Package: ordinalForest (via r-universe)

September 4, 2024

Type Package

Title Ordinal Forests: Prediction and Variable Ranking with Ordinal Target Variables

Version 2.4-3

Date 2022-11-29

Author Roman Hornung

Maintainer Roman Hornung <hornung@ibe.med.uni-muenchen.de>

Imports Rcpp (>= 0.11.2), combinat, nnet, verification

LinkingTo Rcpp

Description The ordinal forest (OF) method allows ordinal regression with high-dimensional and low-dimensional data. After having constructed an OF prediction rule using a training dataset, it can be used to predict the values of the ordinal target variable for new observations. Moreover, by means of the (permutation-based) variable importance measure of OF, it is also possible to rank the covariates with respect to their importance in the prediction of the values of the ordinal target variable. OF is presented in Hornung (2020). NOTE: Starting with package version 2.4, it is also possible to obtain class probability predictions in addition to the class point predictions. Moreover, the variable importance values can also be based on the class probability predictions. Preliminary results indicate that this might lead to a better discrimination between influential and non-influential covariates. The main functions of the package are: ordfor() (construction of OF) and predict.ordfor() (prediction of the target variable values of new observations). References: Hornung R. (2020) Ordinal Forests. Journal of Classification 37, 4–17. <doi:10.1007/s00357-018-9302-x>.

License GPL-2

Encoding UTF-8

RoxygenNote 7.2.2

NeedsCompilation yes

Date/Publication 2022-11-30 13:50:02 UTC
Repository https://romanhornung.r-universe.dev
RemoteUrl https://github.com/cran/ordinalForest
RemoteRef HEAD
RemoteSha 4f827fbeeafc6a8e6bcaa9e441531f73e922f47c

Contents

ordinalForest-package 2
hearth 4
ordfor 5
perff 11
predict.ordfor 13

Index **15**

`ordinalForest-package` *Ordinal Forests: Prediction and Variable Ranking with Ordinal Target Variables*

Description

The ordinal forest (OF) method allows ordinal regression with high-dimensional and low-dimensional data. After having constructed an OF prediction rule using a training dataset, it can be used to predict the values of the ordinal target variable for new observations. Moreover, by means of the (permutation-based) variable importance measure of OF, it is also possible to rank the covariates with respect to their importances in the prediction of the values of the ordinal target variable.
OF is presented in Hornung (2020).

Details

Starting with package version 2.4, it is also possible to obtain class probability predictions in addition to the class point predictions and variable importance values based on the class probabilities through using the (negative) ranked probability score (Epstein, 1969) as performance function (`perffunction="probability"`, new default). Using the ranked probability score in the variable importance can be expected to deliver more stable variable rankings, because the ranked probability score accounts for the ordinal scale of the dependent variable. In situations in which there is no need for predicting class probabilities, but simply class predictions are sufficient, other performance functions may be more suitable. See the documentation of the `ordfor` function for further details.

For a brief, practice-orientated introduction to OF see: `ordfor`

The main functions are: `ordfor` (construction of OF prediction rules) and `predict.ordfor` (prediction of the values of the target variable values of new observations).

NOTE: **ordinalForest** uses R code and C++ code from the R package **ranger** for the involved regression forests. **ordinalForest** does, however, not depend on **ranger** or import **ranger**, because it was necessary to copy the C++ code and parts of the R code from **ranger** to **ordinalForest** instead. The reason for this is that **ranger**'s C++ code had to be altered in part in order to implement ordinal forest.

References

- Hornung R. (2020) Ordinal Forests. Journal of Classification 37, 4–17. <doi: 10.1007/s00357-0189302x>.
- Epstein E.S. (1969) A scoring system for probability forecasts of ranked categories, Journal of Applied Meteorology. 8(6), 985-987.

Examples

```
## Not run:
# Illustration of the key functionalities of the package:
############################################################

# Load example dataset:

data(hearth)

# Inspect the data:
table(hearth$Class)
dim(hearth)

head(hearth)


# Split into training dataset and test dataset:

set.seed(123)
trainind <- sort(sample(1:nrow(hearth), size=floor(nrow(hearth)*(2/3))))
testind <- setdiff(1:nrow(hearth), trainind)

datatrain <- hearth[trainind,]
datatest <- hearth[testind,]


# Construct OF prediction rule using the training dataset (default
# perffunction = "probability" corresponding to the
# (negative) ranked probability score as performance function):

ordforres <- ordfor(depvar="Class", data=datatrain, nsets=1000, ntreeperdiv=100,
  ntreefinal=5000, perffunction = "equal")
ordforres

# Study variable importance values:
sort(ordforres$varimp, decreasing=TRUE)

# Take a closer look at the top variables:
boxplot(datatrain$oldpeak ~ datatrain$Class, horizontal=TRUE)
fisher.test(table(datatrain$exang, datatrain$Class))

# Predict values of the ordinal target variable in the test dataset:

preds <- predict(ordforres, newdata=datatest)
preds
```

```
# Compare predicted values with true values:
table(data.frame(true_values=datatest$Class, predictions=preds$ypred))

## End(Not run)
```

`hearth` *Data on Coronary Artery Disease*

Description

This data includes 294 patients undergoing angiography at the Hungarian Institute of Cardiology in Budapest between 1983 and 1987.

Format

A data frame with 294 observations, ten covariates and one ordinal target variable

Details

The variables are as follows:

- `age`. numeric. Age in years
- `sex`. factor. Sex (1 = male; 0 = female)
- `chest_pain`. factor. Chest pain type (1 = typical angina; 2 = atypical angina; 3 = non-anginal pain; 4 = asymptomatic)
- `trestbps`. numeric. Resting blood pressure (in mm Hg on admission to the hospital)
- `chol`. numeric. Serum cholestoral in mg/dl
- `fbs`. factor. Fasting blood sugar > 120 mg/dl (1 = true; 0 = false)
- `restecg`. factor. Resting electrocardiographic results (1 = having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV); 0 = normal)
- `thalach`. numeric. Maximum heart rate achieved
- `exang`. factor. Exercise induced angina (1 = yes; 0 = no)
- `oldpeak`. numeric. ST depression induced by exercise relative to rest
- `Class`. factor. Ordinal target variable - severity of coronary artery disease (determined using angiograms) (1 = no disease; 2 = degree 1; 3 = degree 2; 4 = degree 3; 5 = degree 4)

The original openML dataset was pre-processed in the following way:

1. The variables were re-named according to the description given on openML.

2. The missing values which were coded as "-9" were replaced by NA values.

3. The variables `slope`, `ca`, and `thal` were excluded, because these featured too many missing values.

4. The categorical covariates were transformed into factors.

5. There were 6 `restecg` values of "2" which were replaced by "1".

6. The missing values were imputed: The missing values of the numerical covariates were replaced by the means of the corresponding non-missing values. The missing values of the categorical covariates were replaced by the modes of the corresponding non-missing values.

Source

OpenML: data.name: heart-h, data.id: 1565, link: https://www.openml.org/d/1565/

References

- Detrano, R., Janosi, A., Steinbrunn, W., Pfisterer, M., Schmid, J.-J., Sandhu, S., Guppy, K. H., Lee, S., Froelicher, V. (1989) International application of a new probability algorithm for the diagnosis of coronary artery disease. The American Journal Of Cardiology, 64, 304–310.
- Vanschoren, J., van Rijn, J. N., Bischl, B., Torgo, L. (2013) OpenML: networked science in machine learning. SIGKDD Explorations, 15(2), 49–60.

Examples

```
data(hearth)

table(hearth$Class)
dim(hearth)

head(hearth)
```

`ordfor` *Ordinal forests*

Description

Constructs prediction rules using the ordinal forest (OF) method presented in Hornung (2020).
The following tasks can be performed using OF: 1) Predicting the values of an ordinal target variable for new observations based on covariate values (see `predict.ordfor`); 2) Ranking the importances of the covariates with respect to predicting the values of the ordinal target variable.
The default values for the hyperparameters `nsets`, `ntreeperdiv`, `ntreefinal`, `npermtrial`, and `nbest` were found to be in a reasonable range in Hornung (2020) and it should not be necessary to alter these values in most situations.
For details on OFs see the 'Details' section below.
NOTE: Starting with package version 2.4, it is also possible to obtain class probability predictions in addition to the class point predictions and variable importance values based on the class probabilities through using the (negative) ranked probability score (Epstein, 1969) as performance function (`perffunction="probability"`). Using the ranked probability score in the variable importance can be expected to deliver more stable variable rankings, because the ranked probability score accounts for the ordinal scale of the dependent variable. In situations in which there is no need for

predicting class probabilities, but simply class predictions are sufficient, other performance functions may be more suitable. See the subsection "Performance functions" in the "Details" section below for further details.

Usage

```
ordfor(
  depvar,
  data,
  nsets = 1000,
  ntreeperdiv = 100,
  ntreefinal = 5000,
  importance = c("rps", "accuracy"),
 perffunction = c("equal", "probability", "proportional", "oneclass", "custom"),
  classimp,
  classweights,
  nbest = 10,
  naive = FALSE,
  num.threads = NULL,
  npermtrial = 500,
  permperdefault = FALSE,
  mtry = NULL,
  min.node.size = NULL,
  replace = TRUE,
  sample.fraction = ifelse(replace, 1, 0.632),
  always.split.variables = NULL,
  keep.inbag = FALSE
)
```

Arguments

`depvar`	character. Name of the dependent variable in `data`.
`data`	data.frame. Data frame containing the covariates and a factor-valued ordinal target variable. The order of the levels of the latter has to correspond to the order of the ordinal classes of the target variable.
`nsets`	integer. Number of score sets tried prior to the approximation of the optimal score set.
`ntreeperdiv`	integer. Number of trees in the smaller regression forests constructed for each of the `nsets` different score sets tried.
`ntreefinal`	integer. Number of trees in the larger regression forest constructed using the optimized score set (i.e., the OF).
`importance`	character. The type of variable importance measure to use. The default `"rps"` uses the ranked probability score as an error measure. If set to `"accuracy"`, the importance measure is based on the accuracy. The latter choice corresponds to the default importance measure of random forests and does not take the ordinal scale of the target variable into account. NOTE: If the ranked probability score is used as performance function (`perffunction="probability"`), `importance`

is set to "rps" automatically. Preliminary results indicate that the option "rps" might lead to a better discrimination between influential and non-influential covariates.

perffunction character. Performance function. The default is "equal". See 'Details', subsection 'Performance functions' below and perff.

classimp character. Class to prioritize if perffunction="oneclass".

classweights numeric. Needed if perffunction="custom": vector of length equal to the number of classes. Class weights - the higher the weight w_j assigned to class j is chosen, the higher the accuracy of the OF with respect to discerning observations in class j from observations not in class j will tend to be.

nbest integer. Number of best score sets used to calculate the optimized score set.

naive boolean. If set to TRUE, a naive ordinal forest is constructed, that is, the score set used for the classes of the target variable is not optimized, but instead the following (naive) scores are used: 1,2,3,... Note that it is strongly recommended to set naive=FALSE (default). The only advantage of choosing naive=TRUE is that the computational burden is reduced. However, the precision of the predictions of a prediction rule obtained using naive ordinal forest can be considerably worse than that of a corresponding prediction rule obtained using ordinal forest.

num.threads integer. Number of threads. Default is number of CPUs available (passed to the modified ranger code).

npermtrial integer. Number of permutations of the class width ordering to try for the second to the nsetsth score set tried prior to the calculation of the optimized score set.

permperdefault boolean. If set to TRUE, npermtrial different permutations will per default be tried for the 2th to the nsetsth score set used during the optimization - also for J! < nsets. Default is FALSE.

mtry integer. Number of variables to sample as candidate variables for each split. Default is the (rounded down) square root of the number of variables.

min.node.size integer. Minimal node size. Default is 5, except if perffunction="probability", in which case the default is 10.

replace boolean. Sample with replacement. Default is TRUE.

sample.fraction
numeric. Fraction of observations to sample. Default is 1 for sampling with replacement and 0.632 for sampling without replacement.

always.split.variables
character. Character vector with variable names to be always selected in addition to the mtry variables tried for splitting.

keep.inbag boolean. Save how often observations are in-bag in each tree. Default is FALSE.

Details

Introduction: The ordinal forest (OF) method allows ordinal regression with high-dimensional and low-dimensional data. After having constructed an OF prediction rule using a training dataset, it can be used to predict the values of the ordinal target variable for new observations. Moreover, by means of the (permutation-based) variable importance measure of OF, it is also possible to rank the covariates with respect to their importance in the prediction of the values of the ordinal

target variable.
OF is presented in Hornung (2020). See the latter publication for details on the method. In the following, a brief, practice-orientated introduction to OF is provided.

Methods: The concept of OF is based on the following assumption: There exists a (possibly latent) refined continuous variable y* underlying the observed ordinal target variable y (y in {1,...,J}, J number of classes), where y* determines the values of y. The functional relationship between y* and y takes the form of a monotonically increasing step function. Depending on which of J intervals]c_1, c_2],]c_2, c_3], ...,]c_J, c_{J+1}[contains the value of y*, the ordinal target variable y takes a different value.

In situations in which the values of the continuous target variable y* are known, they can be used in regression techniques for continuous response variables. The OF method is, however, concerned with settings in which only the values of the classes of the ordinal target variable are given. The main idea of OF is to optimize score values s_1,...,s_J to be used in place of the class values 1,...,J of the ordinal target variable in standard regression forests by maximizing the out-of-bag (OOB) prediction performance measured by a performance function g (see section "Performance functions").

The approximation of the optimal score set consists of two steps:
1) Construct a large number of regression forests (b in 1,...,`nsets`) featuring limited numbers of trees, where each of these uses as the values of the target variable a randomly generated score set s_{b,1},...,s_{b,J}. For each forest constructed, calculate the value of the performance function g using the OOB estimated predictions of the values of the ordinal target variable and the corresponding true values.
2) Calculate the approximated optimal score set s_1,...,s_J as a summary over the `nbest` best score sets generated in 1), that is, those `nbest` score sets that were associated with the highest values of the performance function g.

After calculating the optimized score set, a larger regression forest is constructed using this optimized score set s_1,...,s_J for the class values 1,...,J of the target variable. This regression forest is the OF prediction rule.

Except in the case of using the (negative) ranked probabilty score as performance function, prediction is performed by majority voting of the predictions of the individual trees in the OF. If the (negative) ranked probabilty score is used as performance function, both class predictions and predicted class probabilities are provided: The class probabilities are obtained by averaging over the class probabilities predicted by the individual trees and the class predictions are obtained as the classes with maximum class probabilites.

OF features a permutation variable importance measure that, if `importance` is set to `"rps"` (default), uses the ranked probability score as error measure and the misclassification error else (`importance="accuracy"`).

Hyperparameters: There are several hyperparameters, which do, however, not have to be optimized by the user in general, because the default values used for these hyperparameters were seen to be in a reasonable range and the results seem to be quite robust with respect to the choices of the hyperparameter values.

These hyperparameters are described in the following:

- `nsets` Default value: 1000. The default value of the number of considered score sets in the approximation of the optimal score set is quite large. A large number of considered score sets is necessary to attain a high chance that some of the score sets are close enough to the optimal

score set, that is, the score set that leads to the optimal OOB prediction performance with respect to the considered performance function (provided with the argument `perffunction`).

- `ntreeperdiv` Default value: 100. A very small number of trees considered per tried score set might lead to a too strong variability in the assessments of the performances achieved for the individual score sets. For ultra-high dimensional covariate data it might be necessary to choose a higher value for `ntreeperdiv` than the default value 100.
- `ntreefinal` Default value: 5000. The number of trees `ntreefinal` plays the same role as in conventional regression forests.
- `npermtrial` Default value: 500. As stated above it is necessary to consider a large number of tried score sets `nsets` in the optimization in order to increase the chance that the best of the considered score sets are close to the optimal score set. To further increase this chance, it is in addition necessary that the collection of score sets tried is heterogeneous enough across the iterations. OF uses a particular algorithm for sampling the score sets tried that leads to a strongly heterogeneous collection of sets. This algorithm features the hyperparameter `npermtrial`, where it has been seen in Hornung (2020) that the results are quite robust with respect to the choice of the value of this parameter.
- `nbest` Default value: 10. In the case of a relatively small value of `nsets`, it is important that the number `nbest` of best score sets used to calculate the optimized score set is not strongly misspecified. A too large value of `nbest` leads to including suboptimal score sets into the calculation of the optimized score set that are too distinct from the optimal score set. Conversely, a too small value of `nbest` leads to a high variance of the optimized score set. The combination `nsets=1000` and `nbest=10` should lead to a good trade-off between the heterogeneity of the considered score sets and the variance in the estimation. In Hornung (2020) this combination delivered good results and it was seen that using a very large value of `nbest` can lead to worse results.

Performance functions: As noted above, the different score sets tried during the estimation of the optimal score set are assessed with respect to their OOB prediction performance. The choice of the specific performance function used in these assessments determines the specific kind of performance the ordinal forest should feature:

- `perffunction="equal"` This choice should be made if it is of interest to classify observations from each class with the same accuracy independent of the class sizes. Youden's J statistic is calculated with respect to each class ("observation/prediction in class j" vs. "observation/prediction NOT in class j" (j=1,...,J)) and the simple average of the J results taken.
- `perffunction="probability"` This choice should be made if it is of interest to predict class probabilties for the observations. The ranked probability score is calculated between the predicted probabilities for the J classes and the observed class values. Because smaller values of the ranked probability score correspond to a better prediction, the negative ranked probability score is considered as performance functions.
- `perffunction="proportional"` This choice should be made if the main goal is to classify correctly as many observations as possible. The latter is associated with a preference for larger classes at the expense of a lower classification accuracy with respect to smaller classes. Youden's J statistic is calculated with respect to each class and subsequently a weighted average of these values is taken - with weights proportional to the number of observations representing the respective classes in the training data.
- `perffunction="oneclass"` This choice should be made if it is merely relevant that observations in class `categ` can be distinguished as reliably as possible from observations

not in class categ. Class categ must be passed to ordfor via the argument categ. Youden's J statistic is calculated with respect to class categ.

- perffunction="custom" This choice should be made if there is a particular ranking of the classes with respect to their importance. Youden's J statistic is calculated with respect to each class. Subsequently, a weighted average with user-specified weights (provided via the argument classweights) is taken. In this way, classes with higher weights are prioritized by the OF algorithm over classes with smaller weights.

Value

ordfor returns an object of class ordfor. An object of class "ordfor" is a list containing the following components:

forestfinal	object of class "ranger". Regression forest constructed using the optimized score set (i.e., the OF). Required by predict.ordfor.
bordersbest	vector of length J+1. Average over the nbest best partitions of [0,1]. Required by predict.ordfor.
forests	list of length nsets. The regression forests constructed for the nsets different score sets tried prior to the approximation of the optimal score set.
perffunctionvalues	vector of length nsets. Performance function values for all score sets tried prior to the approximation of the optimal score set.
bordersb	matrix of dimension nsets x (J+1). All nsets partitions of [0,1] considered.
classes	character vector of length J. Classes of the target variable.
nsets	integer. Number of score sets tried prior to the approximation of the optimal score set.
ntreeperdiv	integer. Number of trees per score set considered.
ntreefinal	integer. Number of trees of the OF prediction rule.
perffunction	character. Performance function used.
classimp	character. If perffunction="oneclass": class to priorize, NA else.
nbest	integer. Number of best score sets used to approximate the optimal score set.
classfreq	table. Class frequencies.
varimp	vector of length p. Permutation variable importance for each covariate. If perffunction="probability", the ranked probability score is used as error measure in the variable importance. For all other choices of the performance function, the misclassification error is used.

References

- Hornung R. (2020) Ordinal Forests. Journal of Classification 37, 4–17. <doi: 10.1007/s00357-0189302x>.
- Epstein E.S. (1969) A scoring system for probability forecasts of ranked categories, Journal of Applied Meteorology. 8(6), 985-987.

Examples

```
## Not run:
data(hearth)

set.seed(123)
hearthsubset <- hearth[sort(sample(1:nrow(hearth), size=floor(nrow(hearth)*(1/2)))),]
ordforres <- ordfor(depvar="Class", data=hearthsubset, nsets=50, nbest=5, ntreeperdiv=100,
  ntreefinal=1000)
# NOTE: nsets=50 is not enough, because the prediction performance of the resulting
# ordinal forest will be suboptimal!! In practice, nsets=1000 (default value) or a
# larger number should be used.

ordforres

sort(ordforres$varimp, decreasing=TRUE)

## End(Not run)
```

---

perff *Performance functions based on Youden's J statistic*

Description

In ordfor so-called performance functions are used to measure the performance of the smaller regression forests constructed prior to the approximation of the optimal score set. Except for one, which uses the ranked probability score (enabling class probability estimation), all of these performance functions are based on Youden's J statistic. These functions may, however, also be used to measure the precision of predictions on new data or the precision of OOB predictions. Note that the performance function using the ranked probability score is not covered in this help page. The function rps from the package verification (version 1.42) can be used to calculate the ranked probability score.

Usage

```
perff_equal(ytest, ytestpred, categ, classweights)

perff_proportional(ytest, ytestpred, categ, classweights)

perff_oneclass(ytest, ytestpred, categ, classweights)

perff_custom(ytest, ytestpred, categ, classweights)
```

Arguments

ytest	factor. True values of the target variable.
ytestpred	factor. Predicted values of the target variable.

categ	character. Needed in the case of perff_oneclass: Class to prioiritize.
classweights	numeric. Needed in the case of perff_custom: Vector of length equal to the number of classes. Class weights - classes with higher weights are to be prioiritzed over those with smaller weights.

Details

perff_equal should be used if it is of interest to classify observations from each class with the same accuracy independent of the class sizes. Youden's J statistic is calculated with respect to each class ("observation/prediction in class j" vs. "observation/prediction NOT in class j" (j=1,...,J)) and the simple average of the J results taken.

perff_proportional should be used if the main goal is to classify correctly as many observations as possible. The latter is associated with a preference for larger classes at the expense of a lower classification accuracy with respect to smaller classes. Youden's J statistic is calculated with respect to each class and subsequently a weighted average of these values is taken - with weights proportional to the number of observations representing the respective classes in the training data.

perff_oneclass should be used if it is merely relevant that observations in class categ can be distinguished as reliably as possible from observations not in class categ. Class categ must be passed to perff_oneclass via the argument categ. Youden's J statistic is calculated with respect to class categ.

perff_custom should be used if there is a particular ranking of the classes with respect to their importance. Youden's J statistic is calculated with respect to each class. Subsequently, a weighted average with user-specified weights (provided via the argument classweights) is taken. In this way, classes with higher weights are prioritized by the OF algorithm over classes with smaller weights.

References

- Hornung R. (2020) Ordinal Forests. Journal of Classification 37, 4–17. <doi: 10.1007/s00357-0189302x>.

Examples

```
## Not run:
data(hearth)

set.seed(123)
trainind <- sort(sample(1:nrow(hearth), size=floor(nrow(hearth)*(1/2))))
testind <- sort(sample(setdiff(1:nrow(hearth), trainind), size=20))

datatrain <- hearth[trainind,]
datatest <- hearth[testind,]

ordforres <- ordfor(depvar="Class", data=datatrain, nsets=50, nbest=5, ntreeperdiv=100,
  ntreefinal=1000)
# NOTE: nsets=50 is not enough, because the prediction performance of the resulting
# ordinal forest will be suboptimal!! In practice, nsets=1000 (default value) or a larger
# number should be used.
```

```
preds <- predict(ordforres, newdata=datatest)

table('true'=datatest$Class, 'predicted'=preds$ypred)


perff_equal(ytest=datatest$Class, ytestpred=preds$ypred)

perff_proportional(ytest=datatest$Class, ytestpred=preds$ypred)

perff_oneclass(ytest=datatest$Class, ytestpred=preds$ypred, categ="1")
perff_oneclass(ytest=datatest$Class, ytestpred=preds$ypred, categ="2")
perff_oneclass(ytest=datatest$Class, ytestpred=preds$ypred, categ="3")
perff_oneclass(ytest=datatest$Class, ytestpred=preds$ypred, categ="4")
perff_oneclass(ytest=datatest$Class, ytestpred=preds$ypred, categ="5")


perff_custom(ytest=datatest$Class, ytestpred=preds$ypred, classweights=c(1,2,1,1,1))


# perff_equal, perff_proportional, and perff_oneclass are special cases of perff_custom:

perff_custom(ytest=datatest$Class, ytestpred=preds$ypred, classweights=c(1,1,1,1,1))
perff_equal(ytest=datatest$Class, ytestpred=preds$ypred)

perff_custom(ytest=datatest$Class, ytestpred=preds$ypred, classweights=table(datatest$Class))
perff_proportional(ytest=datatest$Class, ytestpred=preds$ypred)

perff_custom(ytest=datatest$Class, ytestpred=preds$ypred, classweights=c(0,0,0,1,0))
perff_oneclass(ytest=datatest$Class, ytestpred=preds$ypred, categ="4")

## End(Not run)
```

`predict.ordfor` *Prediction using ordinal forest objects*

Description

Prediction of test data using ordinal forest.

Usage

```
## S3 method for class 'ordfor'
predict(object, newdata, ...)
```

Arguments

`object`	object of class `ordfor`. See function `ordfor`.
`newdata`	data.frame. Data frame containing new data.
`...`	further arguments passed to or from other methods.

Value

`predict.ordfor` returns an object of class `ordforpred`. An object of class "`ordforpred`" is a list containing the following components:

`ypred`	vector of length `nrow(newdata)`. Factor-valued test data predictions.
`classprobs`	predicted class probabilities. Only provided, if the performance function based on the ranked probability score was used, while training the ordinal forest (see `ordfor`). Matrix of dimension `nrow(newdata)` x J (NA, if the ranked probability was not used while training). The value in the j-th column of the i-th row contains the predicted probability that test observation i is of class j.

References

- Hornung R. (2020) Ordinal Forests. Journal of Classification 37, 4–17. <doi: 10.1007/s00357-0189302x>.

Examples

```
## Not run:
data(hearth)

set.seed(123)
trainind <- sort(sample(1:nrow(hearth), size=floor(nrow(hearth)*(1/2))))
testind <- sort(sample(setdiff(1:nrow(hearth), trainind), size=20))

datatrain <- hearth[trainind,]
datatest <- hearth[testind,]

ordforres <- ordfor(depvar="Class", data=datatrain, perffunction = "probability", nsets=50,
  nbest=5, ntreeperdiv=100, ntreefinal=1000)
# NOTE: nsets=50 is not enough, because the prediction performance of the resulting
# ordinal forest will be suboptimal!! In practice, nsets=1000 (default value) or a larger
# number should be used.

preds <- predict(ordforres, newdata=datatest)
preds

table(data.frame(true_values=datatest$Class, predictions=preds$ypred))

head(preds$classprobs)

## End(Not run)
```

Index

hearth, 4

ordfor, *2*, 5, *11*, *13, 14*
ordinalForest *(*ordinalForest-package*)*, 2
ordinalForest-package, 2

perff, *7*, 11
perff_custom *(*perff*)*, 11
perff_equal *(*perff*)*, 11
perff_oneclass *(*perff*)*, 11
perff_proportional *(*perff*)*, 11
predict.ordfor, *2*, *5*, *10*, 13