Vignette for simpleEnsembleGroup22 Package

Sravya Yarlagadda, Hasan Md Moonam, Aman Arya, Eric Seeram

2024-05-03

Contents

0.1 Introduction

This vignette describes the usage and internal workings of the `simpleEnsembleGroup22` package, which includes various techniques for regression and ensemble learning.

0.2 Importing the Package

To use the `simpleEnsembleGroup22` package, you will have to install the package like this specifying the location of the package.

```
install.packages("C:/Users/sryarlagadda/Downloads/simpleEnsembleGroup22_0.0.0.9000.tar.gz")
```

```
## Installing package into 'C:/Users/sryarlagadda/AppData/Local/R/win-library/4.3'
## (as 'lib' is unspecified)
```

```
## inferring 'repos = NULL' from 'pkgs'
```

```
library("simpleEnsembleGroup22")
```

0.3 Internal Workings of Various Functions and Examples

This package is better compatible when the function simple_ensemble() is used to specify the requirements and input variables (The dataset, whether ensembling of model types is required, option for bagging on a single model type, option to select top K predictors of the dataset to perform model fitting). The datasets used should have the response variables as binary or continuous and the explanatory variables can be a combination of binary, discrete and continuous variables. Please ensure that the dataset doesn't contain any missing values.

Disclaimer: The package also provides the following functions individually. The datasets and other parameters have to be given as inputs to the functions in the format mentioned in the examples. Checks like this have to be ensured - For a simple linear model, number of predictors has to be less than the sample size. For other models, the sample size has to be atleast 200. For These are not compatible for datasets with missing values. The predicted values here are the fitted values of the models.

1. Fitting various models for a dataset (linear_model(), ridge_model(), lasso_model(), elastic_net_regression(), rf_model()).
2. Ensemble Learning of various models - "elastic_net" and "random_forest". (Since elastic_net model is an ensemble of ridge and lasso models, and since ridge and lasso are penalized linear regression methods) More on this can be found from the ensemble_predict() documentation.
3. Bagging of a single model type - The dataset is sampled using replacement and the predictions of these multiple models are averaged.
4. Screen for top K predictors of a dataset. Models are usually suitable to be fitted when n»p (n=sample size, p=number of predcitor variables). This function screens for top K predictors(if specified by the user or if n<20*p, it screens for top K predictors based on a certain threshold value). More on this can be known from the variable_pre_screening() documentation.

0.3.1 1. Linear Regression Model (`linear_model`)

The linear_model function fits a linear model using either linear regression for continuous response variables or logistic regression for binary response variables.

Description:

This function accepts two arguments:

X: Matrix of predictors . Each row represents an observation, and each column represents a predictor. It can be a mix of continuous, discrete, and binary predictors.

y: Response variable. For binary outcomes, it should be a binary vector (0 or 1). For continuous outcomes, it should be a numeric vector.

It returns a list containing the fitted model and predicted values.

Details:

For continuous response variables, linear_model performs linear regression.

For binary response variables, it performs logistic regression.

Examples:

```r
library(MASS)
## Warning: package 'MASS' was built under R version 4.3.3
data(Boston)
str(Boston)
## 'data.frame':    506 obs. of  14 variables:
##  $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
##  $ chas   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ rm     : num  6.58 6.42 7.18 7 7.15 ...
##  $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ rad    : int  1 2 2 3 3 3 5 5 5 5 ...
##  $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
##  $ black  : num  397 397 393 395 397 ...
##  $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
##  $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
X <- Boston[, 2:14]
y <- Boston[,1]
linear_results <- linear_model(X, y) #For Continuous Response variable
## [1] "Linear regression is being done - "
linear_results
## $model
##
## Call:
## lm(formula = y ~ ., data = data.frame(y, X))
##
## Coefficients:
## (Intercept)           zn        indus         chas          nox
##   17.033228     0.044855    -0.063855    -0.749134   -10.313535
##  [ reached getOption("max.print") -- omitted 9 entries ]
##
##
## $predicted.value
##          1          2          3          4          5
## -0.7849541 -0.9798716 -3.8967710 -4.1275214 -4.3235862
##  [ reached getOption("max.print") -- omitted 501 entries ]

data(Pima.te)
str(Pima.te)
## 'data.frame':    332 obs. of  8 variables:
##  $ npreg: int  6 1 1 3 2 5 0 1 3 9 ...
##  $ glu  : int  148 85 89 78 197 166 118 103 126 119 ...
##  $ bp   : int  72 66 66 50 70 72 84 30 88 80 ...
##  $ skin : int  35 29 23 32 45 19 47 38 41 35 ...
##  $ bmi  : num  33.6 26.6 28.1 31 30.5 25.8 45.8 43.3 39.3 29 ...
##  $ ped  : num  0.627 0.351 0.167 0.248 0.158 0.587 0.551 0.183 0.704 0.263 ...
##  $ age  : int  50 31 21 26 53 51 31 33 27 29 ...
##  $ type : Factor w/ 2 levels "No","Yes": 2 1 1 2 2 2 2 1 1 2 ...
X <- Pima.te[, 1:7]
y <- Pima.te[,8]
```

```r
logistic_results <- linear_model(X, as.numeric(y) - 1) #For Binary Response variable
## [1] "Linear regression is being done - "
logistic_results
## $model
##
## Call:  glm(formula = y ~ ., family = "binomial", data = data.frame(y,
##     X))
##
## Coefficients:
## (Intercept)        npreg          glu           bp         skin
##   -9.514019     0.140944     0.037481    -0.008675     0.013167
##  [ reached getOption("max.print") -- omitted 3 entries ]
##
## Degrees of Freedom: 331 Total (i.e. Null);  324 Residual
## Null Deviance:       420.3
## Residual Deviance: 285.8     AIC: 301.8
##
## $predicted.value
##          1          2          3          4          5
## 0.72444856 0.03460851 0.02863966 0.04797794 0.84253480
##  [ reached getOption("max.print") -- omitted 327 entries ]
```

0.3.2 2. Ridge Regression (`ridge_model`)

The ridge_model function performs ridge regression, which is a regularization technique used to mitigate multicollinearity in the data and prevent overfitting.

Ridge regression adds an L2 penalty term to the standard linear regression objective function, minimizing the sum of squared residuals plus a penalty term that shrinks the coefficients towards zero. The function performs cross-validation to select the optimal lambda value, which controls the amount of regularization applied.We have used 10-fold cross-validation as the standard here. It fits the final ridge regression model using the optimal lambda value.

Description:

This function accepts two arguments:

X: Matrix of predictors . Each row represents an observation, and each column represents a predictor. It can be a mix of continuous, discrete, and binary predictors.

y: Response variable. For binary outcomes, it should be a binary vector (0 or 1). For continuous outcomes, it should be a numeric vector.

Value:

The function returns a list containing two elements:

model: Fitted ridge regression model.

predicted.value: Predicted values based on the fitted model.

Example:

```r
library(MASS)
data(Boston)
str(Boston)
```

```
## 'data.frame':    506 obs. of  14 variables:
##  $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
##  $ chas   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ rm     : num  6.58 6.42 7.18 7 7.15 ...
##  $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ rad    : int  1 2 2 3 3 3 5 5 5 5 ...
##  $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
##  $ black  : num  397 397 393 395 397 ...
##  $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
##  $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

```
X <- Boston[, 2:14]
y <- Boston[,1]
ridge_results <- ridge_model(X, y) #For Continuous Response variable
```

```
## [1] "Ridge regression is being done - "
```

```
ridge_results
```

```
## $model
##
## Call:  glmnet::glmnet(x = X, y = y, alpha = 0, lambda = opt_lambda)
##
##   Df  %Dev Lambda
## 1 13 44.84 0.5375
##
## $predicted.value
##               s1
## 1   -0.627856839
## 2   -1.051320994
## 3   -3.354386868
## 4   -3.512163446
## 5   -3.515500943
##  [ reached getOption("max.print") -- omitted 501 rows ]
```

```
data(Pima.te)
str(Pima.te)
```

```
## 'data.frame':    332 obs. of  8 variables:
##  $ npreg: int  6 1 1 3 2 5 0 1 3 9 ...
##  $ glu  : int  148 85 89 78 197 166 118 103 126 119 ...
##  $ bp   : int  72 66 66 50 70 72 84 30 88 80 ...
##  $ skin : int  35 29 23 32 45 19 47 38 41 35 ...
##  $ bmi  : num  33.6 26.6 28.1 31 30.5 25.8 45.8 43.3 39.3 29 ...
##  $ ped  : num  0.627 0.351 0.167 0.248 0.158 0.587 0.551 0.183 0.704 0.263 ...
##  $ age  : int  50 31 21 26 53 51 31 33 27 29 ...
##  $ type : Factor w/ 2 levels "No","Yes": 2 1 1 2 2 2 2 1 1 2 ...
```

```r
X <- Pima.te[, 1:7]
y <- Pima.te[,8]
ridge_results <- ridge_model(X, as.numeric(y) - 1) #For Binary Response variable
```

```
## [1] "Ridge regression is being done - "
```

```r
ridge_results
```

```
## $model
##
## Call:  glmnet::glmnet(x = X, y = y, alpha = 0, lambda = opt_lambda)
##
##   Df  %Dev  Lambda
## 1  7 35.63 0.02442
##
## $predicted.value
##                s1
## 1    0.6392620116
## 2   -0.0251996658
## 3   -0.0502536493
## 4    0.0091178773
## 5    0.7823994875
##  [ reached getOption("max.print") -- omitted 327 rows ]
```

0.3.3 3. Lasso Regression (`lasso_model`)

The lasso_model function fits a Lasso regression model where an L1 penalty is applied to the standard linear regression objective function. L1 penalty has the effect of forcing some of the coefficient estimates to be exactly equal to zero, which means this model is type of variable selection. The function using the glmnet package with cross-validation to select the optimal lambda value. We have used 10-fold cross-validation as the standard here.

Description:

This function accepts three arguments:

X: Matrix of predictors . Each row represents an observation, and each column represents a predictor. It can be a mix of continuous, discrete, and binary predictors.

y: Response variable. For binary outcomes, it should be a binary vector (0 or 1). For continuous outcomes, it should be a numeric vector.

It returns a list containing the fitted model and predicted values.

Example:

```r
library(MASS)
data(Boston)
str(Boston)
```

```
## 'data.frame':    506 obs. of  14 variables:
##  $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
```

```
##  $ chas   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ rm     : num  6.58 6.42 7.18 7 7.15 ...
##  $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ rad    : int  1 2 2 3 3 3 5 5 5 5 ...
##  $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
##  $ black  : num  397 397 393 395 397 ...
##  $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
##  $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

```
X <- Boston[, 2:14]
y <- Boston[,1]
lasso_results <- lasso_model(X, y) #For Continuous Response variable
```

```
## [1] "Lasso regression is being done - "
```

```
lasso_results
```

```
## $model
##
## Call:  glmnet::glmnet(x = X, y = y, alpha = 1, lambda = opt_lambda)
##
##   Df  %Dev  Lambda
## 1 12 45.37 0.02024
##
## $predicted.value
##              s1
## 1   -0.79373280
## 2   -1.06527176
## 3   -3.79854045
## 4   -3.96448809
## 5   -4.13566421
##  [ reached getOption("max.print") -- omitted 501 rows ]
```

```
data(Pima.te)
str(Pima.te)
```

```
## 'data.frame':    332 obs. of  8 variables:
##  $ npreg: int  6 1 1 3 2 5 0 1 3 9 ...
##  $ glu  : int  148 85 89 78 197 166 118 103 126 119 ...
##  $ bp   : int  72 66 66 50 70 72 84 30 88 80 ...
##  $ skin : int  35 29 23 32 45 19 47 38 41 35 ...
##  $ bmi  : num  33.6 26.6 28.1 31 30.5 25.8 45.8 43.3 39.3 29 ...
##  $ ped  : num  0.627 0.351 0.167 0.248 0.158 0.587 0.551 0.183 0.704 0.263 ...
##  $ age  : int  50 31 21 26 53 51 31 33 27 29 ...
##  $ type : Factor w/ 2 levels "No","Yes": 2 1 1 2 2 2 2 1 1 2 ...
```

```
X <- Pima.te[, 1:7]
y <- Pima.te[,8]
lasso_results <- lasso_model(X, as.numeric(y) - 1) #For Binary Response variable
```

```
## [1] "Lasso regression is being done - "
```

```
lasso_results
```

```
## $model
##
## Call:  glmnet::glmnet(x = X, y = y, alpha = 1, lambda = opt_lambda)
##
##   Df  %Dev  Lambda
## 1  6 35.49 0.01033
##
## $predicted.value
##                s1
## 1    6.181752e-01
## 2   -2.640010e-02
## 3   -3.430702e-02
## 4   -4.518609e-03
## 5    7.662633e-01
##  [ reached getOption("max.print") -- omitted 327 rows ]
```

0.3.4 4. Elastic Net Regression (`elastic_net_regression`)

The elastic_net_regression function fits an elastic net regression model using the glmnet package with cross-validated alpha selection. It selects an optimal alpha value (The alpha parameter in elastic net regression controls the balance between the L1 (Lasso) and L2 (Ridge) penalties in the regularization term. A value of 1 corresponds to pure Lasso regression, while a value of 0 corresponds to pure Ridge regression. Values between 0 and 1 represent a mixture of Lasso and Ridge penalties).

Description:

This function accepts two arguments:

X: Matrix of predictors . Each row represents an observation, and each column represents a predictor. It can be a mix of continuous, discrete, and binary predictors.

y: Response variable. For binary outcomes, it should be a binary vector (0 or 1). For continuous outcomes, it should be a numeric vector.

It returns a list containing three elements:

model: Fitted elastic net regression model object.

optimal.alpha: Optimal alpha value selected through cross-validation.

predicted.value: Predicted values based on the fitted model.

Example:

```
library(MASS)
data(Boston)
str(Boston)
```

```
## 'data.frame':    506 obs. of  14 variables:
##  $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
##  $ chas   : int  0 0 0 0 0 0 0 0 0 0 ...
```

```
##  $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ rm     : num  6.58 6.42 7.18 7 7.15 ...
##  $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ rad    : int  1 2 2 3 3 3 5 5 5 5 ...
##  $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
##  $ black  : num  397 397 393 395 397 ...
##  $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
##  $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

```
X <- Boston[, 2:14]
y <- Boston[,1]
elastic_net_results <- elastic_net_regression(X, y) #For Continuous Response variable
elastic_net_results
```

```
## $model
##
## Call:  glmnet::glmnet(x = X, y = y, family = family_type, alpha = opt_alpha,      lambda = opt_lambda
##
##   Df  %Dev Lambda
## 1 12 45.31 0.0358
##
## $optimal.alpha
## [1] 0.9
##
## $predicted.value
##               s1
## 1   -0.795079656
## 2   -1.092680696
## 3   -3.723765181
## 4   -3.860681592
## 5   -4.010173508
##  [ reached getOption("max.print") -- omitted 501 rows ]
```

```
data(Pima.te)
str(Pima.te)
```

```
## 'data.frame':    332 obs. of  8 variables:
##  $ npreg: int  6 1 1 3 2 5 0 1 3 9 ...
##  $ glu  : int  148 85 89 78 197 166 118 103 126 119 ...
##  $ bp   : int  72 66 66 50 70 72 84 30 88 80 ...
##  $ skin : int  35 29 23 32 45 19 47 38 41 35 ...
##  $ bmi  : num  33.6 26.6 28.1 31 30.5 25.8 45.8 43.3 39.3 29 ...
##  $ ped  : num  0.627 0.351 0.167 0.248 0.158 0.587 0.551 0.183 0.704 0.263 ...
##  $ age  : int  50 31 21 26 53 51 31 33 27 29 ...
##  $ type : Factor w/ 2 levels "No","Yes": 2 1 1 2 2 2 2 1 1 2 ...
```

```
X <- Pima.te[, 1:7]
y <- Pima.te[,8]
elastic_net_results <- elastic_net_regression(X, as.numeric(y) - 1) #For Binary Response variable
elastic_net_results
```

```
## $model
##
## Call:  glmnet::glmnet(x = X, y = y, family = family_type, alpha = opt_alpha,      lambda = opt_lambda
##
##   Df %Dev  Lambda
## 1  6 31.5 0.01562
##
## $optimal.alpha
## [1] 0.5
##
## $predicted.value
##             s1
## 1   0.67598830
## 2   0.05100028
## 3   0.04350028
## 4   0.05784590
## 5   0.81274344
##  [ reached getOption("max.print") -- omitted 327 rows ]
```

0.3.5 5. Random Forest (`rf_model`)

A Random Forest is machine learning model that combines the predictions of multiple decision trees to improve the overall predictive accuracy and reduce overfitting.

The number of trees (ntree) and the number of variables randomly sampled as candidates at each split (mtry) are set to default values. For classification tasks, mtry is set to the square root of the number of predictors (sqrt(ncol(X))), and for regression tasks, it is set to one-third of the number of predictors (ncol(X) / 3). These are default values.

Description:

This function accepts two arguments:

X: Matrix of predictors . Each row represents an observation, and each column represents a predictor. It can be a mix of continuous, discrete, and binary predictors.

y: Response variable. For binary outcomes, it should be a binary vector (0 or 1). For continuous outcomes, it should be a numeric vector.

The function returns a fitted Random Forest model object.

Example:

```
library(MASS)
data(Boston)
str(Boston)
```

```
## 'data.frame':    506 obs. of  14 variables:
##  $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
##  $ chas   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ rm     : num  6.58 6.42 7.18 7 7.15 ...
##  $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
```

```
##  $ rad    : int  1 2 2 3 3 3 5 5 5 5 ...
##  $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
##  $ black  : num  397 397 393 395 397 ...
##  $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
##  $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

```
X <- Boston[, 2:14]
y <- Boston[,1]
rf_results <- rf_model(X, y) #For Continuous Response variable
```

```
## [1] "Random Forest method is being applied - "
```

```
rf_results
```

```
##
## Call:
##  randomForest(x = X, y = y, ntree = 100, mtry = mtry, importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 100
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 36.06918
##                     % Var explained: 51.15
```

```
rf_results$predicted
```

```
##         1         2         3         4         5
## 0.31527430 0.14198410 0.12122700 0.05358639 0.04188990
##  [ reached getOption("max.print") -- omitted 501 entries ]
```

```
rf_results$importance
```

```
##            %IncMSE IncNodePurity
## zn       0.0635800      2.454197
## indus    2.8303987   1513.333265
##  [ reached getOption("max.print") -- omitted 11 rows ]
```

```
data(Pima.te)
str(Pima.te)
```

```
## 'data.frame':    332 obs. of  8 variables:
##  $ npreg: int  6 1 1 3 2 5 0 1 3 9 ...
##  $ glu  : int  148 85 89 78 197 166 118 103 126 119 ...
##  $ bp   : int  72 66 66 50 70 72 84 30 88 80 ...
##  $ skin : int  35 29 23 32 45 19 47 38 41 35 ...
##  $ bmi  : num  33.6 26.6 28.1 31 30.5 25.8 45.8 43.3 39.3 29 ...
##  $ ped  : num  0.627 0.351 0.167 0.248 0.158 0.587 0.551 0.183 0.704 0.263 ...
##  $ age  : int  50 31 21 26 53 51 31 33 27 29 ...
##  $ type : Factor w/ 2 levels "No","Yes": 2 1 1 2 2 2 2 1 1 2 ...
```

```r
X <- Pima.te[, 1:7]
y <- Pima.te[,8]
rf_results <- rf_model(X, as.numeric(y) - 1) #For Binary Response variable
```

```
## [1] "Random Forest method is being applied - "
```

```r
rf_results
```

```
##
## Call:
##  randomForest(x = X, y = y, ntree = 100, mtry = mtry, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 23.19%
## Confusion matrix:
##     0  1 class.error
## 0 190 33   0.1479821
##  [ reached getOption("max.print") -- omitted 1 row ]
```

```r
rf_results$predicted
```

```
## 1 2 3 4 5
## 0 0 0 0 0
##  [ reached getOption("max.print") -- omitted 327 entries ]
## Levels: 0 1
```

```r
rf_results$importance
```

```
##                  0           1 MeanDecreaseAccuracy MeanDecreaseGini
## npreg  0.011155310 0.012651616          0.011726635         12.13414
##  [ reached getOption("max.print") -- omitted 6 rows ]
```

0.3.6 6. Bagging (`bagged_model`)

This function enables bagging with different machine learning models. Bagging, short for Bootstrap Aggregating, is an ensemble learning technique that improves the stability and accuracy of models by training multiple models(of a single type) on different subsets of the training data and combining their predictions.

Description:

This function accepts the following arguments:

- X: Matrix of predictors. Each row represents an observation, and each column represents a predictor. It can include a mix of continuous, discrete, and binary predictors.
- y: Response variable. For binary outcomes, it should be a binary vector (0 or 1). For continuous outcomes, it should be a numeric vector.
- r.bagging: Number of bagging iterations.

- modelType: Type of machine learning model to use for bagging. Options include “linear”, “ridge”, “lasso”, “elastic_net”, and “random_forest”. If the model type is “random_forest”, the function warns the user that further bagging is not allowed since random forests already use bagging internally.

The function iteratively performs bagging by resampling the original dataset, fitting the model to each resampled dataset, and aggregating the predictions (In case of continuous response variable, taking the simple mean and n case of binary response variable, majority vote).

Value:

The function returns a list with the following elements:

- fitted.values: Predicted values based on the fitted bagged model.
- naive: A vector containing the count of naive variables selected in each bagging. (In case of a ridge model, the naive value is not needed, because all the variables are selected when model is fit for each resampling.)

Examples:

```
library(MASS)
data(Boston)
str(Boston)
```

```
## 'data.frame':    506 obs. of  14 variables:
##  $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
##  $ chas   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ rm     : num  6.58 6.42 7.18 7 7.15 ...
##  $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ rad    : int  1 2 2 3 3 3 5 5 5 5 ...
##  $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
##  $ black  : num  397 397 393 395 397 ...
##  $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
##  $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

```
X <- Boston[, 2:14]
y <- Boston[,1]
bagged_results <- bagged_model(X, y, r.bagging = 10, modelType = "linear")
```

```
## [1] "linear"
## [1] "Linear regression is being done - "
## [1] "linear"
## [1] "Linear regression is being done - "
## [1] "linear"
## [1] "Linear regression is being done - "
## [1] "linear"
## [1] "Linear regression is being done - "
## [1] "linear"
```

```
## [1] "Linear regression is being done - "
## [1] "linear"
## [1] "Linear regression is being done - "
## [1] "linear"
## [1] "Linear regression is being done - "
## [1] "linear"
## [1] "Linear regression is being done - "
## [1] "linear"
## [1] "Linear regression is being done - "
## [1] "linear"
## [1] "Linear regression is being done - "
```

```
print(bagged_results)
```

```
## $fitted.values
## [1] 6.089789 8.370473 3.680168 2.319192 4.979019
##  [ reached getOption("max.print") -- omitted 501 entries ]
##
## $naive
## [1] 10  0  0  4  2
##  [ reached getOption("max.print") -- omitted 8 entries ]
```

0.3.7 7. Ensemble Learning (`ensemble_model`)

Ensemble learning is a powerful technique in machine learning where multiple models are combined to improve overall performance. The intuition behind ensemble learning is that by combining the predictions of multiple models, we can reduce the risk of making a wrong prediction and often achieve better results than any individual model.

Here we are using 2 models as base learners - Elastic Net provides regularization and variable selection capabilities, while Random Forest offers robustness and non-linear modeling capabilities.

Description:

This function accepts two arguments:

X: Matrix of predictors. Each row represents an observation, and each column represents a predictor. It can be a mix of continuous, discrete, and binary predictors.

y: Response variable. For binary outcomes, it should be a binary vector (0 or 1). For continuous outcomes, it should be a numeric vector.

The function returns a list containing the predictions and accuracies of the individual models and the ensemble model(ensembled_prediction). (For calculating the predictions - In case of continuous response variable, it is taking the simple mean and in case of binary response variable, it takes majority vote.)

elastic.net: List containing information about the Elastic Net model, including predicted values.

random.forest: List containing information about the Random Forest model, including predicted values.

Example:

```
library(MASS)
data(Boston)
str(Boston)
```

```
## 'data.frame':    506 obs. of  14 variables:
##  $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
##  $ chas   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ rm     : num  6.58 6.42 7.18 7 7.15 ...
##  $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ rad    : int  1 2 2 3 3 3 5 5 5 5 ...
##  $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
##  $ black  : num  397 397 393 395 397 ...
##  $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
##  $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

```
X <- Boston[, 2:14]
y <- Boston[,1]
ensemble_predictions <- ensemble_predict(X, y) #For Continuous Response variable
```

```
## [1] "Random Forest method is being applied - "
```

```
ensemble_predictions
```

```
## $ensembled_prediction
##          1          2          3          4          5
## -0.7781527 -1.0789881 -3.7596729 -3.9151359 -4.0706724
##  [ reached getOption("max.print") -- omitted 501 entries ]
##
## $elastic.net
## $elastic.net$model
##
## Call:  glmnet::glmnet(x = X, y = y, family = family_type, alpha = opt_alpha,      lambda = opt_lambda
##
##   Df  %Dev  Lambda
## 1 12 45.34 0.04442
##
## $elastic.net$optimal.alpha
## [1] 0.5
##
## $elastic.net$predicted.value
##               s1
## 1   -0.778152701
## 2   -1.078988106
## 3   -3.759672893
## 4   -3.915135925
## 5   -4.070672397
##  [ reached getOption("max.print") -- omitted 501 rows ]
##
##
## $random.forest
##
## Call:
```

```
##  randomForest(x = X, y = y, ntree = 100, mtry = mtry, importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 100
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 35.49756
##                     % Var explained: 51.93
```

```
data(Pima.te)
str(Pima.te)
```

```
## 'data.frame':    332 obs. of  8 variables:
##  $ npreg: int  6 1 1 3 2 5 0 1 3 9 ...
##  $ glu  : int  148 85 89 78 197 166 118 103 126 119 ...
##  $ bp   : int  72 66 66 50 70 72 84 30 88 80 ...
##  $ skin : int  35 29 23 32 45 19 47 38 41 35 ...
##  $ bmi  : num  33.6 26.6 28.1 31 30.5 25.8 45.8 43.3 39.3 29 ...
##  $ ped  : num  0.627 0.351 0.167 0.248 0.158 0.587 0.551 0.183 0.704 0.263 ...
##  $ age  : int  50 31 21 26 53 51 31 33 27 29 ...
##  $ type : Factor w/ 2 levels "No","Yes": 2 1 1 2 2 2 2 1 1 2 ...
```

```
X <- Pima.te[, 1:7]
y <- Pima.te[,8]
ensemble_predictions <- ensemble_predict(X, as.numeric(y) - 1) #For Binary Response variable
```

```
## [1] "Random Forest method is being applied - "
```

```
ensemble_predictions
```

```
## $ensembled_prediction
## 1 2 3 4 5
## 1 0 0 0 1
##  [ reached getOption("max.print") -- omitted 327 entries ]
##
## $elastic.net
## $elastic.net$model
##
## Call:  glmnet::glmnet(x = X, y = y, family = family_type, alpha = opt_alpha,      lambda = opt_lambda
##
##   Df  %Dev  Lambda
## 1  6 31.34 0.02162
##
## $elastic.net$optimal.alpha
## [1] 0.3
##
## $elastic.net$predicted.value
##             s1
## 1   0.67226475
## 2   0.05562853
## 3   0.04621946
## 4   0.06224374
## 5   0.80897761
```

```
##  [ reached getOption("max.print") -- omitted 327 rows ]
##
##
## $random.forest
##
## Call:
##  randomForest(x = X, y = y, ntree = 100, mtry = mtry, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 21.08%
## Confusion matrix:
##     0  1 class.error
## 0 196 27   0.1210762
##  [ reached getOption("max.print") -- omitted 1 row ]
```

0.3.8 8. Variable Screening (`variable_screening`)

Variable Screening for Top Predictors

This function performs Univariate filtering (Univariate filtering evaluates each feature's individual association with the target variable through statistical tests, selecting the most significant features based on measures such as correlation, ANOVA, or chi-squared tests.) on each column of the predictor matrix X based on the provided output variable y.

Various statistical tests are applied depending on the data types of X and y to identify significant predictors -

If y is numeric and X is numeric, Pearson's correlation test is applied.

If y is numeric and X is binary, Kruskal-Wallis test or Simple Linear Regression is used.

If y is binary and X is numeric, ANOVA or Fisher's Exact Test is used.

If both y and X are binary, Chi-squared test is applied.

Arguments

X: Matrix of predictors. Each row represents an observation, and each column represents a predictor. It can be a mix of continuous, discrete, and binary predictors.

y: Response variable. Can be continuous and normally distributed or binary responses.

K: Number of top predictors to select. If not specified, default is half the number of original number of predictor variables.

This returns a dataframe containing the selected predictor variables in order of significance (if pValue for the above tests are <0.05, then ordering is based on this value, i.e. lesser pValue => more significant predictor)

Example:

```
library(MASS)
data(survey)
str(survey)
```

```
## 'data.frame':    237 obs. of  12 variables:
##  $ Sex   : Factor w/ 2 levels "Female","Male": 1 2 2 2 2 1 2 1 2 2 ...
##  $ Wr.Hnd: num  18.5 19.5 18 18.8 20 18 17.7 17 20 18.5 ...
```

```
##  $ NW.Hnd: num  18 20.5 13.3 18.9 20 17.7 17.7 17.3 19.5 18.5 ...
##  $ W.Hnd : Factor w/ 2 levels "Left","Right": 2 1 2 2 2 2 2 2 2 2 ...
##  $ Fold  : Factor w/ 3 levels "L on R","Neither",..: 3 3 1 3 2 1 1 3 3 3 ...
##  $ Pulse : int  92 104 87 NA 35 64 83 74 72 90 ...
##  $ Clap  : Factor w/ 3 levels "Left","Neither",..: 1 1 2 2 3 3 3 3 3 3 ...
##  $ Exer  : Factor w/ 3 levels "Freq","None",..: 3 2 2 2 3 3 1 1 3 3 ...
##  $ Smoke : Factor w/ 4 levels "Heavy","Never",..: 2 4 3 2 2 2 2 2 2 2 ...
##  $ Height: num  173 178 NA 160 165 ...
##  $ M.I   : Factor w/ 2 levels "Imperial","Metric": 2 1 NA 2 2 1 1 2 2 2 ...
##  $ Age   : num  18.2 17.6 16.9 20.3 23.7 ...
```

```
X <- survey[, -10]
y <- survey$Height
selected_variables <- variable_pre_screening(X, y, 4)
```

```
## [1] "Some top predictors are being selected - "
## [1] 8.162734e-23
## [1] 8.227549e-22
## [1] 2.02001e-20
## [1] 0.3457057
## [1] 0.5530583
## [1] 0.2750344
## [1] 0.6559836
## [1] 0.001435077
## [1] 0.1655918
## [1] 0.2133218
## [1] 0.5920513
## [1] 1 2 3 8
```

```
print(selected_variables)
```

```
##      Sex Wr.Hnd NW.Hnd Exer
## 1 Female   18.5     18 Some
##  [ reached 'max' / getOption("max.print") -- omitted 236 rows ]
```

0.3.9 9. Combining Ensemble, Bagging, Models and Feature Selection (`simple_ensemble`)

The simple_ensemble function allows users to specify a set of models to include in the ensemble and to give option for top K number of predictors to be selected for the model (By default, if n<20*p, where n=sample size and p=number of explanatory variables, it selects for some top predictors cross a certain threshold value for their univariate pValue), allows users to give the number of times to sample to perform bagging for each model, and whether to perform ensemble learning of the models.

Function Arguments

X: Matrix of predictors. Each row represents an observation, and each column represents a predictor. It can be a mix of continuous, discrete, and binary predictors.

y: Response variable. Can be continuous and normally distributed or binary responses.

models: A character vector specifying the types of models to include in the ensemble. Options include "elastic_net" and "random_forest". Default is "elastic_net".

k: Top number of predictors to select in the pre-screening step. Default is NULL.

r.bagging: Number of times model will perform sampling with replacement for the samples. Default is 50.

is.ensemble: Logical parameter indicating whether to perform ensemble learning of models passed as a parameter. Default is FALSE.

```
library(MASS)
data(Boston)
str(Boston)
```

```
## 'data.frame':    506 obs. of  14 variables:
##  $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
##  $ chas   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ rm     : num  6.58 6.42 7.18 7 7.15 ...
##  $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ rad    : int  1 2 2 3 3 3 5 5 5 5 ...
##  $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
##  $ black  : num  397 397 393 395 397 ...
##  $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
##  $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

```
X <- Boston[, 2:14]
y <- Boston[,1]
results <- simple_ensemble(X, y, c("elastic_net", "random_forest"), TRUE, k=8) #For Continuous Response
```

```
## [1] "Some top predictors are being selected - "
## [1] 5.506472e-06
## [1] 1.450349e-21
## [1] 0.2094345
## [1] 3.751739e-23
## [1] 6.346703e-07
## [1] 2.854869e-16
## [1] 8.519949e-19
## [1] 2.693844e-56
## [1] 2.357127e-47
## [1] 2.942922e-11
## [1] 2.487274e-19
## [1] 2.654277e-27
## [1] 1.173987e-19
## [1]  8  9 12  4  2
##  [ reached getOption("max.print") -- omitted 3 entries ]
## [1] "Bagging is being done - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "Bagging is being done - "
## [1] "Random Forest is a bagging process! Further bagging is not allowed."
```

```
results$ensembled$random.forest$importanceSD
```

```
## NULL
```

```
data(Pima.te)
str(Pima.te)
```

```
## 'data.frame':    332 obs. of  8 variables:
##  $ npreg: int  6 1 1 3 2 5 0 1 3 9 ...
##  $ glu  : int  148 85 89 78 197 166 118 103 126 119 ...
##  $ bp   : int  72 66 66 50 70 72 84 30 88 80 ...
##  $ skin : int  35 29 23 32 45 19 47 38 41 35 ...
##  $ bmi  : num  33.6 26.6 28.1 31 30.5 25.8 45.8 43.3 39.3 29 ...
##  $ ped  : num  0.627 0.351 0.167 0.248 0.158 0.587 0.551 0.183 0.704 0.263 ...
##  $ age  : int  50 31 21 26 53 51 31 33 27 29 ...
##  $ type : Factor w/ 2 levels "No","Yes": 2 1 1 2 2 2 2 1 1 2 ...
```

```
X <- Pima.te[, 1:7]
y <- Pima.te[,8]
ensemble_predictions <- simple_ensemble(X, y, c("elastic_net"), 50, FALSE, k=4 ) #For Binary Response v
```

```
## [1] "Some top predictors are being selected - "
## [1] 9.062448e-06
## [1] 2.178114e-24
## [1] 0.001820003
## [1] 7.405458e-07
## [1] 4.58182e-09
## [1] 3.428031e-06
## [1] 1.571065e-07
## [1] 2 5 7 4
## [1] "Bagging is being done - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
```

```
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
```

```
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
## [1] "elastic_net"
## [1] "elastic net bagging - "
```

```
ensemble_predictions$elastic_net
```

```
## $fitted.values
## [1] 0.03039153 0.01369515 0.01715645 0.01136320 0.02277718
##  [ reached getOption("max.print") -- omitted 327 entries ]
##
## $naive
## [1] 50 49 50 45
```

0.4 Conclusion

The `simpleEnsembleGroup22` package provides tools for robust machine learning applications, integrating advanced regression and ensemble techniques to tackle complex predictive modeling tasks.