Package 'costsensitive'

May 8, 2026

Type Package

Title Cost-Sensitive Multi-Class Classification

Version 0.1.2.10

Date 2019-07-28

Author David Cortes

Maintainer David Cortes <david.cortes.rivera@gmail.com>

Description Reduction-based techniques for cost-sensitive multi-class classification, in which each observation has a different cost for classifying it into one class, and the goal is to predict the class with the minimum expected cost for each new observation.
Implements Weighted All-Pairs (Beygelzimer, A., Langford, J., & Zadrozny, B., 2008, <doi:10.1007/978-0-387-79361-0_1>), Weighted One-Vs-Rest (Beygelzimer, A., Dani, V., Hayes, T., Langford, J., & Zadrozny, B., 2005, <https://dl.acm.org/citation.cfm?id=1102358>) and Regression One-Vs-Rest.
Works with arbitrary classifiers taking observation weights, or with regressors. Also implements cost-proportionate rejection sampling for working with classifiers that don't accept observation weights.

Suggests parallel

URL https://github.com/david-cortes/costsensitive

License BSD_2_clause + file LICENSE

RoxygenNote 6.1.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2019-07-28 10:00:02 UTC

Contents

`cost.proportionate.classifier`
Cost-Proportionate Classifier

Description

Fits a classifier with sample weights by reducing the problem to classification without sample weights through rejection sampling.

Usage

```
cost.proportionate.classifier(X, y, weights, classifier, nsamples = 10,
  extra_rej_const = 0.1, nthreads = 1, seed = 1, ...)
```

Arguments

`X`	Features/Covariates for each observation.
`y`	Class for each observation.
`weights`	Weights for each observation.
`classifier`	Base classifier to use.
`nsamples`	Number of resamples to take.
`extra_rej_const`	Extra rejection constant - the higher, the smaller each sample ends up being, but the smallest the chance that the highest-weighted observations would end up in each sample.
`nthreads`	Number of parallel threads to use (not available on Windows systems). Note that, unlike the Python version, this is not a shared memory model and each additional thread will require more memory from the system. Not recommended to use when the algorithm is itself parallelized.
`seed`	Random seed to use for the random number generation.
`...`	Additional arguments to pass to 'classifier'.

References

Beygelzimer, A., Langford, J., & Zadrozny, B. (2008). Machine learning techniques-reductions between prediction quality metrics.

Examples

```
## Not run:
### example here requires 'caret' package
library(costsensitive)
data(iris)
set.seed(1)
X <- iris[, c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width")]
y <- factor(iris$Species == "setosa", labels = c("class1", "class2"))
weights <- rgamma(100, 1)
classifier <- caret::train
model <- cost.proportionate.classifier(X, y, weights, classifier,
  method = "glm", family = "binomial",
  trControl=caret::trainControl(method="none"), tuneLength=1)
predict(model, X, aggregation = "raw", type = "raw")
predict(model, X, aggregation = "weighted", type = "prob")

## End(Not run)
```

`predict.costprop` *Predict method for Cost-Proportionate Classifier*

Description

Predicts either the class or score according to predictions from classifiers fit to different resamples each. Be aware that the base classifier with which it was built must provide appropriate outputs that match with the arguments passed here (‘type‘ and ‘criterion‘). This is usually managed through argument ‘type‘ that goes to its ‘predict‘ method.

Usage

```
## S3 method for class 'costprop'
predict(object, newdata, aggregation = "raw",
  output_type = "score", ...)
```

Arguments

`object`	An object of class ‘costprop‘ as output by function ‘cost.proportionate.classifier‘.
`newdata`	New data on which to make predictions.
`aggregation`	One of "raw" (will take the class according to votes from each classifier. The predictions from classifiers must in turn be 1-dimensional vectors with the predicted class, not probabilities, scores, or two-dimensional arrays - in package ‘caret‘ for example, this corresponds to ‘type = "raw"‘), or "weighted" (will

take a weighted vote according to the probabilities or scores predicted by each classifier. The predictions from classifiers must in turn be either 1-dimensional vectors with the predicted probability/score, or two-dimensional matrices with the second column having the probability/score for the positive class = in package ‘caret‘ for example, this corresponds to ‘type = "prob‘).

`output_type` One of "class" (will output the predicted class) or "score" (will output the predicted score).

`...` Additional arguments to pass to the predict method of the base classifier.

Examples

```
## Not run:
### example here requires 'caret' package
library(costsensitive)
data(iris)
set.seed(1)
X <- X <- iris[, c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width")]
y <- factor(iris$Species == "setosa", labels = c("class1", "class2"))
weights <- rgamma(100, 1)
classifier <- caret::train
model <- cost.proportionate.classifier(X, y, weights, classifier,
  method = "glm", family = "binomial",
  trControl=caret::trainControl(method="none"), tuneLength=1)
predict(model, X, aggregation = "raw", type = "raw")
predict(model, X, aggregation = "weighted", type = "prob")

## End(Not run)
```

---

predict.rovr *Predict method for Regression One-Vs-Rest*

Description

Predicts either class with expected minimum cost or the expected cost (less is better) for new data.

Usage

```
## S3 method for class 'rovr'
predict(object, newdata, type = "class", ...)
```

Arguments

`object` An object of class ‘rovr‘ as output by function ‘regression.one.vs.rest‘.

`newdata` New data on which to make predictions.

`type` One of "class" (will output the class with minimum expected cost) or "score" (will output the predicted cost for each class, i.e. less is better).

`...` Additional arguments to pass to the predict method of the base regressor.

Value

When passing ‘type = "class"‘, a vector with class numbers or names (if the cost matrix had them). When passing ‘type = "score"‘, will output a ‘data.frame‘ with the same number of columns as ‘C‘ (passed to the ‘regression.one.vs.rest‘ function) and the predicted cost for each observation and class.

`predict.wap` *Predict method for Weighted All-Pairs*

Description

Predicts either the class with expected minimum cost or scores (more is better) for new data.

Usage

```
## S3 method for class 'wap'
predict(object, newdata, type = "class",
  criterion = "most-wins", ...)
```

Arguments

`object`	An object of class ‘wap‘ as output by function ‘weighted.all.pairs‘.
`newdata`	New data on which to make predictions.
`type`	One of "class" (will output the class with minimum expected cost) or "score" (will output the predicted score for each class, i.e. more is better).
`criterion`	One of "goodness" (will use the sum of probabilities output by each classifier) or "most-wins" (will use the predicted class by each classifier).
`...`	Additional arguments to pass to the predict method of the base classifier.

Value

When passing ‘type = "class"‘, a vector with class numbers or names (if the cost matrix had them). When passing ‘type = "score"‘, will output a ‘matrix‘ with the same number of columns as ‘C‘ (passed to the ‘weighted.all.pairs‘ function) and the predicted score for each observation and class.

predict.wovr *Predict method for Weighted One-Vs-Rest*

Description

Predicts either the class with expected minimum cost or scores (more is better) for new data.

Usage

```
## S3 method for class 'wovr'
predict(object, newdata, type = "class", ...)
```

Arguments

object	An object of class ‘wovr‘ as output by function ‘weighted.one.vs.rest‘.
newdata	New data on which to make predictions.
type	One of "class" (will output the class with minimum expected cost) or "score" (will output the predicted score for each class, i.e. more is better).
...	Additional arguments to pass to the predict method of the base classifier.

Value

When passing ‘type = "class"‘, a vector with class numbers or names (if the cost matrix had them). When passing ‘type = "score"‘, will output a ‘data.frame‘ with the same number of columns as ‘C‘ (passed to the ‘weighted.one.vs.rest‘ function) and the predicted score for each observation and class.

print.costprop *Get information about Cost-Proportionate classifier object*

Description

Prints basic information about a ‘costprop‘ object (Number of samples, rejection constant, random seed, class names, classifier class).

Usage

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

Arguments

x	An object of class "costprop".
...	Extra arguments (not used).

print.rovr *Get information about Regression One-Vs-Rest object*

Description

Prints basic information about a ‘rovr‘ object (Number of classes, regressor class).

Usage

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

Arguments

x	An object of class "rovr".
...	Extra arguments (not used).

print.wap *Get information about Weighted All-Pairs object*

Description

Prints basic information about a ‘wap‘ object (Number of classes and classifiers, classifier class).

Usage

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

Arguments

x	An object of class "wap".
...	Extra arguments (not used).

print.wovr *Get information about Weighted One-Vs-Rest object*

Description

Prints basic information about a ‘wovr‘ object (Number of classes, classifier class).

Usage

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

Arguments

x	An object of class "wovr".
...	Extra arguments (not used).

regression.one.vs.rest *Regression One-Vs-Rest*

Description

Creates a cost-sensitive classifier by creating one regressor per class to predict cost. Takes as input a regressor rather than a classifier. The objective is to create a model that would predict the class with the minimum cost.

Usage

```
regression.one.vs.rest(X, C, regressor, nthreads = 1, ...)
```

Arguments

X	The data (covariates/features).
C	matrix(n_samples, n_classes) Costs for each class for each observation.
regressor	function(X, y, ...) -> object, that would create regressor with method ‘predict‘.
nthreads	Number of parallel threads to use (not available on Windows systems). Note that, unlike the Python version, this is not a shared memory model and each additional thread will require more memory from the system. Not recommended to use when the algorithm is itself parallelized.
...	Extra arguments to pass to ‘regressor‘.

References

Beygelzimer, A., Langford, J., & Zadrozny, B. (2008). Machine learning techniques-reductions between prediction quality metrics.

Examples

```
library(costsensitive)
wrapped.lm <- function(X, y, ...) {
return(lm(y ~ ., data = X, ...))
}
set.seed(1)
X <- data.frame(feature1 = rnorm(100), feature2 = rnorm(100), feature3 = runif(100))
C <- data.frame(cost1 = rgamma(100, 1), cost2 = rgamma(100, 1), cost3 = rgamma(100, 1))
model <- regression.one.vs.rest(X, C, wrapped.lm)
predict(model, X, type = "class")
predict(model, X, type = "score")
print(model)
```

---

summary.costprop *Get information about Cost-Proportionate classifier object*

Description

Prints basic information about a 'costprop' object (Number of samples, rejection constant, random seed, class names, classifier class). Same as function 'print'.

Usage

```
## S3 method for class 'costprop'
summary(object, ...)
```

Arguments

object	An object of class "costprop".
...	Extra arguments (not used).

---

summary.rovr *Get information about Regression One-Vs-Rest object*

Description

Prints basic information about a 'rovr' object (Number of classes, regressor class). Same as function 'print'.

Usage

```
## S3 method for class 'rovr'
summary(object, ...)
```

Arguments

object	An object of class "rovr".
...	Extra arguments (not used).

---

summary.wap *Get information about Weighted All-Pairs object*

Description

Prints basic information about a ‘wap‘ object (Number of classes and classifiers, classifier class). Same as function ‘print‘.

Usage

```
## S3 method for class 'wap'
summary(object, ...)
```

Arguments

object	An object of class "wap".
...	Extra arguments (not used).

---

summary.wovr *Get information about Weighted One-Vs-Rest object*

Description

Prints basic information about a ‘wovr‘ object (Number of classes, classifier class). Same as function ‘print‘.

Usage

```
## S3 method for class 'wovr'
summary(object, ...)
```

Arguments

object	An object of class "wovr".
...	Extra arguments (not used).

`weighted.all.pairs` *Weighted All-Pairs*

Description

Creates a cost-sensitive classifier by creating one classifier per pair of classes to predict cost. Takes as input a classifier accepting observation weights. The objective is to create a model that would predict the class with the minimum cost.

Usage

```
weighted.all.pairs(X, C, classifier, predict_type_prob = "prob",
  wap_weights = TRUE, nthreads = 1, ...)
```

Arguments

`X`	The data (covariates/features).
`C`	matrix(n_samples, n_classes) Costs for each class for each observation.
`classifier`	function(X, y, weights=w, ...) -> object, that would create a classifier with method 'predict'. The 'y' vector passed to it is of class 'integer' with values 0/1 only.
`predict_type_prob`	argument to pass to method 'predict' from the classifier passed to 'classifier' in order to output probabilities (must be between zero and one) instead of classes (i.e. 'predict(object, newdata, type=predict_type_prob')).
`wap_weights`	Whether to use the weighting technique from the 'Weighted-All-Pairs' algorithm.
`nthreads`	Number of parallel threads to use (not available on Windows systems). Note that, unlike the Python version, this is not a shared memory model and each additional thread will require more memory from the system. Not recommended to use when the algorithm is itself parallelized.
`...`	Extra arguments to pass to 'classifier'.

References

Beygelzimer, A., Langford, J., & Zadrozny, B. (2008). Machine learning techniques-reductions between prediction quality metrics.

Examples

```
library(costsensitive)
wrapped.logistic <- function(X, y, weights, ...) {
return(glm(y ~ ., data = X, weights = weights, family = "quasibinomial", ...))
}
set.seed(1)
X <- data.frame(feature1 = rnorm(100), feature2 = rnorm(100), feature3 = runif(100))
```

```
C <- data.frame(cost1 = rgamma(100, 1), cost2 = rgamma(100, 1), cost3 = rgamma(100, 1))
model <- weighted.all.pairs(X, C, wrapped.logistic, predict_type_prob = "response")
predict(model, X, type = "class")
predict(model, X, type = "score")
print(model)
```

`weighted.one.vs.rest` *Weighted One-Vs-Rest*

Description

Creates a cost-sensitive classifier by creating one classifier per class to predict cost. Takes as input a classifier accepting observation weights. The objective is to create a model that would predict the class with the minimum cost.

Usage

```
weighted.one.vs.rest(X, C, classifier, predict_type_prob = "prob",
  wap_weights = FALSE, nthreads = 1, ...)
```

Arguments

`X`	The data (covariates/features).
`C`	matrix(n_samples, n_classes) Costs for each class for each observation.
`classifier`	function(X, y, weights=w, ...) -> object, that would create a classifier with method ‘predict‘. The ‘y‘ vector passed to it is of class ‘integer‘ with values 0/1 only.
`predict_type_prob`	argument to pass to method ‘predict‘ from the classifier passed to ‘classifier‘ in order to output probabilities or numeric scores instead of classes (i.e. ‘predict(object, newdata, type=predict_type_prob‘)).
`wap_weights`	Whether to use the weighting technique from the ‘Weighted-All-Pairs‘ algorithm.
`nthreads`	Number of parallel threads to use (not available on Windows systems). Note that, unlike the Python version, this is not a shared memory model and each additional thread will require more memory from the system. Not recommended to use when the algorithm is itself parallelized.
`...`	Extra arguments to pass to ‘classifier‘.

References

Beygelzimer, A., Dani, V., Hayes, T., Langford, J., & Zadrozny, B. (2005, August). Error limiting reductions between classification tasks.

Examples

```
library(costsensitive)
wrapped.logistic <- function(X, y, weights, ...) {
return(glm(y ~ ., data = X, weights = weights, family = "quasibinomial", ...))
}
set.seed(1)
X <- data.frame(feature1 = rnorm(100), feature2 = rnorm(100), feature3 = runif(100))
C <- data.frame(cost1 = rgamma(100, 1), cost2 = rgamma(100, 1), cost3 = rgamma(100, 1))
model <- weighted.one.vs.rest(X, C, wrapped.logistic, predict_type_prob = "response")
predict(model, X, type = "class")
predict(model, X, type = "score")
print(model)
```

Index