Package ‘discrim’

October 11, 2019

Title Model Wrappers for Discriminant Analysis
Version 0.0.1
License GPL-2
Depends parsnip, R (>= 2.10)
Imports purrr, rlang, tibble, withr, dials
Suggests testthat, MASS, mda, klaR, earth, mlbench, covr, ggplot2, xml2, spelling
Encoding UTF-8
LazyData true
RoxygenNote 6.1.1
Language en-US
NeedsCompilation no
Author Max Kuhn [aut, cre], RStudio [cph]
Maintainer Max Kuhn <max@rstudio.com>
Repository CRAN
Date/Publication 2019-10-11 12:10:02 UTC

R topics documented:

discrim_flexible .................................................. 2
discrim_linear .................................................... 4
discrim_regularized ............................................. 5
frac_common_cov .................................................. 7
discrim_flexible

General Interface for Flexible Discriminant Models

Description

discrim_flexible() is a way to generate a specification of a flexible discriminant model using features created using multivariate adaptive regression splines (MARS).

Usage

discrim_flexible(mode = "classification", num_terms = NULL,
prod_degree = NULL, prune_method = NULL)

## S3 method for class 'discrim_flexible'
update(object, num_terms = NULL,
prod_degree = NULL, prune_method = NULL, fresh = FALSE, ...)

Arguments

- **mode**: A single character string for the type of model. The only possible value for this model is "classification".
- **num_terms**: The number of features that will be retained in the final model, including the intercept.
- **prod_degree**: The highest possible interaction degree.
- **prune_method**: The pruning method.
- **object**: A flexible discriminant model specification.
- **fresh**: A logical for whether the arguments should be modified in-place or replaced wholesale.
- **...**: Not used for update().

Details

Flexible discriminant analysis (FDA) uses the work of Hastie et al (1994) to create a discriminant model using different feature expansions. For this function, MARS (Friedman, 1991) hinge functions are used to non-linearly model the class boundaries (see example below). The mda and earth packages are needed to fit this model.

The main arguments for the model are:

- **num_terms**: The number of features that will be retained in the final model.
- **prod_degree**: The highest possible degree of interaction between features. A value of 1 indicates and additive model while a value of 2 allows, but does not guarantee, two-way interactions between features.
prune_method: The type of pruning. Possible values are listed in ?earth.

These arguments are converted to their specific names at the time that the model is fit. Other options and argument can be set using set_engine(). If left to their defaults here (NULL), the values are taken from the underlying model functions. If parameters need to be modified, update() can be used in lieu of recreating the object from scratch.

The model can be created using the fit() function using the following engines:

- R: "earth" (the default)

Engine Details

Engines may have pre-set default arguments when executing the model fit call. For this type of model, the template of the fit calls are:

**earth** engine:

```r
mda::fda(formula = missing_arg(), data = missing_arg(),
    nprune = num_terms, degree = prod_degree, pmethod = prune_method,
    method = earth::earth)
```

References


Examples

```r
parabolic_grid <- expand.grid(X1 = seq(-5, 5, length = 100),
    X2 = seq(-5, 5, length = 100))

fda_mod <-
    discrim_flexible(num_terms = 3) %>%
    # increase 'num_terms' to find smoother boundaries
    set_engine("earth") %>%
    fit(class ~ ., data = parabolic)

parabolic_grid$fda <-
    predict(fda_mod, parabolic_grid, type = "prob")$.pred_Class1

library(ggplot2)

ggplot(parabolic, aes(x = X1, y = X2)) +
    geom_point(aes(col = class), alpha = .5) +
    geom_contour(data = parabolic_grid, aes(z = fda), col = "black", breaks = .5) +
    theme_bw() +
    theme(legend.position = "top") +
    coord_equal()

model <- discrim_flexible(num_terms = 10)
model
update(model, num_terms = 6)
```
General Interface for Linear Discriminant Models

discrim_linear

Description
discrim_linear() is a way to generate a specification of a linear discriminant analysis (LDA) model before fitting and allows the model to be created using different packages in R.

Usage
discrim_linear(mode = "classification", penalty = NULL)

## S3 method for class 'discrim_linear'
update(object, penalty = NULL, fresh = FALSE, ...

Arguments

mode A single character string for the type of model. The only possible value for this model is "classification".

penalty An non-negative number representing the amount of regularization used by some of the engines.

object A linear discriminant model specification.

fresh A logical for whether the arguments should be modified in-place of or replaced wholesale.

... Not used for update().

Details

For discrim_linear(), the mode will always be "classification".
The model can be created using the fit() function using the following engines:

- R: "MASS" (the default) or "mda"

The main argument for the model is:

- penalty: The total amount of regularization in the model. Note that this only used for the "mda" engine where it is a pure L2 penalty (a.k.a ridge regression).

This argument is converted to its specific names at the time that the model is fit. Other options and argument can be set using set_engine(). If left to their defaults here (NULL), the values are taken from the underlying model functions. If parameters need to be modified, update() can be used in lieu of recreating the object from scratch.
Engine Details

Engines may have pre-set default arguments when executing the model fit call. For this type of model, the template of the fit calls are:

**MASS** engine:

MASS::lda(x = missing_arg(), grouping = missing_arg())

**mda** engine:

mda::fda(formula = missing_arg(), data = missing_arg(), lambda = penalty, method = mda::gen.ridge, keep.fitted = FALSE)

Examples

parabolic_grid <-
expand.grid(X1 = seq(-5, 5, length = 100),
X2 = seq(-5, 5, length = 100))

lda_mod <-
discrim_linear(penalty = .1) %>%
set_engine("mda") %>%
fit(class ~ ., data = parabolic)

parabolic_grid$lda <-
predict(lda_mod, parabolic_grid, type = "prob")$.pred_Class1

library(ggplot2)
ggplot(parabolic, aes(x = X1, y = X2)) +
  geom_point(aes(col = class), alpha = .5) +
  geom_contour(data = parabolic_grid, aes(z = lda), col = "black", breaks = .5) +
  theme_bw() +
  theme(legend.position = "top") +
  coord_equal()
model <- discrim_linear(penalty = 0.1)
model
update(model, penalty = 1)
Usage

discrim_regularized(mode = "classification", frac_common_cov = NULL,
fract_identity = NULL)

## S3 method for class 'discrim_regularized'
update(object, frac_common_cov = NULL,
fract_identity = NULL, fresh = FALSE, ...)

Arguments

mode A single character string for the type of model. The only possible value for this
model is "classification".
frac_common_cov, frac_identity Numeric values between zero and one.
object A linear discriminant model specification.
fresh A logical for whether the arguments should be modified in-place of or replaced
wholesale.
... Not used for update().

Details

The model is from Friedman (1989) and can create LDA models, QDA models, and regularized
mixtures of the two. It does not conduct feature selection. The main arguments for the model are:

- fac_common_cov: The fraction of the regularized covariance matrix that is based on the
  LDA model (i.e., computed from all classes). A value of 1 is the linear discriminant analysis
  assumption while a value near zero assumes that there should be separate covariance matrices
  for each class.
- frac_identity: The fraction of the final, class-specific covariance matrix that is the identity
  matrix.

See klaR::rda() for the equations that define these parameters.

These arguments are converted to their specific names at the time that the model is fit. Other options
and argument can be set using set_engine(). If left to their defaults here (NULL), the values are
taken from the underlying model functions. If parameters need to be modified, update() can be
used in lieu of recreating the object from scratch.

For discrim_regularized(), the mode will always be "classification".

The model can be created using the fit() function using the following engines:

- R: "klaR"(the default)

Engine Details

Engines may have pre-set default arguments when executing the model fit call. For this type of
model, the template of the fit calls are:

klaR engine:

klaR::rda(x = missing_arg(), grouping = missing_arg(), lambda = frac_common_cov,
gamma = frac_identity, crossval = FALSE)
References


Examples

```r
parabolic_grid <- expand.grid(X1 = seq(-5, 5, length = 100),
                              X2 = seq(-5, 5, length = 100))

rda_mod <-
  discrim_regularized(frac_common_cov = .5, frac_identity = .5) %>%
  set_engine("klaR") %>%
  fit(class ~ ., data = parabolic)

parabolic_grid$rda <-
  predict(rda_mod, parabolic_grid, type = "prob")$pred_Class1

library(ggplot2)
ggplot(parabolic, aes(x = X1, y = X2)) +
  geom_point(aes(col = class), alpha = .5) +
  geom_contour(data = parabolic_grid, aes(z = rda), col = "black", breaks = .5) +
  theme_bw() +
  theme(legend.position = "top") +
  coord_equal()

model <- discrim_regularized(frac_common_cov = 10)
model
update(model, frac_common_cov = 1)
```

frac_common_cov | Parameter objects for Regularized Discriminant Models

Description

discrim_regularized() describes the effect of frac_common_cov() and frac_identity(). smoothness() is an alias for the adjust parameter in stats::density().

Usage

frac_common_cov(range = c(0, 1), trans = NULL)
frac_identity(range = c(0, 1), trans = NULL)
smoothness(range = c(0.5, 1.5), trans = NULL)
Arguments

- **range**: A two-element vector holding the *defaults* for the smallest and largest possible values, respectively.
- **trans**: A trans object from the scales package, such as `scales::log10_trans()` or `scales::reciprocal_trans()`. If not provided, the default is used which matches the units used in `range`. If no transformation, NULL.

Details

These parameters can modulate a RDA model to go between linear and quadratic class boundaries.

Value

A function with classes "quant_param" and "param"

Examples

`frac_common_cov()`

---

naive_Bayes

General Interface for Naive Bayes Models

Description

`naive_Bayes()` is a way to generate a *specification* of a model before fitting and allows the model to be created using different packages in R.

Usage

`naive_Bayes(mode = "classification", smoothness = NULL, Laplace = NULL)`

```r
# S3 method for class 'naive_Bayes'
update(object, smoothness = NULL, Laplace = NULL, fresh = FALSE, ...)
```

Arguments

- **mode**: A single character string for the type of model. The only possible value for this model is "classification".
- **smoothness**: An non-negative number representing the the relative smoothness of the class boundary. Smaller examples result in model flexible boundaries and larger values generate class boundaries that are less adaptable
- **Laplace**: A non-negative value for the Laplace correction to smoothing low-frequency counts.
- **object**: A linear discriminant model specification.
fresh  A logical for whether the arguments should be modified in-place of or replaced wholesale.

...  Not used for update().

Details

The main arguments for the model are:

- **smoothness**: The total amount of regularization in the model. Note that this only used for the "klaR" engine where it is a pure L2 smoothness (a.k.a ridge regression).
- **Laplace**: Laplace correction for smoothing low-frequency counts.

These arguments are converted to their specific names at the time that the model is fit. Other options and argument can be set using set_engine(). If left to their defaults here (NULL), the values are taken from the underlying model functions. If parameters need to be modified, update() can be used in lieu of recreating the object from scratch.

For naive_Bayes(), the mode will always be "classification".

The model can be created using the fit() function using the following engines:

- **R**: "klaR"(the default)

Engine Details

Engines may have pre-set default arguments when executing the model fit call. For this type of model, the template of the fit calls are:

**klaR**

klaR::NaiveBayes(x = missing_arg(), grouping = missing_arg(),
                  adjust = 0.8, usekernel = TRUE)

Note that usekernel is always set to TRUE here. This model does not need to make dummy variables from factor predictors. However, if parsnip::fit() is used to fit the model, dummy variables would be created while parsnip::fit_xy() will preserve the factor predictors in their original encoding.

Examples

parabolic_grid <-
  expand.grid(X1 = seq(-5, 5, length = 100),
              X2 = seq(-5, 5, length = 100))

nb_mod <-
  naive_Bayes(smoothness = .8) %>%
  set_engine("klaR") %>%
  fit(class ~ ., data = parabolic)

parabolic_grid$nb <-
predict(nb_mod, parabolic_grid, type = "prob")$pred_Class1
library(ggplot2)
ggplot(parabolic, aes(x = X1, y = X2)) +
  geom_point(aes(col = class), alpha = .5) +
  geom_contour(data = parabolic_grid, aes(z = nb), col = "black", breaks = .5) +
  theme_bw() +
  theme(legend.position = "top") +
  coord_equal()

model <- naive_Bayes(smoothness = 0.1)
model
update(model, smoothness = 1)
update(model, smoothness = 1, fresh = TRUE)

---

**parabolic**  
*Parabolic class boundary data*

**Description**
Parabolic class boundary data

**Details**
These data were simulated. There are two correlated predictors and two classes in the factor outcome.

**Value**

parabolic  
a data frame

**Examples**

data(parabolic)

library(ggplot2)
ggplot(parabolic, aes(x = X1, y = X2, col = class)) +
  geom_point(alpha = .5) +
  theme_bw()
Index

*Topic datasets
  parabolic, 10

  discrimFlexible, 2
  discrimLinear, 4
  discrimRegularized, 5

  fracCommonCov, 7
  fracIdentity(fracCommonCov), 7

naiveBayes, 8

parabolic, 10

  smoothness(fracCommonCov), 7

update.discrimFlexible
discrimFlexible, 2

update.discrimLinear(discrimLinear), 4

update.discrimRegularized
discrimRegularized, 5

update.naiveBayes(naiveBayes), 8