal <- function(y, model, newdata){
    pred_model <- predict(model, newdata=newdata, type="response", se.fit=TRUE)
    mu <- pred_model$fit
    sigma <- pred_model$residual.scale
    return(dnorm(x=y, mean=mu, sd=sigma))
}

cdf_normal<-function(y, model, newdata){
    pred_model <- predict(model, newdata=newdata, type="response", se.fit=TRUE)
    mu <- pred_model$fit
    sigma <- pred_model$residual.scale
    return(pnorm(q=y, mean=mu, sd=sigma))
}

mu_normal<-function(model, newdata){
    mu <- predict(model, newdata=newdata, type="response")
    mu <- pmax(mu, 0)
    return(mu)
}

sim_normal<-function(model, newdata){
    pred_model <- predict(model, newdata=newdata, type="response", se.fit=TRUE)
    mu <- pred_model$fit
    sigma <- pred_model$residual.scale

    sim <- rnorm(length(mu), mean=mu, sd=sigma)
    sim <- pmax(sim, 0)
    return(sim)
}

base_component1 = adlp_component(
    model_train = base_model1,
    model_full = base_model1_full,
    calc_dens = dens_normal,
    calc_cdf = cdf_normal,
    calc_mu = mu_normal,
    sim_fun = sim_normal
)
```

---

print.adlp_components *Accident and Development period Adjusted Linear Pools Component Models*

Description

Accident and Development period Adjusted Linear Pools Component Models

Usage

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

adlp_components(...)
```

Arguments

x	Object of class adlp_components
...	Individual adlp_components

Details

Class to structure a list of adlp_components.

Value

An object of class adlp_components

Examples

```
data(test_adlp_component)
test_component1 <- test_adlp_component
test_component2 <- test_adlp_component
test_components <- adlp_components(
    component1 = test_component1,
    component2 = test_component2
)
```

---

test_adlp_component *Test ADLP Component*

Description

A adlp_component object created for examples.

Usage

```
test_adlp_component
```

Format

A adlp_component format, see adlp_component.

Examples

```
test_adlp_component
```

---

test_claims_dataset *Claims Data in data.frame Format*

Description

A data.frame of claims, with the corresponding Accident Period (`origin`) and Development Period (`dev`). A `calendar` column is also included (as the sum of `dev` and `origin`. This format is required for the ADLP package

Usage

```
test_claims_dataset
```

Format

A `data.frame` with 4 components:

origin Accident Period

dev Development Period

calendar Accident Period + Development Period

claims Claim amount

Examples

```
test_claims_dataset$claims
```

---

train_val_split *Train-Validation Split of Claims Triangle*

Description

General framework for any user-defined training/validation/testing split of claims triangle data. The ADLP package contains three default splitting algorithms.

Usage

```
train_val_split(df, ...)
```

Arguments

`df`	Claims Triangle and other information. `data.frame` format of claims and related information for each cell. Dataframe will have columns `origin` and `dev` as columns 1 and 2 respectively.
`...`	Other parameters used to calculate train/test splitting.

Value

List containing `$train`, `$valid`, `$test`, which should partition the input `df`.

See Also

train_val_split_by_AP, train_val_split_method1, train_val_split_method2

`train_val_split_by_AP` *Train-Validation Split by Accident Period*

Description

Function for training/validation splitting.

Usage

```
train_val_split_by_AP(df, accident_periods, max_dev_periods, test = FALSE)
```

Arguments

`df`	Claims Triangle and other information. `data.frame` format of claims and related information for each cell. Dataframe will have columns `origin` and `dev` as columns 1 and 2 respectively.
`accident_periods`	Vector of accident periods. Will be equivalent to `1:Triangle_Size`
`max_dev_periods`	Vector of development periods
`test`	Returns the test set if `TRUE`

Details

Assigns training set defined by a maximum development period for each accident period: $(x_{ij} <= MaxDP(i))$.

Validation set is therefore cells outside of this period but within the upper triangle. The test set is all observations in the lower triangle.

Value

List containing `$train`, `$valid`, `$test`, which should partition the input `df`.

See Also

train_val_split

Examples

```
data("test_claims_dataset")

train_val <- train_val_split_by_AP(
    df = test_claims_dataset,
    accident_periods = 1:40,
    max_dev_periods = 40:1,
    test = TRUE
)
```

`train_val_split_method1` *Train-Validation Split by Accident Period Method 1*

Description

Function for training/validation splitting.

Usage

```
train_val_split_method1(df, tri.size, val_ratio, test = FALSE)
```

Arguments

`df`	Claims Triangle and other information. `data.frame` format of claims and related information for each cell. Dataframe will have columns `origin` and `dev` as columns 1 and 2 respectively.
`tri.size`	Triangle size.
`val_ratio`	Value between 0 and 1 as the approximate size of validation set.
`test`	Returns the test set if `TRUE` .

Details

Approximates the validation set by taking the `n` most recent calendar years as validation to best fit `val_ratio`.

Validation set is therefore cells outside of this period but within the upper triangle. The test set is all observations in the lower triangle.

Note that accident period 1 and development period 1 will always be within the training set.

Value

List containing `$train`, `$valid`, `$test`, which should partition the input `df`.

See Also

train_val_split

Examples

```
data("test_claims_dataset")

train_val <- train_val_split_method1(
    df = test_claims_dataset,
    tri.size = 40,
    val_ratio = 0.3,
    test = TRUE
)
```

`train_val_split_method2`

Train-Validation Split by Accident Period Method 2

Description

Function for training/validation splitting.

Usage

```
train_val_split_method2(df, tri.size, val_ratio, test = FALSE)
```

Arguments

`df`	Claims Triangle and other information. `data.frame` format of claims and related information for each cell. Dataframe will have columns `origin` and `dev` as columns 1 and 2 respectively.
`tri.size`	Triangle size.
`val_ratio`	Value between 0 and 1 as the approximate size of validaiton set.
`test`	Returns the test set if `TRUE` .

Details

Approximates the validation set by defining the training set as the cells below the function $((b^{1/a} - x^{1/a})^a)$. Where b is equal to the triangle size and a is optimised to best fit `val_ratio`.

The training set is therefore cells outside of this period but within the upper triangle. The test set is all observations in the lower triangle.

Note that accident period 1 and development period 1 will always be within the training set.

Value

List containing `$train`, `$valid`, `$test`, which should partition the input `df`.

See Also

train_val_split

Examples

```
data("test_claims_dataset")

train_val <- train_val_split_method1(
    df = test_claims_dataset,
    tri.size = 40,
    val_ratio = 0.3,
    test = TRUE
)
```

Index