Package 'ExplainPrediction'

April 3, 2017

Title Explanation of Predictions for Classification and Regression Models

Version 1.1.8

Date 2017-04-03

Author Marko Robnik-Sikonja

Maintainer Marko Robnik-Sikonja <marko.robnik@fri.uni-lj.si>

Description Generates explanations for classification and regression models and visualizes them. Explanations are generated for individual predictions as well as for models as a whole. Two explanation methods
are included, EXPLAIN and IME. The EXPLAIN method is fast but might miss explanations expressed redundantly
in the model. The IME method is slower as it samples from all feature subsets.
For the EXPLAIN method see Robnik-
Sikonja and Kononenko (2008) <doi:10.1109/TKDE.2007.190734>,
and the IME method is described in Strumbelj and Kononenko (2010, JMLR, vol. 11:1-18).
All models in package 'CORElearn' are natively supported, for other prediction models a wrapper function is provided
and illustrated for models from packages 'randomForest', 'nnet', and 'e1071'.

License GPL-3

URL http://lkm.fri.uni-lj.si/rmarko/software/

Imports CORElearn (>= 1.50.2),semiArtificial (>= 2.2.5)

Suggests nnet,e1071,randomForest

NeedsCompilation no

Repository CRAN

Date/Publication 2017-04-03 15:30:41 UTC

R topics documented:

ExplainPrediction-package

Explanation of individual predictions and models

Description

The package ExplainPrediction contains methods to generate explanations for individual predictions of classification and regression models.

Details

The explanation methodology used is based on measuring contributions of individual features on an individual predictions. The contributions of all attributes present an explanation of individual prediction. Explanations can be visualized with a nomogram. If we average the explanations, we get an explanation of the whole model. Two explanation methods are implemented:

- EXPLAIN (described in *Explaining Classifications For Individual Instances*). The EXPLAIN method is much faster than IME and works for any number of attributes in the model, but cannot explain dependencies expressed disjunctively in the model. For details see `explainVis`.
- IME can in principle explain any type of dependencies in the model. It uses sampling based method to avoid exhaustive search for dependencies and works reasonably fast for up to a few dozen attributes in the model. The details see the references.

Currently prediction models implemented in package CORElearn are supported, for other models a wrapper of class `CoreModel` has to be created. The wrapper has to present the model with a list with the following components:

- `formula` of class `formula` representing the response and predictive variables,
- `noClasses` number of class values in class of classification model, 0 in case of regression,
- `class.lev` the levels used in representation of class values (see `factor`),

Additionally it has to implement function `predict` which returns the same components as the function `predict.CoreModel` in the package CORElearn.

Further software and development versions of the package are available at `http://lkm.fri.uni-lj.si/rmarko/software`.

Author(s)

Marko Robnik-Sikonja

References

Marko Robnik-Sikonja, Igor Kononenko: Explaining Classifications For Individual Instances. *IEEE Transactions on Knowledge and Data Engineering*, 20:589-600, 2008

Erik Strumbelj, Igor Kononenko, Igor, Marko Robnik-Sikonja: Explaining Instance Classifications with Interactions of Subsets of Feature Values. *Data and Knowledge Engineering*, 68(10):886-904, Oct. 2009

Erik Strumbelj, Igor Kononenko: An Eficient Explanation of Individual Classifications using Game Theory, *Journal of Machine Learning Research*, 11(1):1-18, 2010.

Some references are available from http://lkm.fri.uni-lj.si/rmarko/papers/

See Also

explainVis

`explanation` *Explanation of instance predictions and models*

Description

Using general explanation methodology EXPLAIN or IME, the function `explainVis` explains predictions of given model and visualizes the explanations. An explanation of prediction is given for an individual instance, but aggregation of these explanations is also possible, which gives explanation of the model. The details are given in the description and references.

Usage

```
explainVis(model, trainData, testData, visLevel=c("both","model","instance"),
        method=c("EXPLAIN", "IME"), problemName="", dirName=getwd(),
        fileType=c("none","pdf","eps","emf","jpg","png","bmp","tif","tiff"),
        naMode=c("avg", "na"), explainType=c("WE","infGain","predDiff"),
        classValue=1, nLaplace=nrow(trainData),estimator=NULL, pError=0.05,
       err=0.05, batchSize=40, maxIter=1000, genType=c("rf", "rbf", "indAttr"),
noAvgBins=20, displayAttributes=NULL, modelVisCompact=FALSE,
displayThreshold=0.0, normalizeTo=0,
        displayColor=c("color","grey"), noDecimalsInValueName=2,
      modelTitle="Model explanation for", instanceTitle="Explaining prediction for",
        showNumbers = FALSE, recall=NULL)
```

Arguments

`model`	The model as returned by CoreModel function.
`trainData`	Data frame with data, which is used to extract average explanations, discretization, and other information needed for explanation of instances and model. Typically this is the data set which was used to train the `model`.
`testData`	Data frame with instances which will be explained. The `testData` data frame shall contain the same columns as `trainData`, with possible exception of target variable, which can be omitted.
`visLevel`	The type of explanations desired. If `visLevel="model"` the explanation of model is generated, meaning that instance explanations obtained on `trainData` are aggregated. If `visLevel="instance"` an explanation for each instance (one row(in testData is generated. The default value `visLevel="both"` generates both the model explantion as well as explanations for all the instances.

`method` The explanation method; two methods are available, EXPLAIN and IME. The EXPLAIN is much faster and works for any number of attributes in the model, but cannot explain dependencies expressed disjunctively in the model (for details see references). The IME can in principle explain any type of dependencies in the model. It uses sampling based method to avoid exhaustive search for dependencies and works reasonably fast for up to a few dozen attributes in the model.

`problemName` A name of the problem to be written in graph titles. If `fileType` other than `"none"` is chosen the problem name is used as a name of a file name, where graphs are stored. See details section for how title is formed.

`dirName` A name of folder where resulting visualization files will be saved if `fileType` other than `"none"` is chosen.

`fileType` The parameter determines the graphical format of the visualization file. If `fileType="none"` (default) visualizations are generated in a graphical window. Other possible choices are `"pdf"`,`"eps"`,`"emf"`,`"jpg"`,`"png"`,`"bmp"`,`"tif"` and `"tiff"`.

`naMode` For method EXPLAIN this parameter determines how the impact of missing information about certain feature value is estimated. If `naMode="avg"`, the effect is estimated by the weighted average of predictions over all possible feature's values. If `naMode="na"`, the effect is estimated by inserting NA value as feature value. The `"na"` method is faster but we are left to the mercy of adequate treatment of missing values in the function `predict` for a given model.

`explainType` For method EXPLAIN this parameter determines how the prediction with knowledge about given feature and prediction without knowledge of this feature are combined into the final explanation. Values `"WE"`, `"infGain"`, and `"predDiff"` mean that the difference is interpreted as weight of evidence, information gain, or plain difference, respectively. For regression problem only the difference of predictions is available.

`classValue` For classification models this parameter determines for which class value the explanations will be generated. The `classValue` can be given as a factor, character string or class index. By default the first class value is chosen.

`nLaplace` For EXPLAIN method and classification problems the predicted probabilities are corrected with Laplace correction, pushing them away from 0 and 1 and towards uniform distribution. Larger values imply smaller effect. The default value is equal to the number of instances in `trainData`. The value 0 means that Laplace correction is not used and probabilities are estimated with relative frequency.

`estimator` The name of feature evaluation method used to greedily discretize attributes when averaging explanation over intervals. The default value `NULL` means that `"ReliefFexpRank"` will be used in classification problems and `"RReliefFexpRank"` will be used in regression problems. See `discretize` for details.

`pError` For method IME the estimated probability of an error in explanations. Together with parameter `err` this determines the number of needed samples.

`err` For method IME the parameter controls the size of tolerable error. Together with parameter `pError` this determines the number of needed samples. See the paper *An Efficient Explanation of Individual Classifications using Game Theory* for details.

batchSize — For method IME the number of samples processed in batch mode for each explanation. Larger sizes cause less overhead in processing but may process more samples than required.

maxIter — The maximal number of iterations in IME method allowed for a single explanation.

genType — The type of data generator used to generate random part of instances in method IME. The generators from package semiArtificial-package are used: "rf" stands for random forest based generator, "rbf" invokes RBF network based generator, and "indAttr" assumes independent attributes and generates values for each attribute independently.

noAvgBins — For IME method the number of discretization bins used to present model explanations and average explanations above individual explanations.

displayAttributes — The vector of attribute names which are shown in model explanation. The default value displayThreshold=NULL displays all attributes and their values.

modelVisCompact — The logical value controlling if attribute values are displayed in visualization of model explanation. The default value modelVisCompact=FALSE displays all values of attributes (subject to displayThreshold), and value modelVisCompact=TRUE displays only contributions for the whole attributes (without their values).

displayThreshold — The threshold value for absolute values of explanations below which feature contributions are not displayed in instance and model explanation graphs. The threshold applies after the values are normalized, see the explanation for parameter normalizeTo. The default value displayThreshold=0 displays contributions of all attributes.

normalizeTo — For visualization of instance explanations the absolute values of feature contributions are summed and normalized to the value of normalizeTo. In model explanation the normalization depends on parameter modelVisCompact. If its value is TRUE, the absolute values of all feature explanations are summed up and normalized to normalizeTo, otherwise the absolute values of all feature values' contributions are summed up. The value of normalizeTo common in some areas (e.g., in medicine) is 100. The default value 0 implies no normalization. The displayThreshold parameter refers to already normalized values.

displayColor — The parameter determines if the visualization will be color or grayscale.

noDecimalsInValueName — With how many decimal places will the numeric feature values be presented in visualizations. The default value is 2.

modelTitle — The value of the parameter becomes the first part of the title of model explanation graph. The information contained in problemName, class name and selected classValue are concatenated to the end of provided character string. If modelTitle=NULL the title is not shown.

instanceTitle — The value of parameter becomes the first part of the title in instance explanation graph. The information contained in problemName, class name and selected classValue are concatenated to the end of provided character string. If instanceTitle=NULL the title is not shown.

showNumbers — The logical value with default value FALSE. If showNumbers=TRUE the explanation values on shown on the graph inside each explanation box (or beside it if there is not enough space within the box).

recall — If parameter is different from NULL, it shall contain the list invisibly returned by one of previous calls to function explainVis. In this case the function reuses already computed explanations, average explanations, discretization, etc., and only display data differently according to other supplied parameters. In this case values of parameters model, testData and classValueshould be identical to the original call. Values of parameters trainData, method, naMode, explainType, nLaplace, estimator, pError, err, batchSize, maxIter, genType, and noAvgBins are ignored. The parameters that do matter in this case are the ones that affect the display of already precomputed results: visLevel, problemName, dirName, fileType, displayAttributes, modelVisCompact, displayThreshold, normalizeTo, displayColor, noDecimalsInValueName, modelTitle, and instanceTitle.

Details

The function explainVis generates explanations and their visualizations given the trained model, its training data, and data for which we want explanations. This is the frontend explanation function which takes care of everything, internally calling other functions. The produced visualizations are output to a graphical device or saved to a file. If one requires internal information about the explanations, they are returned invisibly. Separate calls to internal functions (explain, ime, prepareForExplanations, and explanationAverages) are also possible.

In the model explanation all feature values of nominal attributes and intervals of numeric attributes are visualized, as well as weighted summary over all these values. In the instance visualizations the contributions of each feature are presented (thick bars) as well as average contributions of these feature values in the trainData (thin bars above them). For details see the references below.

The graph title is composed of problemName, response variable, class value name in case of classification, type of model, and instance name, extracted from corresponding row.names in testData.

Value

The function explainVis generates explanations and saves their visualizations to a file or outputs them to graphical device, based on the value of fileType. It invisibly returns a list with three components containing explanations, average explanations and additional data like discretization used and data generator. The main ingredients of these three components are:

- expl, a matrix of generated explanations (of size dim(testData)),
- pCXA, a vector of predictions,
- pCXna, (for method EXPLAIN only) a matrix of predictions estimating missing knowledge of given attribute (of size dim(testData)).
- stddev, (for method IME only) a matrix with standard deviations of explanations,
- noIter, (for method IME only) a matrix with number of iterations executed for each explanation,
- discPoints, (for method EXPLAIN only) a list containing values of discrete features or centers of discretization intervals for numeric features,

- pAV, (for method EXPLAIN only) a list with probabilities for discrete values or discretization intervals in case of numeric features,
- discretization, a list with discretization intervals output by discretize function, used in estimating averages and model based explanations,
- avNames, a list containing the names of discrete values/intervals,
- generator, (for IME method only) a generator used to generate random part of instances in IME method,
- explAvg, a list with several components giving average explanations on the trainingData. Averages are given for attributes, their values (for discrete attributes) and discretization intervals (for numeric features). These average explanations are used in visualization to give impression how the model works on average. This can be contrasted with explanation for the specific instance.

Author(s)

Marko Robnik-Sikonja

References

Marko Robnik-Sikonja, Igor Kononenko: Explaining Classifications For Individual Instances. *IEEE Transactions on Knowledge and Data Engineering*, 20:589-600, 2008

Erik Strumbelj, Igor Kononenko, Igor, Marko Robnik-Sikonja: Explaining Instance Classifications with Interactions of Subsets of Feature Values. *Data and Knowledge Engineering*, 68(10):886-904, Oct. 2009

Erik Strumbelj, Igor Kononenko: An Efficient Explanation of Individual Classifications using Game Theory, *Journal of Machine Learning Research*, 11(1):1-18, 2010.

Marko Robnik-Sikonja, Igor Kononenko: Discretization of continuous attributes using ReliefF. *Proceedings of ERK'95*, B149-152, Ljubljana, 1995

Some references are available from http://lkm.fri.uni-lj.si/rmarko/papers/

See Also

CORElearn, predict.CoreModel, attrEval, discretize, semiArtificial-package

Examples

```
require(CORElearn) # contains several prediction models
# use iris data set, split it randomly into a training and testing set
trainIdxs <- sample(x=nrow(iris), size=0.7*nrow(iris), replace=FALSE)
testIdxs <- c(1:nrow(iris))[-trainIdxs]
# build random forests model with certain parameters
modelRF <- CoreModel(Species ~ ., iris[trainIdxs,], model="rf",
             selectionEstimator="MDL",minNodeWeightRF=5,
             rfNoTrees=100, maxThreads=1)

# generate model explanation and visualization
# turn on history in the visualization window to see all graphs
explainVis(modelRF, iris[trainIdxs,], iris[testIdxs[1:5],], method="EXPLAIN",visLevel="both",
```

```
            problemName="iris", fileType="none",
            naMode="avg", explainType="WE", classValue=1, displayColor="color")

## Not run:
#store instance explanations to file
explainVis(modelRF, iris[trainIdxs,], iris[testIdxs[5:10], ], method="EXPLAIN", visLevel="instance",
            problemName="iris", fileType="pdf",
            naMode="avg", explainType="WE", classValue=1, displayColor="color")
destroyModels(modelRF) # clean up

# build a regression tree
trainReg <- regDataGen(100)
testReg <- regDataGen(20)
modelRT <- CoreModel(response~., trainReg, model="regTree", modelTypeReg=1)
# generate both model and instance explanations using the defaults
explainVis(modelRT, trainReg, testReg[1:3,]) # the first three testing instances
destroyModels(modelRT) #clean up

## End(Not run)
```

`wrap4Explanation` *Wrap prediction model for explanations*

Description

The function wraps given prediction model to be used with ExplainPrediction package. Currently `nnet` from `nnet` package and models of class `svm` from package `e1071` are supported, but others can easily be added. Please, note that models from `CORElearn-package` can be used directly and need no wrapper. If inclusion of other models into ExplainPrediction is desired, please, contact the author.

Usage

```
wrap4Explanation(model)
```

Arguments

`model`	The model as returned by `nnet` or any of `svm` functions in `e1071`package.

Details

The function adds necessary components to the prediction model so that function `explainVis` can generate explanations and their visualizations. Currently, four components are added:

- `formula`, a formula specifying the dependent and independent variables used by the supplied model.
- `model`, a name of the supplied model.
- `noClasses`, a number of class values for classification problems and 0 for regression.

- `class.lev`, for classification problem a vector of class value names.

If for a given `model` the method `predict` returns the class value probabilities as matrix or in a list with component `probabilities`, nothing else is needed, otherwise the internal function `getPredictions` has to be adequately modified.

Value

The function returns unchanged `model` with the components described in Details.

Author(s)

Marko Robnik-Sikonja

See Also

`explainVis`

Examples

```
## Not run:
# use iris data set, split it randomly into a training and testing set
trainIdxs <- sample(x=nrow(iris), size=0.7*nrow(iris), replace=FALSE)
testIdxs <- c(1:nrow(iris))[-trainIdxs]
# build a nnet model with certain parameters
require(nnet)
modelNN <- nnet(Species ~ ., iris[trainIdxs,], size=20)

# use wrapper
modelNNet <- wrap4Explanation(modelNN)

# generate model explanation and visualization
# turn on history in the visualization window to see all graphs
explainVis(modelNNet, iris[trainIdxs,], iris[testIdxs,], method="EXPLAIN",visLevel="both",
           problemName="iris", fileType="none",
           naMode="avg", explainType="WE", classValue=1, displayColor="color")

## End(Not run)
```

Index