CS 634 Final Term Project

-Alfred Zane Rajan Velladurai

UCID: ar2257

Email address: ar2257@njit.edu

Option no: 1

Dataset: https://www.kaggle.com/mlg-ulb/creditcardfraud#creditcard.csv

Classification methods used: Linear Regression and SVM models in R

Tools:

R Studio, R packages

My Code:

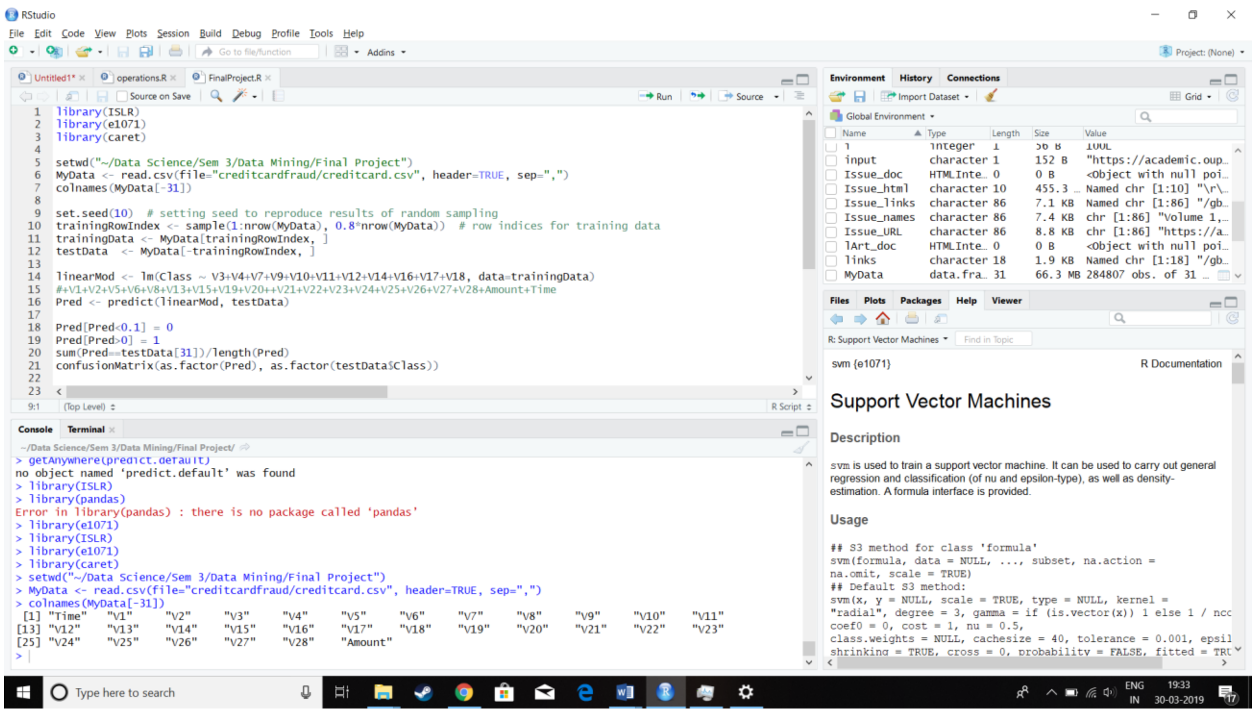
```
library(ISLR)

library(e1071)

library(caret)


setwd("~/Data Science/Sem 3/Data Mining/Final Project")

MyData <- read.csv(file="creditcardfraud/creditcard.csv", header=TRUE, sep=",")


set.seed(10)  # setting seed to reproduce results of random sampling

trainingRowIndex <- sample(1:nrow(MyData), 0.7*nrow(MyData))  # row indices
for training data

trainingData <- MyData[trainingRowIndex, ]

testData  <- MyData[-trainingRowIndex, ]


linearMod <- lm(Class ~ V3+V4+V7+V9+V10+V11+V12+V14+V16+V17+V18,
data=trainingData)

Pred <- predict(linearMod, testData)

Pred[Pred<0.1] = 0

Pred[Pred>0] = 1

sum(Pred==testData[31])/length(Pred)

confusionMatrix(as.factor(Pred), as.factor(testData$Class))


trainingData$Class = as.factor(trainingData$Class)
```

SVMmod <- svm(trainingData[c(4,5,8,10,11,12,13,15,17,18,19)], trainingData$Class, type = "C" ,kernel = "linear", tolerance = 0.01)

Pred1 <- predict(SVMmod, testData[c(4,5,8,10,11,12,13,15,17,18,19)])

confusionMatrix(Pred1, as.factor(testData$Class))

Step by step:



```r
library(ISLR)
library(e1071)
library(caret)

setwd("~/Data Science/Sem 3/Data Mining/Final Project")
MyData <- read.csv(file="creditcardfraud/creditcard.csv", header=TRUE, sep=",")
colnames(MyData[-31])

set.seed(10)  # setting seed to reproduce results of random sampling
trainingRowIndex <- sample(1:nrow(MyData), 0.8*nrow(MyData))  # row indices for training data
trainingData <- MyData[trainingRowIndex, ]
testData  <- MyData[-trainingRowIndex, ]

linearMod <- lm(Class ~ V3+V4+V7+V9+V10+V11+V12+V14+V16+V17+V18, data=trainingData)
#+V1+V2+V5+V6+V8+V13+V15+V19+V20++V21+V22+V23+V24+V25+V26+V27+V28+Amount+Time
Pred <- predict(linearMod, testData)

Pred[Pred<0.1] = 0
Pred[Pred>0] = 1
sum(Pred==testData[31])/length(Pred)
confusionMatrix(as.factor(Pred), as.factor(testData$Class))
```

```
> getAnywhere(predict.default)
no object named ‘predict.default’ was found
> library(ISLR)
> library(pandas)
Error in library(pandas) : there is no package called ‘pandas’
> library(e1071)
> library(ISLR)
> library(e1071)
> library(caret)
> setwd("~/Data Science/Sem 3/Data Mining/Final Project")
> MyData <- read.csv(file="creditcardfraud/creditcard.csv", header=TRUE, sep=",")
> colnames(MyData[-31])
 [1] "Time"   "V1"     "V2"     "V3"     "V4"     "V5"     "V6"     "V7"     "V8"     "V9"     "V10"    "V11"
[13] "V12"    "V13"    "V14"    "V15"    "V16"    "V17"    "V18"    "V19"    "V20"    "V21"    "V22"    "V23"
[25] "V24"    "V25"    "V26"    "V27"    "V28"    "Amount"
>
```

```
Name          Type        Length  Size     Value
i             integer     1       56 B     100L
input         character   1       152 B    "https://academic.oup…
Issue_doc     HTMLInte…   0       0 B      <Object with null poi…
Issue_html    character   10      455.3 …  Named chr [1:10] "\r\…
Issue_links   character   86      7.1 KB   Named chr [1:86] "/gb…
Issue_names   character   86      7.4 KB   chr [1:86] "Volume 1,…
Issue_URL     character   86      8.8 KB   chr [1:86] "https://a…
lArt_doc      HTMLInte…   0       0 B      <Object with null poi…
links         character   18      1.9 KB   Named chr [1:18] "/gb…
MyData        data.fra…   31      66.3 MB  284807 obs. of 31 …
```

svm {e1071} R Documentation

Support Vector Machines

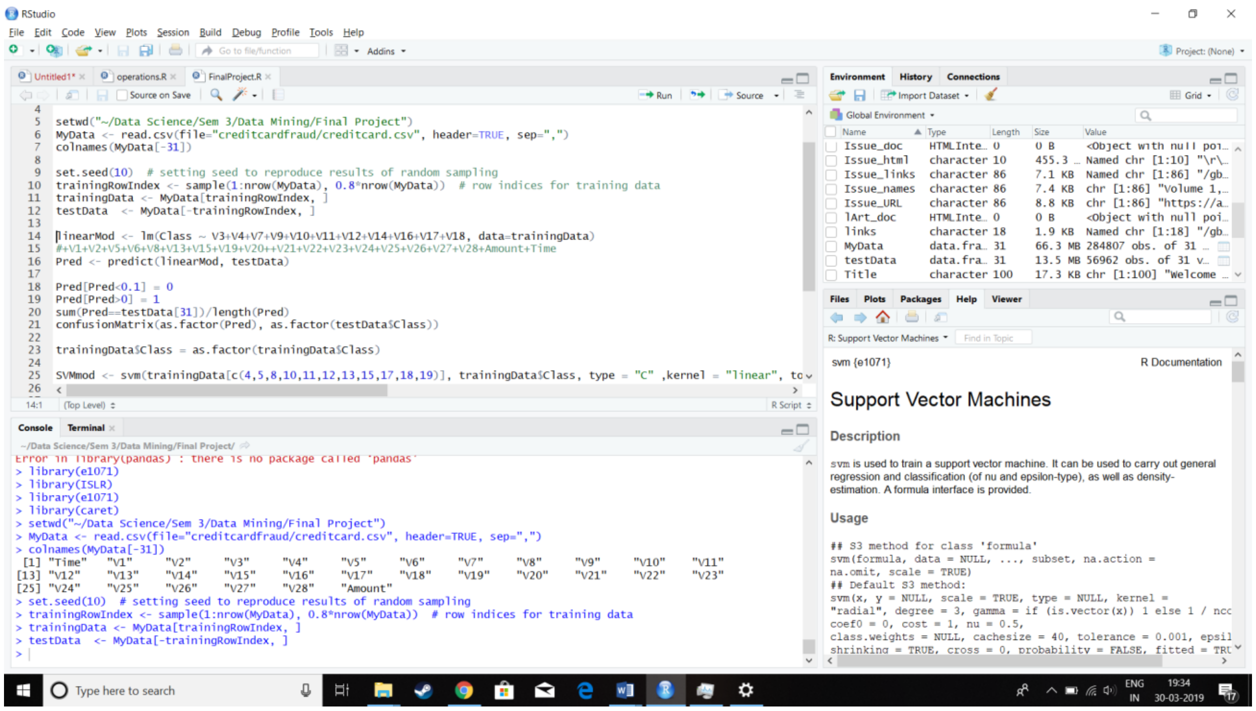
Description

svm is used to train a support vector machine. It can be used to carry out general regression and classification (of nu and epsilon-type), as well as density-estimation. A formula interface is provided.

Usage

```
## S3 method for class 'formula'
svm(formula, data = NULL, ..., subset, na.action =
na.omit, scale = TRUE)
## Default S3 method:
svm(x, y = NULL, scale = TRUE, type = NULL, kernel =
"radial", degree = 3, gamma = if (is.vector(x)) 1 else 1 / ncc
coef0 = 0, cost = 1, nu = 0.5,
class.weights = NULL, cachesize = 40, tolerance = 0.001, epsil
shrinking = TRUE, cross = 0, probability = FALSE, fitted = TRU
```

Reading data



```
setwd("~/Data Science/Sem 3/Data Mining/Final Project")
MyData <- read.csv(file="creditcardfraud/creditcard.csv", header=TRUE, sep=",")
colnames(MyData[-31])

set.seed(10)  # setting seed to reproduce results of random sampling
trainingRowIndex <- sample(1:nrow(MyData), 0.8*nrow(MyData))  # row indices for training data
trainingData <- MyData[trainingRowIndex, ]
testData  <- MyData[-trainingRowIndex, ]

linearMod <- lm(Class ~ V3+V4+V7+V9+V10+V11+V12+V14+V16+V17+V18, data=trainingData)
#+V1+V2+V5+V6+V8+V13+V15+V19+V20++V21+V22+V23+V24+V25+V26+V27+V28+Amount+Time
Pred <- predict(linearMod, testData)

Pred[Pred<0.1] = 0
Pred[Pred>0] = 1
sum(Pred==testData[31])/length(Pred)
confusionMatrix(as.factor(Pred), as.factor(testData$Class))

trainingData$Class = as.factor(trainingData$Class)

SVMmod <- svm(trainingData[c(4,5,8,10,11,12,13,15,17,18,19)], trainingData$Class, type = "C" ,kernel = "linear", to
```

```
Error in library(pandas) : there is no package called 'pandas'
> library(e1071)
> library(ISLR)
> library(e1071)
> library(caret)
> setwd("~/Data Science/Sem 3/Data Mining/Final Project")
> MyData <- read.csv(file="creditcardfraud/creditcard.csv", header=TRUE, sep=",")
> colnames(MyData[-31])
 [1] "Time"   "V1"     "V2"     "V3"     "V4"     "V5"     "V6"     "V7"     "V8"     "V9"     "V10"    "V11"
[13] "V12"    "V13"    "V14"    "V15"    "V16"    "V17"    "V18"    "V19"    "V20"    "V21"    "V22"    "V23"
[25] "V24"    "V25"    "V26"    "V27"    "V28"    "Amount"
> set.seed(10)  # setting seed to reproduce results of random sampling
> trainingRowIndex <- sample(1:nrow(MyData), 0.8*nrow(MyData))  # row indices for training data
> trainingData <- MyData[trainingRowIndex, ]
> testData  <- MyData[-trainingRowIndex, ]
> 
```

Name	Type	Length	Size	Value
Issue_doc	HTMLInte…	0	0 B	<Object with null poi…
Issue_html	character	10	455.3 …	Named chr [1:10] "\r\…
Issue_links	character	86	7.1 KB	Named chr [1:86] "/gb…
Issue_names	character	86	7.4 KB	chr [1:86] "Volume 1,…
Issue_URL	character	86	8.8 KB	chr [1:86] "https://a…
lArt_doc	HTMLInte…	0	0 B	<Object with null poi…
links	character	18	1.9 KB	Named chr [1:18] "/gb…
MyData	data.fra…	31	66.3 MB	284807 obs. of 31 …
testData	data.fra…	31	13.5 MB	56962 obs. of 31 v…
Title	character	100	17.3 KB	chr [1:100] "Welcome …

svm {e1071}   R Documentation

Support Vector Machines

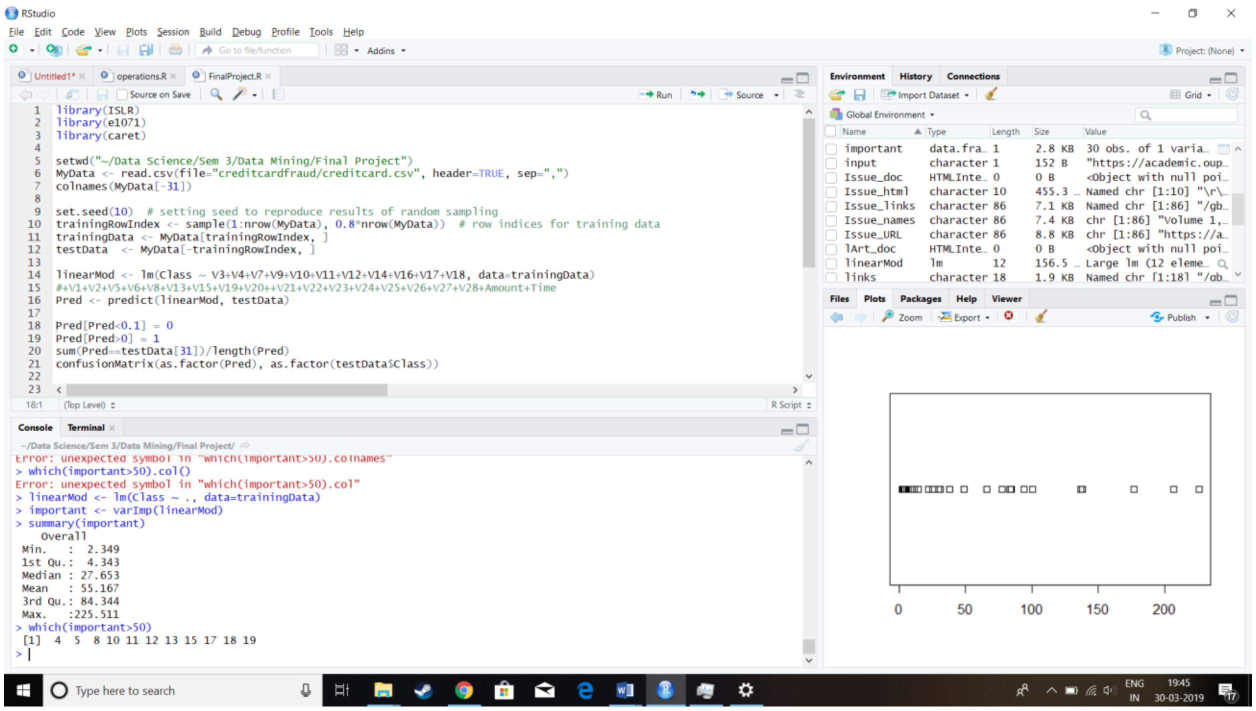
Description

svm is used to train a support vector machine. It can be used to carry out general regression and classification (of nu and epsilon-type), as well as density-estimation. A formula interface is provided.

Usage

```
## S3 method for class 'formula'
svm(formula, data = NULL, ..., subset, na.action =
na.omit, scale = TRUE)
## Default S3 method:
svm(x, y = NULL, scale = TRUE, type = NULL, kernel =
"radial", degree = 3, gamma = if (is.vector(x)) 1 else 1 / ncc
coef0 = 0, cost = 1, nu = 0.5,
class.weights = NULL, cachesize = 40, tolerance = 0.001, epsil
shrinking = TRUE, cross = 0, probability = FALSE, fitted = TRU
```

Splitting Dataset

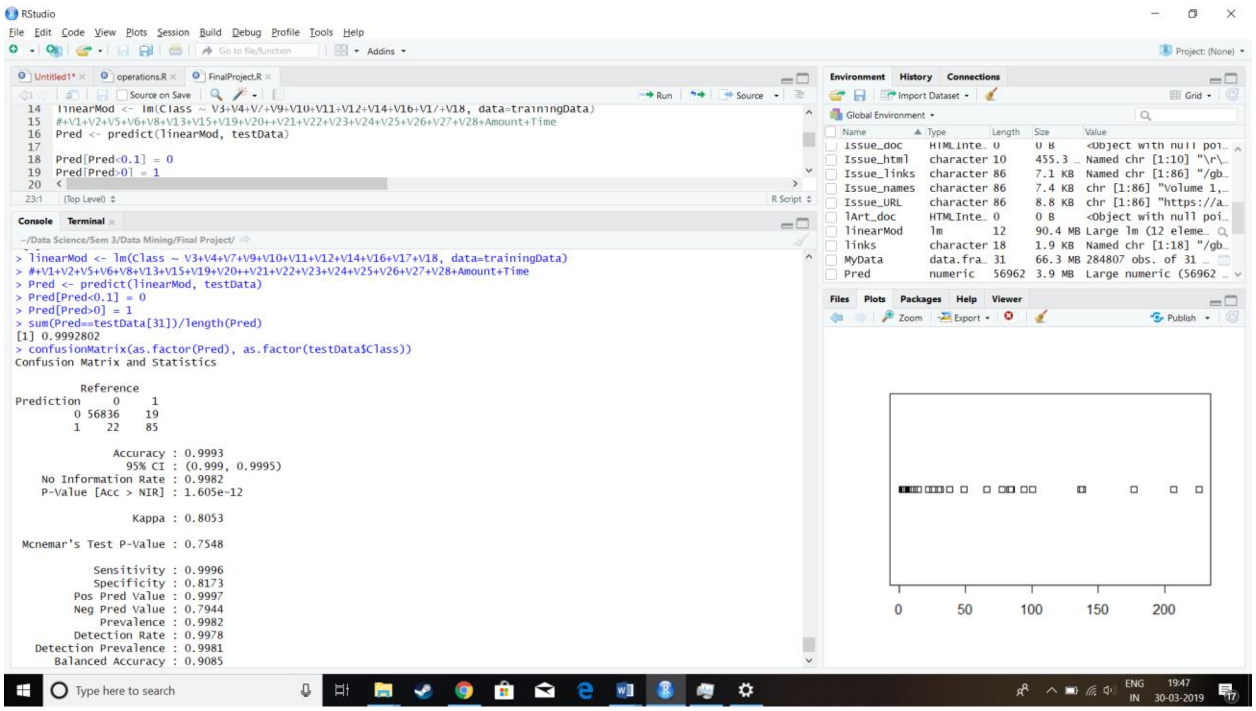


```
library(ISLR)
library(e1071)
library(caret)

setwd("~/Data Science/Sem 3/Data Mining/Final Project")
MyData <- read.csv(file="creditcardfraud/creditcard.csv", header=TRUE, sep=",")
colnames(MyData[-31])

set.seed(10)  # setting seed to reproduce results of random sampling
trainingRowIndex <- sample(1:nrow(MyData), 0.8*nrow(MyData))  # row indices for training data
trainingData <- MyData[trainingRowIndex, ]
testData  <- MyData[-trainingRowIndex, ]

linearMod <- lm(Class ~ V3+V4+V7+V9+V10+V11+V12+V14+V16+V17+V18, data=trainingData)
#+V1+V2+V5+V6+V8+V13+V15+V19+V20++V21+V22+V23+V24+V25+V26+V27+V28+Amount+Time
Pred <- predict(linearMod, testData)

Pred[Pred<0.1] = 0
Pred[Pred>0] = 1
sum(Pred==testData[31])/length(Pred)
confusionMatrix(as.factor(Pred), as.factor(testData$Class))
```

```
Error: unexpected symbol in "which(important>50).colnames"
> which(important>50).col()
Error: unexpected symbol in "which(important>50).col"
> linearMod <- lm(Class ~ ., data=trainingData)
> important <- varImp(linearMod)
> summary(important)
    Overall
 Min.   :  2.349
 1st Qu.:  4.343
 Median : 27.653
 Mean   : 55.167
 3rd Qu.: 84.344
 Max.   :225.511
> which(important>50)
 [1]  4  5  8 10 11 12 13 15 17 18 19
> |
```

Name	Type	Length	Size	Value
important	data.fra…	1	2.8 KB	30 obs. of 1 varia…
input	character	1	152 B	"https://academic.oup…
Issue_doc	HTMLInte…	0	0 B	<Object with null poi…
Issue_html	character	10	455.3 …	Named chr [1:10] "\r\…
Issue_links	character	86	7.1 KB	Named chr [1:86] "/gb…
Issue_names	character	86	7.4 KB	chr [1:86] "Volume 1,…
Issue_URL	character	86	8.8 KB	chr [1:86] "https://a…
lArt_doc	HTMLInte…	0	0 B	<Object with null poi…
linearMod	lm	12	156.5 …	Large lm (12 eleme…
links	character	18	1.9 KB	Named chr [1:18] "/gb…

Feauture Selection

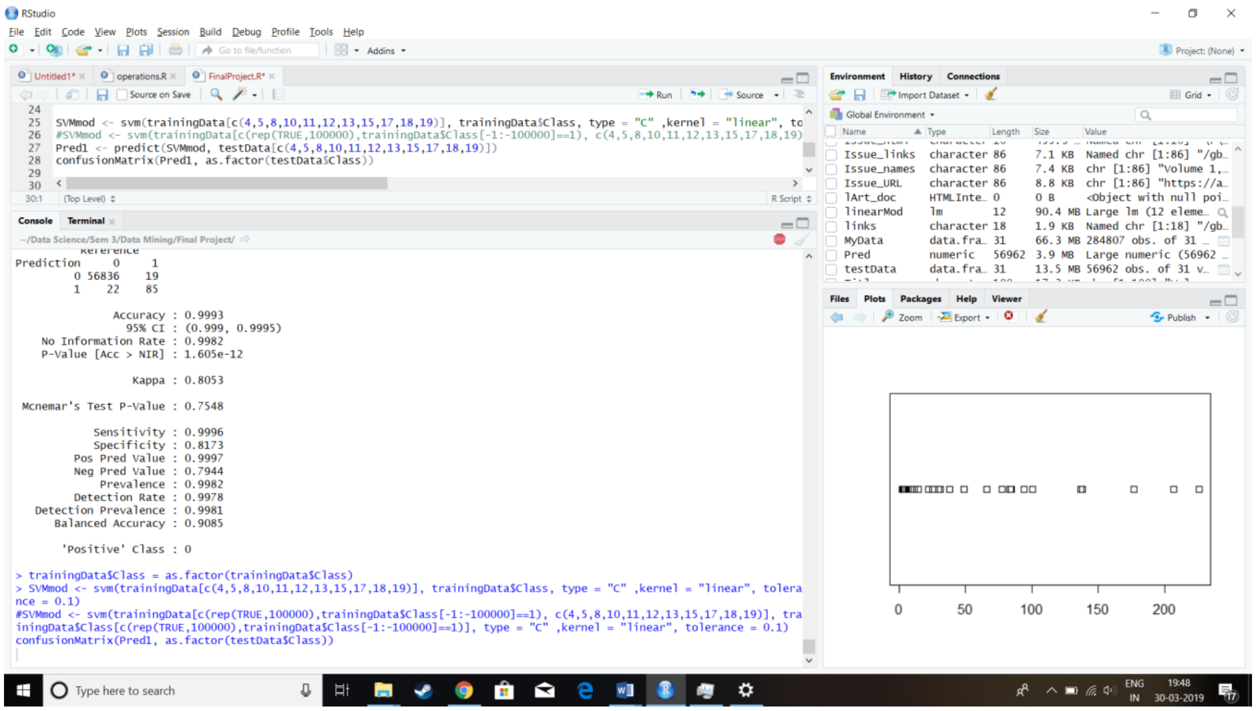


```r
linearMod <- lm(Class ~ V3+V4+V7+V9+V10+V11+V12+V14+V16+V17+V18, data=trainingData)
#+V1+V2+V5+V6+V8+V13+V15+V19+V20++V21+V22+V23+V24+V25+V26+V27+V28+Amount+Time
Pred <- predict(linearMod, testData)

Pred[Pred<0.1] = 0
Pred[Pred>0] = 1
```

```
> linearMod <- lm(Class ~ V3+V4+V7+V9+V10+V11+V12+V14+V16+V17+V18, data=trainingData)
> #+V1+V2+V5+V6+V8+V13+V15+V19+V20++V21+V22+V23+V24+V25+V26+V27+V28+Amount+Time
> Pred <- predict(linearMod, testData)
> Pred[Pred<0.1] = 0
> Pred[Pred>0] = 1
> sum(Pred==testData[31])/length(Pred)
[1] 0.9992802
> confusionMatrix(as.factor(Pred), as.factor(testData$Class))
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 56836    19
         1    22    85

               Accuracy : 0.9993
                 95% CI : (0.999, 0.9995)
    No Information Rate : 0.9982
    P-Value [Acc > NIR] : 1.605e-12

                  Kappa : 0.8053

 Mcnemar's Test P-Value : 0.7548

            Sensitivity : 0.9996
            Specificity : 0.8173
         Pos Pred Value : 0.9997
         Neg Pred Value : 0.7944
             Prevalence : 0.9982
         Detection Rate : 0.9978
   Detection Prevalence : 0.9981
      Balanced Accuracy : 0.9085
```

Name	Type	Length	Size	Value
Issue_doc	HTMLInte…	0	0 B	<Object with null poi…
Issue_html	character	10	455.3 …	Named chr [1:10] "\r\…
Issue_links	character	86	7.1 KB	Named chr [1:86] "/gb…
Issue_names	character	86	7.4 KB	chr [1:86] "Volume 1,…
Issue_URL	character	86	8.8 KB	chr [1:86] "https://a…
lArt_doc	HTMLInte…	0	0 B	<Object with null poi…
linearMod	lm	12	90.4 MB	Large lm (12 eleme…
links	character	18	1.9 KB	Named chr [1:18] "/gb…
MyData	data.fra…	31	66.3 MB	284807 obs. of 31 …
Pred	numeric	56962	3.9 MB	Large numeric (56962 …

Linear Model



```r
SVMmod <- svm(trainingData[c(4,5,8,10,11,12,13,15,17,18,19)], trainingData$Class, type = "C" ,kernel = "linear", to
#SVMmod <- svm(trainingData[c(rep(TRUE,100000),trainingData$Class[-1:-100000]==1), c(4,5,8,10,11,12,13,15,17,18,19)
Pred1 <- predict(SVMmod, testData[c(4,5,8,10,11,12,13,15,17,18,19)])
confusionMatrix(Pred1, as.factor(testData$Class))
```

```
Prediction     0     1
         0 56836    19
         1    22    85

               Accuracy : 0.9993
                 95% CI : (0.999, 0.9995)
    No Information Rate : 0.9982
    P-Value [Acc > NIR] : 1.605e-12

                  Kappa : 0.8053

 Mcnemar's Test P-Value : 0.7548

            Sensitivity : 0.9996
            Specificity : 0.8173
         Pos Pred Value : 0.9997
         Neg Pred Value : 0.7944
             Prevalence : 0.9982
         Detection Rate : 0.9978
   Detection Prevalence : 0.9981
      Balanced Accuracy : 0.9085

       'Positive' Class : 0

> trainingData$Class = as.factor(trainingData$Class)
> SVMmod <- svm(trainingData[c(4,5,8,10,11,12,13,15,17,18,19)], trainingData$Class, type = "C" ,kernel = "linear", tolera
nce = 0.1)
#SVMmod <- svm(trainingData[c(rep(TRUE,100000),trainingData$Class[-1:-100000]==1), c(4,5,8,10,11,12,13,15,17,18,19)], tra
iningData$Class[c(rep(TRUE,100000),trainingData$Class[-1:-100000]==1)], type = "C" ,kernel = "linear", tolerance = 0.1)
confusionMatrix(Pred1, as.factor(testData$Class))
```

Name	Type	Length	Size	Value
Issue_links	character	86	7.1 KB	Named chr [1:86] "/gb…
Issue_names	character	86	7.4 KB	chr [1:86] "Volume 1,…
Issue_URL	character	86	8.8 KB	chr [1:86] "https://a…
lArt_doc	HTMLInte…	0	0 B	<Object with null poi…
linearMod	lm	12	90.4 MB	Large lm (12 eleme…
links	character	18	1.9 KB	Named chr [1:18] "/gb…
MyData	data.fra…	31	66.3 MB	284807 obs. of 31 …
Pred	numeric	56962	3.9 MB	Large numeric (56962 …
testData	data.fra…	31	13.5 MB	56962 obs. of 31 v…

SVM

Description:

The Data was read into a Data Frame.

It was then split into train and test data with a 70:30 ratio.

Linear model was first built with all the features.

Then, varImp(linearMod) was used to find the important features.

```
summary(varImp(linearMod))
    Overall
 Min.   :  2.349
 1st Qu.:  4.343
 Median : 27.653
 Mean   : 55.167
 3rd Qu.: 84.344
 Max.   :225.511
```

The mean(~50) was used to qualify the feature as important. And only features with importance above 50 was selected.

Linear Regeression:

The model was built again with the selected features and predictions were made on the test set.

The prediction output being a continuous regression was converted to classification by selecting a split.

Initially 0.5 was used as the split(>=0.5 being Class 1 and <0.5 being Class 0)

```
> confusionMatrix(as.factor(Pred), as.factor(testData$Class))
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 56850    59
         1     8    45
```

In this case, although it had a high accuracy(99.88%), its specificity was low as the dataset had far more datapoints with Class 0 than Class 1.

So different splits were tried.

```
> Pred[Pred<0.2] = 0
Confusion Matrix and Statistics

          Reference
Prediction     0     1
```

```
          0 85290    27
          1    21   105

> Pred[Pred<0.1] = 0
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 85288    26
         1    23   106

> Pred[Pred<0.05] = 0
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 85267    25
         1    44   107
```

The balanced accuracy was the same in all these cases(~90%) but we have to choose between misclassifying less frauds as genuine and misclassifying less genuine transactions as frauds.

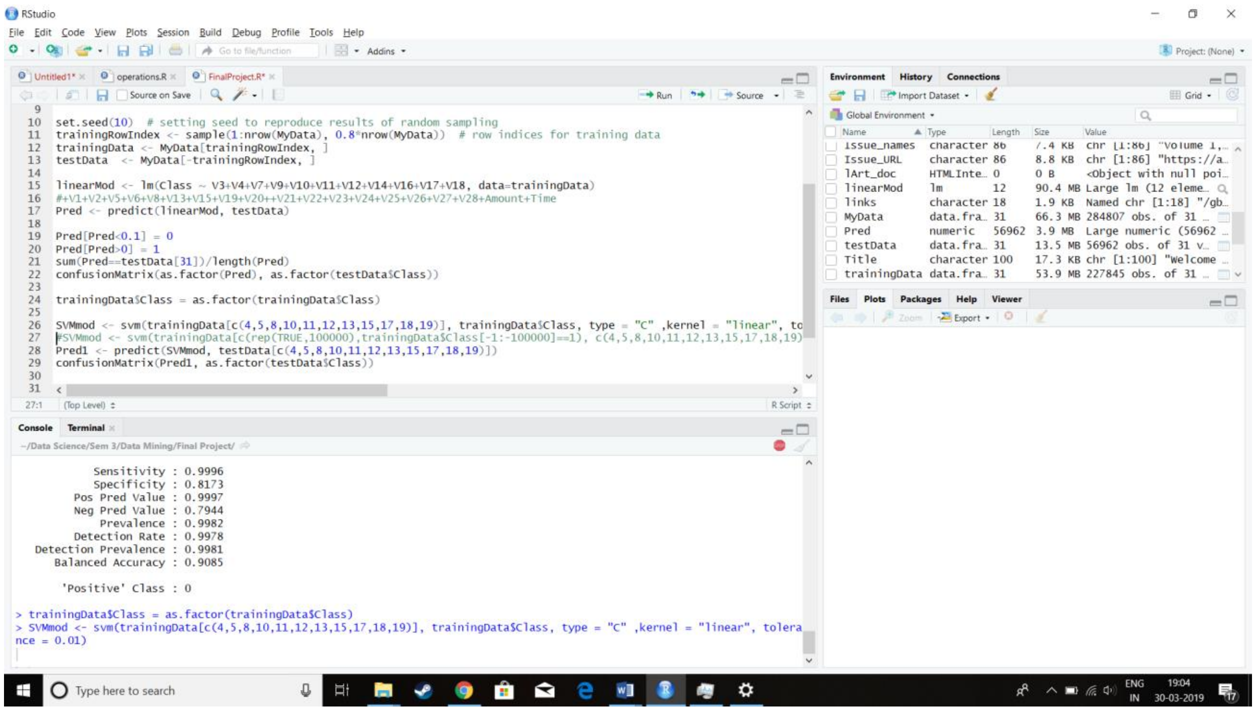
SVM:

Now in SVM we can set the model itself to give the output as classification instead of regression by giving either a factor as response variable or setting the type = “C”.

Although in case of SVM we have to decide the tolerance value which will decide how much it reduces the error but also exponentially increases the time taken to train the model.

First SVM was tried with 0.1 tolerance which trained the model for several minutes and yet during predictions, the balanced accuracy was 88%, slightly lesser than linear regression.

Next, 0.01 tolerance was tried which took a few hours.



```
set.seed(10)  # setting seed to reproduce results of random sampling
trainingRowIndex <- sample(1:nrow(MyData), 0.8*nrow(MyData))  # row indices for training data
trainingData <- MyData[trainingRowIndex, ]
testData  <- MyData[-trainingRowIndex, ]

linearMod <- lm(Class ~ V3+V4+V7+V9+V10+V11+V12+V14+V16+V17+V18, data=trainingData)
#+V1+V2+V5+V6+V8+V13+V15+V19+V20++V21+V22+V23+V24+V25+V26+V27+V28+Amount+Time
Pred <- predict(linearMod, testData)

Pred[Pred<0.1] = 0
Pred[Pred>0] = 1
sum(Pred==testData[31])/length(Pred)
confusionMatrix(as.factor(Pred), as.factor(testData$Class))

trainingData$Class = as.factor(trainingData$Class)

SVMmod <- svm(trainingData[c(4,5,8,10,11,12,13,15,17,18,19)], trainingData$Class, type = "C" ,kernel = "linear", to
#SVMmod <- svm(trainingData[c(rep(TRUE,100000),trainingData$Class[-1:-100000]==1), c(4,5,8,10,11,12,13,15,17,18,19)
Pred1 <- predict(SVMmod, testData[c(4,5,8,10,11,12,13,15,17,18,19)])
confusionMatrix(Pred1, as.factor(testData$Class))
```

Name	Type	Length	Size	Value
Issue_names	character	86	7.4 KB	chr [1:86] "Volume 1,…
Issue_URL	character	86	8.8 KB	chr [1:86] "https://a…
lArt_doc	HTMLInte…	0	0 B	<Object with null poi…
linearMod	lm	12	90.4 MB	Large lm (12 eleme…
links	character	18	1.9 KB	Named chr [1:18] "/gb…
MyData	data.fra…	31	66.3 MB	284807 obs. of 31 …
Pred	numeric	56962	3.9 MB	Large numeric (56962 …
testData	data.fra…	31	13.5 MB	56962 obs. of 31 v…
Title	character	100	17.3 KB	chr [1:100] "Welcome …
trainingData	data.fra…	31	53.9 MB	227845 obs. of 31 …

```
            Sensitivity : 0.9996
            Specificity : 0.8173
         Pos Pred Value : 0.9997
         Neg Pred Value : 0.7944
             Prevalence : 0.9982
         Detection Rate : 0.9978
   Detection Prevalence : 0.9981
      Balanced Accuracy : 0.9085

       'Positive' Class : 0

> trainingData$Class = as.factor(trainingData$Class)
> SVMmod <- svm(trainingData[c(4,5,8,10,11,12,13,15,17,18,19)], trainingData$Class, type = "C" ,kernel = "linear", tolera
nce = 0.01)
```

But the balanced accuracy improved to 90%.

```
Confusion Matrix and Statistics

          Reference
Prediction      0      1
         0 142128     57
         1     30    189

               Accuracy : 0.9994
                 95% CI : (0.9992, 0.9995)
    No Information Rate : 0.9983
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8126

 Mcnemar's Test P-Value : 0.005312

            Sensitivity : 0.9998
            Specificity : 0.7683
         Pos Pred Value : 0.9996
         Neg Pred Value : 0.8630
             Prevalence : 0.9983
         Detection Rate : 0.9981
   Detection Prevalence : 0.9985
      Balanced Accuracy : 0.8840
```

The confusion matrix tells that the model focuses on catching more fraud transactions at the cost of misclassifying genuine transactions.

Conclusion:

Lesser tolerances give better accuracies with longer run times. So choosing between Linear regression and SVM is simply choosing between efficiency and accuracy.

Source codes for the tools:

Linear regression:

lm(): https://rdrr.io/github/Fuzzy-Logix/AdapteR/src/R/FLLinRegr.R

```
#' @include utilities.R
#' @include data_prep.R
#' @include FLTable.R
NULL

## move to file datamining.R
#' An S4 class to represent FLLinRegr
#'
#' @slot offset this can be used to specify a priori known component to be included in
the
#' linear predictor during fitting. This should be NULL or a numeric vector of length
equal to the number of cases
#' An S4 class to represent objects returned by Data-Mining functions
#'
#' @slot deeptable A character vector containing a deeptable (either conversion from a
#' widetable or input deeptable)
#' @slot AnalysisID An output character ID from call to data-mining function
#' @slot wideToDeepAnalysisID An output character ID from FLRegrDataPrep
#' @slot mapTable A character string name for the mapping table in-database if input is
wide-table, generated by FLRegrDataPrep
#' @slot results cache list of results computed
#' @slot table Input data object
#' @export
setClass("FLDataMining",
                  slots=list(AnalysisID="character",
                                    wideToDeepAnalysisID="character",
                                    table="FLTable",
                                    results="list",
                                    deeptable="FLTable",
                                    mapTable="character"))

#' An S4 class to represent objects returned by Regression Functions
#'
#' @slot formula an object of class 'formula': Model Formula
#' @slot scoreTable Name of the in-database table where scoring results are stored
#' @export
setClass("FLRegr",
                  contains="FLDataMining",
                  slots=list(formula="formula",
                                               scoreTable="character",
                    RegrDataPrepSpecs="list"))

#' An S4 class to represent output from Linear Regression(lm) on in-database Objects
#'
#' @slot offset column name used as offset
#' @slot vfcalls information about system tables
#' @method print FLLinRegr
#' @method coefficients FLLinRegr
#' @method residuals FLLinRegr
#' @method influence FLLinRegr
#' @method lm.influence FLLinRegr
#' @method plot FLLinRegr
#' @method summary FLLinRegr
#' @method predict FLLinRegr
#' @export
setClass(
        "FLLinRegr",
        contains="FLRegr",
        slots=list(offset="character",
                                   vfcalls="character"))


#' @export
setClass(
    "FLLinRegrMD",
    contains="FLLinRegr")

#' @export
setClass(
        "FLLinRegrSF",
        contains="FLRegr",
        slots=list(offset="character",
                                   vfcalls="character"))


#' @export
setClass(
        "FLLogRegrSF",
        contains="FLRegr",
        slots=list(offset="character",
                    vfcalls="character"))
#' @export
setClass(
    "FLRobustRegr",
    contains="FLLinRegr")


#' Robust Regression.
#'
#' performs robust regression
#' @examples
#' Example for deeptbl:
#' library(MASS)
```

```r
#' options(debugSQL =TRUE)
#' table  <- FLTable(getTestTableName("tblRobustRegr"), "ObsID","VarID", "Num_Val")
#' flmod <- rlm(a~., data = table)
#' predict(flmod)
#' residuals(flmod)
#' flmod$fitted.values
#' summary(flmod)
#' @section Constraints:
#' plot method not supported
#' Example for widetable:
#' widetbl <- FLTable(getTestTableName("tblautompg"), "ObsID")
#' flmod <- rlm(Weight~ Acceleration , data = widetbl)
#' summary(flmod)
#' coefficients(flmod)
#' residuals(flmod)
#' @export
rlm <- function (formula,data=list(),psi, ...) {
         UseMethod("rlm", data)
}

## move to file rlm.R
#' @export
rlm.default <- MASS::rlm

## move to file rlm.R
#' @export
rlm.FLpreparedData <- function(formula,data,psi = "psi.huber", ...)
{
        vcallObject <- match.call()
        return(lmGeneric(formula=formula,
                         data=data,
                         callObject=vcallObject,
                         familytype="robust",
                         psi = psi,
                         ...))
}

## move to file rlm.R
#' @export
rlm.FLTable <- rlm.FLpreparedData

## move to file rlm.R
#' @export
rlm.FLTableMD <- rlm.FLpreparedData

#' @export
rlm.FLTableDeep <- rlm.FLpreparedData




## move to file lm.R
#' Linear Regression.
#'
#' \code{lm} performs linear regression on FLTable objects.
#'
#' The DB Lytix function called is FLLinRegr. Performs Linear Regression and
#' stores the results in predefined tables.
#'
#' @seealso \code{\link[stats]{lm}} for R reference implementation.
#' @param formula A symbolic description of model to be fitted
#' @param data An object of class FLTable or FLTableMD
#' @param catToDummy Transform categorical variables to numerical values
#' either using dummy variables or by using Empirical
#' Logit. If the value is 1, transformation is done using
#' dummy variables, else if the value is 0,
#' transformation is done using Empirical Logit.
#' @param performNorm 0/1 indicating whether to perform standardization of data.
#' @param performVarReduc 0/1. If the value is 1,
#' the stored procedure eliminates variables based on standard deviation and
#' correlation.
#' @param makeDataSparse If 0,Retains zeroes and NULL values
#' from the input table. If 1, Removes zeroes and NULL. If 2,Removes zeroes
#' but retains NULL values.
#' @param minStdDev Minimum acceptable standard deviation for
#' elimination of variables. Any variable that has a
#' standard deviation below this threshold is
#' eliminated. This parameter is only consequential if
#' the parameter PerformVarReduc = 1. Must be >0.
#' @param maxCorrel Maximum acceptable absolute correlation between
#' a pair of columns for eliminating variables. If the
#' absolute value of the correlation exceeds this
#' threshold, one of the columns is not transformed.
#' Again, this parameter is only consequential if the
#' parameter PerformVarReduc = 1. Must be >0 and <=1.
#' @param classSpec list describing the categorical dummy variables.
#' @param whereconditions takes the where_clause as a string.
#' @section Constraints:
#' The anova method is not yet available for FLLinRegr
#' If \code{data} is FLTableMD, only single formula is accepted.
#' So input deeptable or deeptable produced after data preparation
#' should have same VarIDs'.
#' For FLTableMD data object, only coefficients and summary
#' methods are defined.Predict method on \code{FLTableMD}
#' \code{newdata} is not supported.
#' Properties like \code{print(x),model,plot} might take time as they
#' have to fetch data
#' @return \code{lm} returns an object of class \code{FLLinRegr}
#' @examples
#' widetable  <- FLTable(getTestTableName("tblAbaloneWide"), "ObsID")
#' lmfit <- lm(Rings~Height+Diameter,widetable)
#' lmfit$coefficients
#' lmfit$fitted.values
#' plot(lmfit)
#' mu <- predict(lmfit,newdata=widetable)
#' deeptable <- FLTable(getTestTableName("myLinRegrSmall"),"ObsID","VarID","Num_Val")
#' lmfit <- lm(NULL,deeptable)
#' summary(lmfit)
#' flMDObject <- FLTableMD(table=getTestTableName("tblAutoMPGMD"),
#'                         group_id_colname="GroupID",
#'                         obs_id_colname="ObsID",group_id = c(2,4))
#' vformula <- MPG~HorsePower+Displacement+Weight+Acceleration
#' lmfit <- lm(vformula,
#'             data=flMDObject)
#' coeffList <- coef(lmfit)
```

```r
#' summaryList <- summary(lmfit)
#' @export
lm <- function (formula,data=list(),...) {
        UseMethod("lm", data)
}

## move to file lm.R
#' @export
lm.default <- stats::lm

## move to file lm.R
#' @export
lm.FLpreparedData <- function(formula,data,...)
{
        vcallObject <- match.call()
        return(lmGeneric(formula=formula,
                     data=data,
                     callObject=vcallObject,
                     ...))
}

## move to file lm.R
#' @export
lm.FLTable <- function(formula,data,...)
{
        vcallObject <- match.call()
        data <- setAlias(data,"")
        return(lmGeneric(formula=formula,
                     data=data,
                     callObject=vcallObject,
                     ...))
}

#' @export
lm.FLTableMD <- lm.FLTable

## move to file step.R
#' Choose a model.
#'
#' \code{steps} performs linear regression on FLTable objects.
#' Choose a formula based model by p-values and R-Squared Values.
#'
#' @seealso \code{\link[stats]{step}} for R reference implementation.
#'
#' @param object An object of class FLTable
#' @param scope A symbolic description of model to be fitted.
#' \code{scope} can be a list with upper and lower components
#' or a formula. For a widetable, upper and lower should be formulas
#' describing the range of models. If a formula is given instead of list
#' it will be treated as upper. For a deeptable, upper and lower should
#' be vectors with variable ids'.Provide empty list for deeptable if
#' nothing is to be specified.
#' @param scale currently not used.
#' @param direction character.Must be one of backward,
#' Fbackward,UFbackward,forward.
#' @param trace if positive, information is printed out during the
#' running of the steps.
#' @param catToDummy Transform categorical variables to numerical values
#' either using dummy variables or by using Empirical
#' Logit. If the value is 1, transformation is done using
#' dummy variables, else if the value is 0,
#' transformation is done using Empirical Logit.
#' @param performNorm 0/1 indicating whether to perform standardization of data.
#' @param performVarReduc 0/1. If the value is 1,
#' the stored procedure eliminates variables based on standard deviation and
#' correlation.
#' @param makeDataSparse If 0,Retains zeroes and NULL values
#' from the input table. If 1, Removes zeroes and NULL. If 2,Removes zeroes
#' but retains NULL values.
#' @param minStdDev Minimum acceptable standard deviation for
#' elimination of variables. Any variable that has a
#' standard deviation below this threshold is
#' eliminated. This parameter is only consequential if
#' the parameter PerformVarReduc = 1. Must be >0.
#' @param maxCorrel Maximum acceptable absolute correlation between
#' a pair of columns for eliminating variables. If the
#' absolute value of the correlation exceeds this
#' threshold, one of the columns is not transformed.
#' Again, this parameter is only consequential if the
#' parameter PerformVarReduc = 1. Must be >0 and <=1.
#' @param classSpec list describing the categorical dummy variables.
#' @param whereconditions takes the where_clause as a string.
#' @param highestpAllow1 All the variables whose p-value exceed the value
#' specified by HighestpAllow1 are dropped in one go.
#' Typical value for HighestProbAllow1 could be 0.50. Must be >0 and < 1.
#' Not applicable for forward.
#' @param highestpAllow2 Only one variable is dropped at a time
#' till all the p-Values are below the HighestpAllow2.
#' Typical value could be 0.10. Must be >0 and < 1.
#' Not applicable for forward and backward.
#' @param stepWiseDecrease The StepwiseDecrease is used to
#' decrease the p-Value at each stage. In first step,
#' all variables having pValue exceeding HighestpValue1 are
#' dropped. Then the HighestpValue1 is
#' reduced by StepwiseDecreasepValue
#' and the process is repeated until all
#' the variables have p-value less than HighestpValue2.
#' Must be >0 and <1. Used only for UFbackward.
#' @section Constraints:
#' The anova method is not yet available for FLLinRegr.
#' Properties like \code{print(fit$x),model,plot} might take time as they
#' have to fetch data
#'
#' @return \code{step} performs linear regression and replicates equivalent R output.
#' @examples
#' widetable  <- FLTable(getTestTableName("tblAbaloneWide"), "ObsID")
#' s <- step(widetable,
#'                          scope=list(lower=Rings~Height+Diameter),
#'                     direction = "UFbackward")
#' plot(s)
#' s$coefficients
#' s <- step(widetable,
#'                         scope=list(lower=Rings~Height+Diameter,
#'                                     upper=Rings~Height+Diameter+Sex+Num_Length),
#'                         direction = "UFbackward")
#' plot(s)
#' s$coefficients
#' s <- step(widetable,
#'                         scope=list(lower=Rings~Num_Length),
#'                         direction = "UFbackward",
#'                         performNorm=1,performVarReduc=1,maxCorrel=0.6)
#' plot(s)
#' s$coefficients
#' s <- step(widetable,
#'
        scope=list(upper=Rings~Height+Diameter+Sex+Num_Length+DummyCat),
#'                direction = "Fbackward")
#' plot(s)
#' s$coefficients
#' s <- step(widetable,
#'                         scope=Rings~Height+Diameter+Sex+Num_Length+DummyCat,
#'                direction = "forward")
#' plot(s)
#' s$coefficients
#' s <- step(widetable,
#'                         scope=Rings~Height+Diameter+Sex+Num_Length+DummyCat,
#'                direction = "Fbackward")
#' plot(s)
#' s$coefficients
#' s <- step(widetable,
#'
        scope=list(upper=Rings~Height+Diameter+Sex+Num_Length+DummyCat),
#'                direction = "forward")
#' plot(s)
#' s$coefficients
#' deeptable <- FLTable(getTestTableName("myLinRegrSmall"),"ObsID","VarID","Num_Val")
#' s <- step(deeptable,
#'                         scope=list(upper=c("-1","0","1")),
#'                direction = "backward")
#' s <- step(deeptable,
#'                         scope=list(upper=c("1","2"),lower=c("1")),
#'                direction = "Fbackward")
#' s <- step(deeptable,
#'                         scope=list(lower=c("2")),
#'                direction = "UFbackward")
#' s <- step(deeptable,
#'                         scope=list(),
#'                direction = "forward")
#' deeptable1 <- FLTable(getTestTableName("tblLogRegr"),
#'                                          "ObsID","VarID","Num_Val",
#'                    whereconditions=c("ObsID < 7001","VarID<5"))
#' s <- step(deeptable1,
#'          scope=list(lower=c("2")),
#'          direction = "UFbackward",familytype = "logistic")
#' s <- step(deeptable1,
#'                         scope=list(),
#'                direction = "forward",familytype="logistic")
#' plot(s)
#' s <- step(deeptable1,
#'                         scope=list(upper=c("-1","0","1","2","3")),
#'                direction = "backward",
#'                         familytype="multinomial",pRefLevel=1)
#' s <- step(deeptable1,
#'                         scope=list(upper=c("1","2","3"),lower=c("2")),
#'                 direction = "Fbackward",familytype="multinomial",pRefLevel=1)
#' deeptable2 <- FLTable(getTestTableName("tblLogRegrMN10000"),
#'                                           "ObsID","VarID","Num_Val",
#'                     whereconditions=c("ObsID < 7001","VarID<5"))
#' s <- step(deeptable2,
#'          scope=list(lower=c("2")),
#'          direction = "UFbackward",familytype = "multinomial",pRefLevel=1)
#' summary(s)
#' @export
step <- function (object,scope,...){
        UseMethod("step", object)
}

## move to file step.R
#' @export
step.default <- stats::step

## move to file step.R
#' @export
step.FLTable <- function(object, scope, scale = 0,
                                    direction = "forward",
                                    trace = 1,
                                    familytype="linear",
                                    ...){
        if (!direction %in% c("forward","Fbackward","backward","UFbackward","sf"))
        stop("direction must be in c(forward,Fbackward,backward,UFbackward)")
        if(!is.list(scope) && !class(scope)=="formula")
        stop("scope argument must be a list or formula.\n",
                " empty list accepted for deeptable.\n")
        if(!familytype %in%
c("linear","logistic","multinomial","linearSF","logisticSF"))
        stop("familytype argument must be one of linear,logistic or multinomial",
                "in step.FLTable\n")
        if(familytype=="multinomial" && direction=="forward")
        stop("forward not supported in multinomial logistic regr currently")
        vupperformula <- ""
        if(class(scope)=="formula")
        {
                if(isDotFormula(scope))
                        scope <- genDeepFormula(pColnames=setdiff(colnames(object),

getVariables(object)[["obs_id_colname"]]),

        pDepColumn=ifelse(isDeep(object),

                        NULL,

                        all.vars(scope)[1]))
                vupperformula <- scope
        }

        object <- setAlias(object,"")

        vinclude <- c()
        vexclude <- c()
        if(is.list(scope))
        {#browser()
                vlower <- scope[["lower"]]
```

```
                    vupper <- scope[["upper"]]

                    ## To account for . in formula
                    if(isDotFormula(vlower))
                    vlower <- genDeepFormula(pColnames=setdiff(colnames(object),

getVariables(object)[["obs_id_colname"]]),

         pDepColumn=ifelse(isDeep(object),

                          NULL,

                          all.vars(vlower)[1]))
                    if(isDotFormula(vupper))
                    vupper <- genDeepFormula(pColnames=setdiff(colnames(object),

getVariables(object)[["obs_id_colname"]]),

         pDepColumn=ifelse(isDeep(object),

                          NULL,

                          all.vars(vupper)[1]))

                    ##If only lower is given. Upper includes all.
                    if(is.null(vupper) && !is.null(vlower)){
                          if(!isDeep(object)){
                                  if(class(vlower)!="formula") stop("for wide table
scope should have formula as components\n")
                                  vupperformula <-
formula(paste0(all.vars(vlower)[1],"~",

         paste0(setdiff(colnames(object),

         c(all.vars(vlower)[1],

         getVariables(object)[["obs_id_colname"]])),

         collapse="+")))
                                  vinclude <-
all.vars(vlower)[2:length(all.vars(vlower))]
                          }
                          else{
                                  if(!is.vector(vlower)) stop("for deep table scope
should have vectors as components\n")
                                  vinclude <- vlower
                          }
                    }
                    else if(is.null(vlower) && !is.null(vupper)){
                          if(!isDeep(object)){
                                  if(class(vupper)!="formula") stop("for wide table
scope should have formula as components\n")
                                  vupperformula <- vupper
                          }
                          else{
                                  if(!is.vector(vupper)) stop("for deep table scope
should have vectors as components\n")
                                  vexclude <- setdiff(colnames(object),vupper)
```

```
                                }
                        }
                        else if(!is.null(vupper) && !is.null(vlower)){
                                if(!isDeep(object)){
                                        if(class(vupper)!="formula" ||
class(vlower)!="formula")
                                        stop("for wide table scope should have formula as
components\n")
                                        vupperformula <- vupper
                                        vinclude <-
all.vars(vlower)[2:length(all.vars(vlower))]
                                }
                                else{
                                        if(!is.vector(class(vupper)) ||
!is.vector(class(vlower)))
                                        stop("for deep table scope should have vectors as
components\n")
                                        vinclude <- vlower
                                        vexclude <- setdiff(colnames(object),vupper)
                                }
                        }
                        else if(!isDeep(object)) stop("scope cannot be empty list for
widetable")
        }

        vinclude <- setdiff(vinclude,c("-1"))
        vexclude <- setdiff(vexclude,c("0","-1"))
        if(!length(vinclude)>0) vinclude <- NULL
        if(!length(vexclude)>0) vexclude <- NULL

        if(!is.null(vinclude) || !is.null(vexclude))
        specID <- list(include=vinclude,
                                        exclude=vexclude)
        else specID <- list()

        vcallObject <- match.call()
        return(lmGeneric(formula=vupperformula,
                                        data=object,
                                        callObject=vcallObject,
                                        familytype=familytype,
                                        specID=specID,
                                        direction=direction,
                                        trace=trace,
                                        ...))
}

## move to file lmGeneric.R
lmGeneric <- function(formula,data,
                      callObject=NULL,
                      familytype="linear",
                      specID=list(),
                      direction="",
                      trace=1,
                      ...)
{
    if(inherits(data,"FLTable"))
        prepData <- prepareData(formula,data,
                                callObject=callObject,
                                familytype=familytype,
                                specID=specID,
                                direction=direction,
                                trace=trace,
                                ...)
    else if(inherits(data,"FLpreparedData")){
        prepData <- data
        data <- prepData$wideTable
    }
         for(i in names(prepData))
        assign(i,prepData[[i]])
    deepx <- setAlias(deepx,"")
    deeptable <- getTableNameSlot(deepx)
                                        #for more generic output:
    mod <- c(FLCoeffCorrelWithRes="CORRELWITHRES",
             FLCoeffNonZeroDensity="NONZERODENSITY",
             FLCoeffTStat="TSTAT",
             FLCoeffStdErr="STDERR",
             FLCoeffPValue="PVALUE",
             nCoeffEstim = "COEFFVALUE",
             nID = "COEFFID"
             )
    ## todo: create a list for this lookup
         if(familytype=="linear"){
                 if(direction=="sf") vfcalls<-c(functionName="FLLinRegrSF",

         infotableName="fzzlLinRegrInfo",
                                    note="SingleFactorLinRegr",
                                    coefftablename="fzzlLinRegrCoeffs",
                                    statstablename="fzzlLinRegrStats")
                 else
                 vfcalls <- c(functionName=ifelse(is.FLTableMD(data),

                                            "FLLinRegrMultiDataSet",

                                            "FLLinRegr"),

         infotableName="fzzlLinRegrInfo",

         Note="linregr",

         coefftablename="fzzlLinRegrCoeffs",

         statstablename="fzzlLinRegrStats",

         valcolnamescoretable="Y",

         scoretablename="FLLinRegrScore")}

         else if(familytype=="logistic"){
                 if(direction=="sf")  vfcalls<-c(functionName="FLLogRegrSF",

         infotableName="fzzlLogRegrInfo",
                                        note="SingleFactorLogRegr",
                                        coefftablename="fzzlLogRegrCoeffsSF",
                                        statstablename="fzzlLogRegrStatsSF")
                 else
                 vfcalls <- c(functionName=ifelse(is.FLTableMD(data),
         "FLLogRegrMultiDataSet",

         "FLLogRegr"),

         infotableName="fzzlLogRegrInfo",

         Note="logregr",

         coefftablename="fzzlLogRegrCoeffs",

         statstablename="fzzlLogRegrStats",

         valcolnamescoretable="Y",

         scoretablename="FLLogRegrScore")}

         else if(familytype=="poisson") vfcalls <- c(functionName="FLPoissonRegr",

         infotableName="fzzlPoissonRegrInfo",

         Note="poissonregr",

         coefftablename="fzzlPoissonRegrCoeffs",

         statstablename="fzzlPoissonRegrStats",

         valcolnamescoretable="Mu",

         scoretablename="FLPoissonRegrScore")
         else if(familytype=="logisticwt"){
                 vfcalls <- c(functionName="FLLogRegrWt",

         infotableName="fzzlLogRegrInfo",

         Note="logregrwt",

         coefftablename="fzzlLogRegrCoeffs",

         statstablename="fzzlLogRegrStats",

         valcolnamescoretable="Y",

         scoretablename="FLLogRegrScore")
                 # vtemp <- pThreshold
                 # pThreshold <- maxiter
                 # maxiter <- vtemp
         }
         else if(familytype=="multinomial") vfcalls <- c(functionName="FLLogRegrMN",

         infotableName="fzzlLogRegrMNInfo",

         Note="logregrMN",

         coefftablename="fzzlLogRegrMNCoeffs",

         statstablename="fzzlLogRegrMNStats",
          valcolnamescoretable="Y",
                                                scoretablename="FLLogRegrScore")

         else if(familytype == "robust") vfcalls <- c(functionName="FLRobustRegr",
                                             infotableName="fzzlRobustRegrInfo",
                                             Note="robustregr",

coefftablename="fzzlRobustRegrCoeffs",

statstablename="fzzlRobustRegrStats",
                                             cortablename =

"fzzlRobustRegrVarCov",

                                             scoretablename="FLLinRegrScore"
                                             )
         else if(familytype == "pls") vfcalls <- c(functionName="FLPLSRegr",
                                          infotableName="fzzlPLSRegrInfo",
                                          note="plsregr",
                                          coefftablename="fzzlPLSRegrCoeffs",
                                          statstablename="fzzlPLSRegrConvVec",
                                          scoretablename="FLLinRegrScore"
                                          )
         else if(familytype == "opls") vfcalls <- c(functionName="FLOPLSRegr",
                                          infotableName="fzzlPLSRegrnfo",
                                          Note="oplsregr",
                                          coefftablename="fzzlPLSRegrCentCoeffs",
                                          statstablename="fzzlOPLSRegrConvVec",
                                          scoretablename="FLLinRegrScore",
                                          rcoeff = "fzzlOPLSRegrFactorFit"
                                          )

          functionName <- vfcalls["functionName"]
          infotableName <- vfcalls["infotableName"]
          vnote <- genNote(vfcalls["note"])
          coefftablename <- vfcalls["coefftablename"]
          statstablename <- vfcalls["statstablename"]

          vinputCols <- list()
          if(functionName %in% c("FLLinRegrMultiDataSet",
                                                "FLLogRegrMultiDataSet"))
                   vinputCols <- c(vinputCols,
                                             TableName=deeptable,

          GroupIDCol=getGroupIdSQLExpression(deepx),

          ObsIDCol=getObsIdSQLExpression(deepx),

          VarIDCol=getVarIdSQLExpression(deepx),

          ValueCol=getValueSQLExpression(deepx)
                                             )
          else vinputCols <- c(vinputCols,
                                             TableName=deeptable,

          ObsIDCol=getObsIdSQLExpression(deepx),
                         VarIDCol=getVarIdSQLExpression(deepx),
                         ValueCol=getValueSQLExpression(deepx)
                                             )
```

```
        if(familytype %in% c("multinomial"))
        vinputCols <- c(vinputCols,
                                    pRefLevel=pThreshold)

        if(!familytype %in% c("linear", "robust", "pls", "opls") &&
direction!="forward")
        vinputCols <- c(vinputCols,
                                    MaxIterations=maxiter)
        if(base::grepl("logistic",familytype)
                && direction!="forward")
        vinputCols <- c(vinputCols,
                                    pThreshold=pThreshold)
        if(direction==""){
                vfuncName=functionName
                if(familytype %in% "logisticwt"){
          vinputCols <- as.list(vinputCols)
          vinputCols[["MaxIterations"]] <- NULL
          vinputCols <- unlist(vinputCols)
          vinputCols <- c(vinputCols,
                          MaxIterations=maxiter,
                          EVENTWEIGHT=eventweight,
                          NONEVENTWEIGHT=noneventweight)
        }
         }
        if(direction %in% c("backward","Fbackward","UFbackward")){
                vfuncName <- paste0(functionName,"BW")
                vinputCols <- c(vinputCols,
                                            SPECID=vspecID,
                                            HIGHESTPALLOW1=highestpAllow1)
         }
        if(direction %in% c("Fbackward","UFbackward")){
                vfuncName <- paste0(functionName,"FB")
                vinputCols <- c(vinputCols,
                                            HIGHESTPALLOW2=highestpAllow2)
         }
        if(direction %in% c("UFbackward")){
                vfuncName <- paste0(functionName,"UFB")
                vinputCols <- c(vinputCols,
                                            STEPWISEDECREASE=stepWiseDecrease)
         }
        if(direction %in% c("forward")){
                vfuncName <- paste0(functionName,"SW")
                if(!familytype %in% "linear")
                vinputCols <- c(vinputCols,
                                            pThreshold=pThreshold)
                if(familytype %in% "logistic")
                vinputCols <- c(vinputCols,
                                    MaxIterations=maxiter)
                vinputCols <- c(vinputCols,
                                            TOPN=topN,
                                            HIGHESTPALLOW1=highestpAllow1)
         }
      if(direction %in% "sf"){
          vfuncName <- paste0(functionName)
          vinputCols <- c(vinputCols,
                          MaxIterations=maxiter,
                          pThreshold=pThreshold)
      }
```

```
    ##for rlm defining psi and tuning constant:
    if(familytype %in% "robust")
    {

        weightfn = "huber"
        if(list(...)$psi == "psi.bisquare" )
        {weightfn <- "bisquare"}
        else if(list(...)$psi == "psi.hampel")
            print("dont compute rlm for hampel function currently computing it for
huber")
        else if(list(...)$psi %in% c("cauchy", "fair","logistic", "talwar", "andrews",
"welsch")
                )
            weightfn <- list(...)$psi
        if(is.null(list(...)$u))
        {
            tunconst <- .5
        }
        else
            tunconst <- list(...)$u

        vinputCols <- c(vinputCols,
                        WeightFn = weightfn,
                        TuneConstant= tunconst,
                        MaxIterations =maxiter
                        )
        functionName <- "FLRobustRegr"
        mod <- c(FLCoeffCorrelWithRes="",
                 FLCoeffNonZeroDensity="",
                 FLCoeffTStat="T_VAL",
                 FLCoeffStdErr="STDDEV",
                 FLCoeffPValue="P_VAL",
                 nCoeffEstim = "EST",
                 nID = "VARID"
                 )  }

    if(familytype %in% "pls")
    {
        functionName <- "FLLinRegr"
        if(!list(...)$nfactor )
        {print(list(...)$nfactor)
            nfactor <- 15}
        vinputCols <- c(vinputCols,
                        NumOfFactors = list(...)$nfactor)
        mod <- c(mod, ncomp = list(...)$nfactor)
    }


    if(familytype %in% "opls")
    {
        functionName <- "FLLinRegr"
        if(!list(...)$nfactor)
        {print("Number of Component is missing insterting default value of 4 ")
            nfactor <- 4
        }
        if(!list(...)$Northo)
        {
            print("Number of Ortho is missing insterting default value of 3")
            northo <- 3
        }

        vinputCols <- c(vinputCols,
                        NumOfFactors = list(...)$nfactor,
                        NumOfOrtho = list(...)$Northo)
        mod <- c(mod, ncomp = list(...)$nfactor, northo = list(...)$Northo)
    }

    vinputCols <- c(vinputCols,
                    Note=vnote)

    retobj <- sqlStoredProc(getFLConnection(),
                            vfuncName,
                            outputParameter=c(AnalysisID="a"),
                            pInputParams=vinputCols
                            )

    retobj <- checkSqlQueryOutput(retobj)
    AnalysisID <- as.character(retobj[1,1])
    ##Find the max modelID to avoid joins later.
    ##For forward find best fit model id.
    vmaxModelID <- NULL
    vmaxLevelID <- NULL
    if(direction=="" && familytype!="poisson"
       &&!is.FLTableMD(data)){
        vmaxModelID <- 1
        vmaxLevelID <- 1
    }
    else if(!direction %in% c("forward","sf") && familytype!="poisson" &&
!is.FLTableMD(data)){
        vsqlstr <- paste0("SELECT MAX(ModelID) AS modelid",
                          ifelse(familytype=="multinomial",",MAX(LevelID) AS levelid
",""),
                          " FROM ",coefftablename," WHERE
AnalysisID=",fquote(AnalysisID))
        vtemp <- sqlQuery(getFLConnection(),vsqlstr)
        vmaxModelID <- vtemp[["modelid"]]
        vmaxLevelID <- vtemp[["levelid"]]
    }

    if(trace>0 && !direction %in% c("","forward","sf"))
    {
        # vsqlstr <- paste0("SELECT a.coeffid,c.* \n",
        #                   " FROM ",coefftablename," a,",statstablename," c \n",
        #                   " WHERE NOT EXISTS(SELECT 1 FROM ",coefftablename," b ",
        #                   " WHERE b.analysisid=a.analysisid AND b.modelid=a.modelid+1
\n",
        #                   " AND a.coeffid = b.coeffid ",ifelse(!is.null(vmaxLevelID),
        #                                                        " AND a.LevelID =
b.LevelID ",""),")\n",
        #                   " AND a.analysisid=",fquote(AnalysisID)," AND
c.analysisid=a.analysisid \n",
        #                   " AND a.modelid<>",vmaxModelID," AND c.modelid=a.modelid\n",
        #                   ifelse(!is.null(vmaxLevelID),paste0(" AND a.LevelID =
",vmaxLevelID),""),
        #                   " \n UNION ALL\n",
        #                    " SELECT 0,a.* FROM ",statstablename," a \n",
        #                    " WHERE a.AnalysisID=",fquote(AnalysisID),
        #                    " AND a.ModelID=",vmaxModelID,"\n",
        #                    " ORDER BY 3")
        vsqlstr <- paste0("SELECT a.coeffid,c.* \n",
                          " FROM (SELECT DISTINCT AnalysisID,modelid,coeffid from
",coefftablename,
                                " WHERE analysisid=",fquote(AnalysisID)," \n EXCEPT \n ",
                                " SELECT DISTINCT AnalysisID,modelid+1,coeffid from
",coefftablename,
                                " WHERE analysisid=",fquote(AnalysisID)," \n ",
                                ") a,",statstablename," c \n",
                          " WHERE c.analysisid=a.analysisid \n"
                          )
        d <- sqlQuery(getFLConnection(),vsqlstr)
        colnames(d)<-toupper(colnames(d))
        d[["ANALYSISID"]] <- NULL
        vdroppedCols <- c()
        if(!isDeep(data))vdroppedCols <- specID[["exclude"]]
        if(nrow(d)>1){
            for(i in unique(setdiff(d[["MODELID"]],vmaxModelID)))
            {
                if(familytype=="linear")
                    cat("Step:    RSQUARED = ",d[d[,"MODELID"]==i,"RSQUARED"][1],"\n")
                else if(familytype=="logistic")
                    cat("Step:    Gini Coefficient =
",d[d[,"MODELID"]==i,"GINICOEFF"][1],"\n")
                                        #browser()
                vdropped <- as.numeric(d[d[,"MODELID"]==i,"COEFFID"])
                vcolnames <- names(vmapping)
                vdroppedCols1 <- sapply(vdropped,function(x)
vcolnames[as.numeric(vmapping)==x])
                if(!isDeep(data))vdroppedCols <- c(vdroppedCols1,vdroppedCols)
                if(isDeep(data)){
                    vallVars <- all.vars(formula)
                    vfr <- genDeepFormula(c(vdropped))
                    vdroppedCols <- c(vdroppedCols,all.vars(vfr)[-1])
                    vdroppedCols1 <- all.vars(genDeepFormula(specID[["exclude"]]))[-1]
                    vcolnames <- vallVars[!vallVars %in% vdroppedCols1]
                }
                cat(vallVars[1],"~",paste0(vcolnames[!toupper(vcolnames) %in%
c(toupper(vdroppedCols)

,toupper(vallVars[1]))],
                                          collapse=" + "),"\n")
                vdataframe <- rbind(d[d[,"MODELID"]==i,][1,],d[d[,"MODELID"]==i+1,][1,])
                rownames(vdataframe) <- c(" - None",paste0(" -
",paste0(vdroppedCols,collapse=" + ")))
                print(vdataframe[,!colnames(vdataframe) %in%
c("COEFFID","BPSTAT","SIGBPSTAT")])
                cat("\n\n\n")
            }
        }
    }
    else if(direction %in% c("forward"))
    {
        if(familytype=="linear")
            vsqlstr <- limitRowsSQL(paste0("SELECT a.*,b.maxPValue \n",
                              " FROM ",statstablename," a,( \n",
                              " SELECT a.ModelID,",
                              " MAX(a.PValue) AS maxPValue \n",
                              " FROM ",coefftablename," a \n",
                              " WHERE a.AnalysisID = ",fquote(AnalysisID),
                              " GROUP BY a.ModelID) AS b \n",
                              " WHERE b.ModelID = a.ModelID \n",
                              " AND a.AnalysisID = ",fquote(AnalysisID),
                              " AND b.MaxPValue < 0.10 \n",
                              " ORDER BY 3 DESC, 2 \n"),1)
        else if(familytype=="logistic")
            vsqlstr <- limitRowsSQL(paste0("SELECT a.*\n",
                              " FROM ",statstablename," a\n",
                              " WHERE a.AnalysisID = ",fquote(AnalysisID),
                              " AND a.HighestPValue < 0.10 \n",
                              " ORDER BY 3 DESC, 2 \n"),1)

        d <- sqlQuery(getFLConnection(),vsqlstr)
        colnames(d) <- toupper(colnames(d))
        d[["ANALYSISID"]] <- NULL
        vmaxModelID <- d[["MODELID"]]
        if(trace>0) print(d)
    }

    vfuncName <- ifelse(familytype %in% c("logisticwt","poisson"),
                        "FLLogRegr",functionName)
    vfuncName <- base::gsub("MultiDataSet","MD",vfuncName)
    vfuncName <- ifelse(familytype %in% c("pls", "opls"), "FLPLSRegr", vfuncName)

    return(new(vfuncName,
                                   formula=formula,
                                   AnalysisID=AnalysisID,
                                   wideToDeepAnalysisID=wideToDeepAnalysisID,
                                   table=data,
                                   results=list(call=callObject,
                          modelID=vmaxModelID,
                          mod = mod),
                                   deeptable=deepx,
                                   mapTable=mapTable,
                                   scoreTable="",
                                   vfcalls=vfcalls,
                                   offset=as.character(offset),
                      RegrDataPrepSpecs=RegrDataPrepSpecs))
}

#' @export
prepareData <- function(formula,...) UseMethod("prepareData")

#' @export
prepareData.FLpreparedData <- function(formula, data, fetchIDs=FALSE, outDeepTable="",
...) {
    template <- formula
    dataCopy <- data
    vRegrDataPrepSpecs <- setDefaultsRegrDataPrepSpecs(x=template$RegrDataPrepSpecs,
                                                       values=list(...))
    deepx <- FLRegrDataPrep(data,depCol=vRegrDataPrepSpecs$depCol,
                            OutDeepTable=outDeepTable,
                            OutObsIDCol=vRegrDataPrepSpecs$outObsIDCol,
                            OutVarIDCol=vRegrDataPrepSpecs$outVarIDCol,
                            OutValueCol=vRegrDataPrepSpecs$outValueCol,
                            CatToDummy=vRegrDataPrepSpecs$catToDummy,
                            PerformNorm=vRegrDataPrepSpecs$performNorm,
                            PerformVarReduc=vRegrDataPrepSpecs$performVarReduc,
                            MakeDataSparse=vRegrDataPrepSpecs$makeDataSparse,
                            MinStdDev=vRegrDataPrepSpecs$minStdDev,
                            MaxCorrel=vRegrDataPrepSpecs$maxCorrel,
                            TrainOrTest=1,
                            ExcludeCols=vRegrDataPrepSpecs$excludeCols,
                            ClassSpec=vRegrDataPrepSpecs$classSpec,
                            WhereClause=vRegrDataPrepSpecs$whereconditions,
                            InAnalysisID=template$wideToDeepAnalysisID,
                            fetchIDs=fetchIDs)
    data <- deepx
    data <- setAlias(data,"")
    data
}


## gk: todo: make this obsolete and refactor to use prepareData.FLpreparedData
#' @export
prepareData.FLRegr <- function(formula, data, outDeepTableName="",
                               fetchIDs=FALSE, ...) {
    if(isDeep(data)) return(data)
    dataCopy <- data
    vRegrDataPrepSpecs <- setDefaultsRegrDataPrepSpecs(x=formula@RegrDataPrepSpecs,
                                                       values=list(...))
    vdepCol <- formula@RegrDataPrepSpecs$depCol
    if(is.null(vdepCol))
         vdepCol <- "NULL"
    deepx <- FLRegrDataPrep(data,depCol=vdepCol,
                            OutDeepTable=outDeepTableName,
                            OutObsIDCol=vRegrDataPrepSpecs$outObsIDCol,
                            OutVarIDCol=vRegrDataPrepSpecs$outVarIDCol,
                            OutValueCol=vRegrDataPrepSpecs$outValueCol,
                            CatToDummy=vRegrDataPrepSpecs$catToDummy,
                            PerformNorm=vRegrDataPrepSpecs$performNorm,
                            PerformVarReduc=vRegrDataPrepSpecs$performVarReduc,
                            MakeDataSparse=vRegrDataPrepSpecs$makeDataSparse,
                            MinStdDev=vRegrDataPrepSpecs$minStdDev,
                            MaxCorrel=vRegrDataPrepSpecs$maxCorrel,
                            TrainOrTest=1,
                            ExcludeCols=vRegrDataPrepSpecs$excludeCols,
                            ClassSpec=vRegrDataPrepSpecs$classSpec,
                            WhereClause=vRegrDataPrepSpecs$whereconditions,
                            InAnalysisID=formula@wideToDeepAnalysisID,
                            fetchIDs=fetchIDs)
    data <- deepx
    data <- setAlias(data,"")

    if(formula@vfcalls["functionName"]=="FLPoissonRegr"){
        ## Insert dependent and offset varids in deeptable
        vVaridCols <- c(-2)
        vcellValCols <- ifelse(formula@offset!="",formula@offset,0)

        ## if(!all.vars(formula@formula)[1] %in% colnames(dataCopy))
                ##      stop("dependent column ",all.vars(formula@formula)[1],
        ##                " not in data \n ")
        vVaridCols <- c(vVaridCols,-1)
        vcellValCols <- c(vcellValCols,all.vars(formula@formula)[1])

                                        # vtablename <- getTableNameSlot(dataCopy)
        vtablename <- getTableNameSlot(table)
        vtablename1 <- getTableNameSlot(data)

        vobsid <- getObsIdSQLExpression(formula@table)
        sqlstr <- paste0(" SELECT ",vobsid," AS obs_id_colname, \n ",
                         vVaridCols," AS var_id_colname, \n ",
                         vcellValCols," AS cell_val_colname \n  ",
                         " FROM ",vtablename,collapse=" UNION ALL ")
        t <- insertIntotbl(pTableName=vtablename1,
                           pSelect=sqlstr)
        data@Dimnames[[2]] <- c("-1","-2",data@Dimnames[[2]])
    }
    data
}


## move to file lmGeneric.R
#' @export
prepareData.formula <- function(formula,data,
                                                                       callObject=NULL,

         familytype="linear",
                                                                       specID=list(),
                                                                       direction="",
                                                                       trace=1,
                                                                       catToDummy=0,
                                                                       performNorm=0,
                                                                       performVarReduc=0,
                                                                       makeDataSparse=1,
                                                                       minStdDev=0,
                                                                       maxCorrel=1,
                                                                       classSpec=list(),
                                                                       whereconditions="",
                                                                       highestpAllow1=0.5,
                                                                       highestpAllow2=0.1,

         stepWiseDecrease=0.05,
                                                                       topN=1,
                                                                       pThreshold=0.1,
                                                                       eventweight=0.8,
                                                                       noneventweight=1,
                                                                       maxiter=25,
                                                                       offset="",
                                                                       pRefLevel=NULL,
                               fetchIDs=FALSE,
                               outDeepTableName="",
                               ...){
    data <- setAlias(data,"")
        if(isDeep(data)){
                vallVars <- colnames(data)
                ##For MultiDataset and deep data
                ##colnames is a list
                ##Assumption: All Models have same formula
         ## Meaning same varIDs
         if(is.FLTableMD(data)){
             if(!length(unique(vallVars))==1)
                  # stop("Datasets should have same columns \n ")
                  vallVars <- colnames(data)[[1]]
             else vallVars <- vallVars[[1]][1]:vallVars[[1]][2]
         }
         formula <- genDeepFormula(vallVars)
    }
    else{
    if(isDotFormula(formula)){
      vexcludeCols <- NULL
      if("excludeCols" %in% names(list(...)))
        vexcludeCols <- list(...)$excludeCols
      if("ExcludeCols" %in% names(list(...)))
        vexcludeCols <- list(...)$ExcludeCols
      formula <- genDeepFormula(pColnames=setdiff(colnames(data),
                        c(vexcludeCols,
                        getObsIdSQLExpression(data))),
                  pDepColumn=all.vars(formula)[1])
    }
         vallVars <- base::all.vars(formula)
         vdependent <- vallVars[1]
         if(is.null(vdependent))
             vdependent <- "NULL"
         vindependent <- vallVars[2:length(vallVars)]
         checkValidFormula(formula,data)
    }

    vcolnames <- colnames(data)
    wideToDeepAnalysisID <- ""
  mapTable <- ""

  if(offset!="" && !toupper(offset) %in% toupper(vcolnames))
  stop("offset not in colnames of data")

  check0To1 <- function(pObject)
  {
    if(!is.numeric(pObject) ||
         pObject < 0 ||
         pObject > 1)
    stop(names(pObject)," should be >0 and <1\n")
  }
  checkSpecID <- function(pObject,pAllVars)
  {
    pObject <- c(pObject[["include"]],pObject[["exclude"]])
    if(length(pObject)>0)
    {
         sapply(pObject,function(x)
        if(!(x %in% pAllVars))
        stop(paste0(x,collapse=","),"  specified in SpecID not in colnames of
data\n"))
    }
  }

  maxiter <- maxiter[1]
  if(!is.numeric(maxiter) || maxiter<=0)
  stop("maxiter should be >0")
```

```
    maxiter <- as.integer(maxiter)

    if(familytype %in% c("logistic","logisticwt"))
    {
        check0To1(c(pThreshold=pThreshold))
        if(familytype %in% "logisticwt"){
                check0To1(c(eventweight=eventweight))
                check0To1(c(noneventweight=noneventweight))
        }
    }

    if(is.FLTableMD(data)){
        if(!familytype %in% c("logistic","linear"))
                stop("only lm and glm with binomial family supported for
MultiDataSet\n")
        direction <- ""
    }
    if(direction=="UFbackward")
    {
        check0To1(c(highestpAllow1=highestpAllow1,
                                highestpAllow2=highestpAllow2,
                                stepWiseDecrease=stepWiseDecrease))
        checkSpecID(specID,vallVars)
    }
    if(direction=="backward")
    {
        check0To1(c(highestpAllow1=highestpAllow1))
        checkSpecID(specID,vallVars)
    }
    if(direction=="forward")
    {
        check0To1(c(highestpAllow1=highestpAllow1))
        if(!is.numeric(topN) || as.integer(topN)<1 || as.integer(topN)>10)
        stop("topN should be >0 and <=10")
        topN <- as.integer(topN)
    }
    if(direction=="Fbackward")
    {
        check0To1(c(highestpAllow1=highestpAllow1,
                                highestpAllow2=highestpAllow2))
        checkSpecID(specID,vallVars)
    }

    if(!isDeep(data)){
       ##browser()
        unused_cols <- setdiff(vcolnames,c(all.vars(formula),specID[["exclude"]]))
       unused_cols <- setdiff(unused_cols,
                              c(getGroupIdSQLExpression(data),
                               getObsIdSQLExpression(data)))
       ## Detect factors and assign classSpec
       vfirstRow <- sqlQuery(getFLConnection(),
                            limitRowsSQL(paste0("SELECT * FROM (",
                                         constructSelect(data),") a "),1))
       vtblInfo <- separateDBName(getTableNameSlot(data))
       vColInfo <- c()
       if(is.TD())
       vColInfo <- sqlQuery(getFLConnection(),
                          paste0("SELECT columnName FROM dbc.columns WHERE \n ",
                                                  "columnType = 'CV' AND databaseName= ",
                                                  fquote(vtblInfo["vdatabase"])," \n ",
                                                  " AND tableName =
",fquote(vtblInfo["vtableName"])))[[1]]
        vfactorCols <- list()
                  ## apply(t,2,function(x){class(x[[1]])}) gives all character
                  for(i in setdiff(colnames(vfirstRow),
                                                                   c(unused_cols,names(classSpec),

         getGroupIdSQLExpression(data),
                                          getObsIdSQLExpression(data),
                                                                            "obs_id_colname",

         getGroupIdSQLExpression(data),
                                                                            "group_id_colname",
                                          list(...)[["doNotTransform"]]))){
             ##browser()
                                        if(length(i)==0) break;
                                        if(is.factor(vfirstRow[[i]])
                                                 || is.character(vfirstRow[[i]])
                                                 || is.logical(vfirstRow[[i]])
               || (i %in% sub("\\s+$", "", vColInfo))){ ## remove trailing spaces
                                                 # if(is.logical(vfirstRow[[i]])){
                                                 #        vtemp <- levels(sqlQuery(getFLConnection(),
                                                 #
         paste0("SELECT DISTINCT(",i,
                                                 #
         ") FROM(",constructSelect(data),") a "))[[1]])[1]
                                                 #        names(vtemp) <- i
                                                 #        classSpec <- c(classSpec,vtemp)
                                                 # }
                                                 # else{
                                                         r<-as.character(vfirstRow[[i]])
                                                         names(r) <- i
                                                         vfactorCols <- c(vfactorCols,r)
                                                 # }
                                        }
                  }
                  if(length(vfactorCols)>0){
                                        if(is.ODBC())
               vrefVars <- sqlQuery(getFLConnection(),
                                          paste0("SELECT ",
                                              paste0("MIN(",names(vfactorCols),
                                                  ") AS ",names(vfactorCols),
                                                  collapse=","),
                                              " FROM (",constructSelect(data),") a "),
                                          as.is=TRUE)
           else vrefVars <- sqlQuery(getFLConnection(),
                                          paste0("SELECT ",
                                              paste0("MIN(",names(vfactorCols),
                                                  ") AS ",names(vfactorCols),
                                                  collapse=","),
                                              " FROM (",constructSelect(data),") a "))
                                        vtempList <- list()
           vrefVarNames <- names(vrefVars)
                                        for(i in colnames(vrefVars)){
               ## Remove variables with NA
               if(is.na(vrefVars[[i]]))
```

```
                                vrefVarNames <- setdiff(vrefVarNames,
                                                        i)
                            else if(is.logical(vrefVars[[i]]))
                                                        vtempList <- c(vtempList,

                     levels(as.factor(sqlQuery(getFLConnection(),

                             paste0("SELECT DISTINCT(",i,

                             ") FROM(",constructSelect(data),") a "))[[1]]))[1])
                                              else vtempList <-
            c(vtempList,as.character(vrefVars[[i]]))
                                      }
                                      names(vtempList) <- vrefVarNames
                                      # vfactorCols[names(vrefVars)] <-
            as.list(apply(vrefVars[names(vrefVars)],
                                      #
                                              2,function(x){
                                      #
                                                      browser()
                                      #
                                                      if(is.logical(x))
                                      #

                     return(levels(sqlQuery(getFLConnection(),
                                      #

                     paste0("SELECT DISTINCT(",names(x),
                                      #

                     ") FROM(",constructSelect(data),") a "))[[1]])[1])
                                      #
                                                      else as.character(x)
                                      #
                                              }))
                                      classSpec <- c(classSpec,vtempList)
                             }

                             # vexcludeCols <- paste0(unused_cols,collapse=",")
                    vexcludeCols <- setdiff(unused_cols,
                                            getObsIdSQLExpression(data))
                }

                     vcallObject <- callObject
                vRegrDataPrepSpecs <- list()
                     if(!isDeep(data))
                     {

                    deepx <- FLRegrDataPrep(data,depCol=vdependent,
                                            OutDeepTable=outDeepTableName,
                                            OutObsIDCol="obsid",
                                            OutVarIDCol="varid",
                                            OutValueCol="numval",
                                            CatToDummy=catToDummy,
                                            PerformNorm=performNorm,
                                            PerformVarReduc=performVarReduc,
                                            MakeDataSparse=makeDataSparse,
                                            MinStdDev=minStdDev,
                                            MaxCorrel=maxCorrel,
                                            TrainOrTest=0,
                                            ExcludeCols=vexcludeCols,
                                            ClassSpec=classSpec,
                                            WhereClause=whereconditions,
                                            InAnalysisID="",
                                            fetchIDs=fetchIDs)
                        vRegrDataPrepSpecs <- list(
                                            outObsIDCol="obsid",
                                            outVarIDCol="varid",
                                            outValueCol="numval",
                                            catToDummy=catToDummy,
                                            performNorm=performNorm,
                                            performVarReduc=performVarReduc,
                                            makeDataSparse=makeDataSparse,
                                            minStdDev=minStdDev,
                                            maxCorrel=maxCorrel,
                                            trainOrTest=0,
                                            excludeCols=vexcludeCols,
                                            classSpec=classSpec)

                            wideToDeepAnalysisID <- deepx@wideToDeepAnalysisID
                            deepx <- setAlias(deepx,"")
                            whereconditions <- ""
                            mapTable <-
            getRemoteTableName(tableName=getSystemTableMapping("fzzlRegrDataPrepMap"),
                                                    temporaryTable=FALSE)
                            if(familytype=="poisson")
                            {
                                    vtablename <- getTableNameSlot(deepx)
                                    vtablename1 <- getTableNameSlot(data)
                                    vobsid <- getObsIdSQLExpression(data)
                                    sqlstr <- paste0(" SELECT ",vobsid," AS obs_id_colname,","\n
            ",
                                                            " -2 AS var_id_colname,","\n
            ",

                     ifelse(offset!="",offset,0)," AS cell_val_colname","\n        ",
                                                            " FROM ",vtablename1)
                                    t <- insertIntotbl(pTableName=vtablename,
                                            pSelect=sqlstr)
                                    deepx@Dimnames[[2]] <- c("-2",deepx@Dimnames[[2]])
                            }

                            ##Get Mapping Information for specID
                            if(!is.FLTableMD(data)){
                                    vmapping <- sqlQuery(getFLConnection(),
                                                            paste0("SELECT a.Column_name
            AS colname,\n",
                                                                     "
            a.Final_VarID AS varid\n",
                                                                  " FROM
            ",mapTable," AS a\n",
                                                                  " WHERE
            a.AnalysisID = ",fquote(wideToDeepAnalysisID),
                                                                  " AND
            a.Final_VarID IS NOT NULL \n",
                                                                                            " ORDER BY
            a.Final_VarID\n"))

                                      vtemp <- vmapping[["varid"]]
                                      names(vtemp) <- vmapping[["colname"]]
                                      vmapping <- vtemp
                                      vallVars <- setdiff(vallVars,specID[["exclude"]])
                              }
                              else vmapping <- ""
                      }
                      else if(class(data@select)=="FLTableFunctionQuery")
                      {
                              #sqlstr <- paste0("CREATE VIEW ",getOption("ResultDatabaseFL"),
                              #                                          ".",deeptablename," AS
            ",constructSelect(data))
                              #sqlSendUpdate(connection,sqlstr)
                              deeptablename <- createView(pViewName=gen_view_name(""),
                                                pSelect=constructSelect(data))

                              #sqlstr <- paste0("CREATE VIEW
            ",getOption("ResultDatabaseFL"),".",deeptablename1,
                              #                                 " AS SELECT * FROM
            ",getOption("ResultDatabaseFL"),".",deeptablename,
                              #                                 constructWhere(whereconditions))
                              #t <- sqlSendUpdate(connection,sqlstr)
                              deeptablename1<-createView(pViewName=gen_view_name("New"),
                                                        pSelect=paste0("SELECT * FROM
            ",deeptablename,

                      constructWhere(whereconditions)))


                              deepx <- FLTable(deeptablename1,
                                     getObsIdSQLExpression(data),
                                     getVarIdSQLExpression(data),
                                     getValueSQLExpression(data)
                                     )
                              deepx <- setAlias(setAlias,"")
                              whereconditions <- ""
                              vmapping <- colnames(deepx)
                              names(vmapping) <- colnames(deepx)
                      }
                      else
                      {
                              deepx <- data
                              data@select@whereconditions <-
            c(data@select@whereconditions,whereconditions)
                              if(length(data@select@whereconditions)>0 &&
                                      data@select@whereconditions!=""){

                          ## Hadoop does not support _ in column names
                          ## for FLLinRegrMultiDataSet
                          if(is.FLTableMD(deepx)){
                              data <- setIndexSQLName(data,1,"groupid")
                              data <- setIndexSQLName(data,2,"obsid")
                              data <- setIndexSQLName(data,3,"varid")
                              data <- setIndexSQLName(data,4,"numval")
                          }
```

```
                             #sqlstr <- paste0("CREATE VIEW
",getOption("ResultDatabaseFL"),".",
                             #                                   deeptablename," AS
",constructSelect(data))
                             #t <- sqlSendUpdate(connection,sqlstr)
                             deeptablename<-createView(pViewName=gen_view_name("New"),
                                         pSelect=constructSelect(data))

                if(is.FLTableMD(deepx))
                    deepx <- FLTableMD(deeptablename,
                                    getGroupIdSQLName(data),
                                    getObsIdSQLName(data),
                                    getVarIdSQLName(data),
                                    getValueSQLName(data))
                else
                             deepx <- FLTable(deeptablename,
                              getObsIdSQLName(data),
                                    getVarIdSQLName(data),
                                    getValueSQLName(data))
                             deepx <- setAlias(deepx,"")
                     }
                     whereconditions <- ""
                     vmapping <- colnames(deepx)
                     names(vmapping) <- colnames(deepx)
         }

         ## Set RefLevel for Multinomial
         if(familytype=="multinomial"){
         if(is.null(pRefLevel)){
                     pRefLevel <- sqlQuery(getFLConnection(),
                                                                        paste0("SELECT
cell_val_colname FROM (",

         constructSelect(data),") a \n ",
                                                                                    "WHERE
obs_id_colname = 1 \n AND ",
                                                                                    "
var_id_colname = -1 \n "))[1,1]
         }
         pThreshold <- pRefLevel
    }

         vinclude <- NULL
         vexclude <- NULL
         ##Insert SpecID
         vspecID <- "NULL"
         if(is.list(specID) && length(specID)>0
                     && direction %in% c("UFbackward","Fbackward","backward"))
         {
                     vspecID <- genRandVarName()
                     vdf <- NULL
                     vspecIDTable <-
getRemoteTableName(tableName=ifelse(familytype=="linear","fzzlLinRegrModelVarSpec",

"fzzlLogRegrModelVarSpec"),temporaryTable=FALSE)
                     if(!is.null(specID[["include"]]))
                     {
                             #browser()
```

```
                            vinclude <-
vmapping[charmatch(specID[["include"]],names(vmapping))]
                            vinclude <- vinclude[!is.na(vinclude)]
                            if(is.null(vinclude) || length(vinclude) < 1)
                            stop("columns in lower are not in deeptable.",
                                     " Might be due to variable reduction during data
preparation")
                            # sqlstr <- c(sqlstr,paste0("INSERT INTO ", vspecIDTable,"
VALUES(",fquote(vspecID),",",
                            #                                   vinclude,",","'I')"))
            vdf <- rbind(vdf,data.frame(vspecID,vinclude,"I"))
                  }
                  if(!is.null(specID[["exclude"]]))
                  {
                            vexclude <-
vmapping[charmatch(specID[["exclude"]],names(vmapping))]
                            vexclude <- vexclude[!is.na(vexclude)]
                            if(length(vexclude)>0 && vexclude!="")
                            # sqlstr <- c(sqlstr,paste0("INSERT INTO ",vspecIDTable,"
VALUES(",fquote(vspecID),",",
                            #                                   vexclude,",","'X')"))
            vdf <- rbind(vdf,data.frame(vspecID,vexclude,"X"))
                  }
                  if(!is.null(vdf))
            t <- insertIntotbl(pTableName=vspecIDTable,
                                  pValues=vdf)
  #                   t <- sqlSendUpdate(getFLConnection(),paste0(sqlstr,collapse=";"))
        }

    result <- list(deepx=deepx,
                   wideTable=data, ## todo: in case a deep table was passed this is not a
wideTable
                   wideToDeepAnalysisID=wideToDeepAnalysisID,
                   formula=formula,
                   vmapping=vmapping,
                   mapTable=mapTable,
                   vspecID=vspecID,
                   highestpAllow1=highestpAllow1,
                   highestpAllow2=highestpAllow2,
                   topN=topN,
                   vexclude=vexclude,
                   vallVars=vallVars,
                   stepWiseDecrease=stepWiseDecrease,
                   eventweight=eventweight,
                   noneventweight=noneventweight,
                   maxiter=maxiter,
                   pThreshold=pThreshold,
                   offset=offset,
                   RegrDataPrepSpecs=vRegrDataPrepSpecs)
    class(result) <- "FLpreparedData"
        return(result)
}

#' @export
prepareData.lmGeneric <- prepareData.formula

#' @export
prepareData.NULL <- prepareData.formula
```

```
#' @export
prepareData.character <- prepareData.formula

## move to file lm.R
#' @export
`$.FLLinRegr`<-function(object,property){
                                        #parentObject <- deparse(substitute(object))
    parentObject <- unlist(strsplit(unlist(strsplit(as.character(sys.call()),
                                "(",fixed=T))[2],",",fixed=T))[1]
    if(property=="coefficients"){
        coefficientsvector <- coefficients(object)
        assign(parentObject,object,envir=parent.frame())
        return(coefficientsvector)
    }
    else if (property=="residuals"){
        residualsvector <- residuals(object)
        assign(parentObject,object,envir=parent.frame())
        return(residualsvector)
    }
    else if(property=="fitted.values")
    {
        fitvector <- fitted.values.FLGAM(object)
        assign(parentObject,object,envir=parent.frame())
        return(fitvector)
    }
    else if(property=="FLCoeffStdErr")
    {
        coeffVector <- coefficients(object)
        assign(parentObject,object,envir=parent.frame())
        return(object@results[["FLCoeffStdErr"]])
    }
    else if(property=="FLCoeffTStat")
    {
        coeffVector <- coefficients(object)
        assign(parentObject,object,envir=parent.frame())
        return(object@results[["FLCoeffTStat"]])
    }
    else if(property=="FLCoeffPValue")
    {
        coeffVector <- coefficients(object)
        assign(parentObject,object,envir=parent.frame())
        return(object@results[["FLCoeffPValue"]])
    }
    else if(property=="FLCoeffNonZeroDensity")
    {
        coeffVector <- coefficients(object)
        assign(parentObject,object,envir=parent.frame())
        return(object@results[["FLCoeffNonZeroDensity"]])
    }
    else if(property=="FLCoeffCorrelWithRes")
    {
        coeffVector <- coefficients(object)
        assign(parentObject,object,envir=parent.frame())
        return(object@results[["FLCoeffCorrelWithRes"]])
    }
    else if(property == "s")
    {
```

```
    if(!is.null(object@results[["s"]]))
    {
        return(object@results[["s"]])
    }
    str <- paste0("SELECT * FROM ",object@vfcalls["statstablename"],
                " a WHERE a.AnalysisID = '",object@AnalysisID,"'")
    t <- sqlQuery(connection, str)
    val <- t[t$Notation == "Mad_S",3]
    object@results <- c(object@results, list(s = val))
    assign(parentObject,object,envir=parent.frame())
    return(val)
    }
else if(property=="call")
{
    return(object@results[["call"]])
}
else if(property=="FLLinRegrStats")
{
    if(!is.null(object@results[["FLLinRegrStats"]]))
        return(object@results[["FLLinRegrStats"]])
    else
    {
        sqlstr <- paste0("SELECT * FROM ",object@vfcalls[["statstablename"]]," \n",
                         " WHERE AnalysisID=",fquote(object@AnalysisID),
                         " \nAND ModelID=",object@results[["modelID"]])

        statsdataframe <- sqlQuery(getFLConnection(),sqlstr)
        object@results <- c(object@results,list(FLLinRegrStats=statsdataframe))
        assign(parentObject,object,envir=parent.frame())
        return(statsdataframe)
    }
}
else if(property=="df.residual")
{
    if(object@vfcalls["functionName"] == "FLRobustRegr")
        return(NULL)
    else
    {
        statsdataframe <- object$FLLinRegrStats
        colnames(statsdataframe) <- toupper(colnames(statsdataframe))
        dfResidualVector <- statsdataframe[["DFRESIDUAL"]]
        object@results <- c(object@results,list(df.residual=dfResidualVector))
        assign(parentObject,object,envir=parent.frame())
        return(dfResidualVector)
    }
}
else if(property=="model")
{
    ## The Column order may not be same as
    ## in formula object because add. columns
    ## may be added by categorical trans.
    ##This might stop any parent script!!
    ##Need something that has wait time and
    ## Default value.
    modelframe <- model.FLLinRegr(object)
    ## Do not store. Better to fetch each time as
    ## it saves memory and not much time loss in
    ## Fetching.
    ##object@results <- c(object@results,list(model=modelframe))
    assign(parentObject,object,envir=parent.frame())
    return(modelframe)
}
else if(property=="x")
{
    if(!is.null(object@results[["x"]]))
        return(object@results[["x"]])

    modelframe <- getXMatrix(object,
                             pDropCols=c(-1))
    object@results <- c(object@results,list(x=modelframe))
    assign(parentObject,object,envir=parent.frame())
    return(modelframe)
}
else if(property=="y")
{
    ##This is safer from simple subsetting of
    ## WideTable as whereConditions may exist
    if(!is.null(object@results[["y"]]))
        return(object@results[["y"]])
    else
    {
        vtablename <- getTableNameSlot(object@deeptable)
        obs_id_colname <- getObsIdSQLExpression(object@deeptable)
        var_id_colname <- getVarIdSQLExpression(object@deeptable)
        cell_val_colname <- getValueSQLExpression(object@deeptable)

        sqlstr <- paste0("SELECT '%insertIDhere%' AS vectorIdColumn,\n",
                         obs_id_colname," AS vectorIndexColumn,\n",
                         cell_val_colname," AS vectorValueColumn\n",
                         " FROM ",vtablename,
                         " \nWHERE ",var_id_colname," = -1 \n")

        tblfunqueryobj <- new("FLTableFunctionQuery",
                              connectionName = getFLConnectionName(),
                              variables = list(
                                  obs_id_colname = "vectorIndexColumn",
                                  cell_val_colname = "vectorValueColumn"),
                              whereconditions="",
                              order = "",
                              SQLquery=sqlstr)

        yvector <- newFLVector(
            select = tblfunqueryobj,
            Dimnames = list(object@deeptable@Dimnames[[1]],
                            "vectorValueColumn"),
            dims = as.integer(c(nrow(object@deeptable),1)),
            isDeep = FALSE)
        object@results <- c(object@results,list(y=yvector))
        assign(parentObject,object,envir=parent.frame())
        return(yvector)
    }
}
else if(property=="qr" || property=="rank")
{
    if(!is.null(object@results[["qr"]]))
    {
        if(property=="qr")
            return(object@results[["qr"]])
        else if(property=="rank")
            return(object@results[["qr"]]$rank)
    }
    else
    {
        modelmatrix <- object$x
        if(nrow(modelmatrix)>700
           || ncol(modelmatrix)>700)
            modelmatrix <- as.matrix(modelmatrix)
                                    # modelmatrix <- as.matrix(object$x)
                                    # qrList <- base::qr(modelmatrix)
        qrList <- qr(modelmatrix)
        vrank <- qrList$rank
        object@results <- c(object@results,list(qr=qrList))
        assign(parentObject,object,envir=parent.frame())
        if(property=="qr")
            return(qrList)
        else if(property=="rank") return(vrank)
    }
}
else if(property=="terms")
{
    if(!is.null(object@results[["terms"]]))
        return(object@results[["terms"]])
    else
    {
        coeffVector <- object$coefficients
        vallVars <- all.vars(object@formula)
        vcolnames <- object@results[["modelColnames"]][-1]
        if(is.null(vcolnames))
            vcolnames <- names(coeffVector)[2:length(coeffVector)]
        vterms <- terms(formula(paste0(vallVars[1],"~",
                                       paste0(vcolnames,collapse="+"))))
        object@results <- c(object@results,list(terms=vterms))
        assign(parentObject,object,envir=parent.frame())
        return(vterms)
    }
}
else if(property=="xlevels")
{
    cat("categorical variables are Transformed \n ")
    return(list())
}
else if(property=="assign")
{
    return(c(0,rep(1,length(all.vars(object@formula))-1)))
}

else if(property=="formula")
{
    if(!is.null(object@results[["formula"]]))
        return(object@results[["formula"]])
    else
    {
        coeffVector <- object$coefficients
        vallVars <- all.vars(object@formula)
```

```
            vcolnames <- object@results[["modelColnames"]][-1]
            if(is.null(vcolnames))
                vcolnames <- names(coeffVector)[2:length(coeffVector)]
            vterms <- terms(formula(paste0(vallVars[1],"~",
                                           paste0(vcolnames,collapse="+"))))
            object@results <- c(object@results,list(terms=vterms))
            assign(parentObject,object,envir=parent.frame())
            return(vterms)
        }
    }

    else if(property=="anova") stop("This feature is not available yet.")

    else stop("That's not a valid property \n ")
}

setMethod("names", signature("FLRobustRegr"), function(x) c("coefficients",
                                                            "residuals",
                                                            "fitted.values",
                                                            "x",
                                                            "y",
                                                            "call" ))

#' @export
setMethod("names", signature("FLLinRegr"), function(x) c("anova", "formula", "assign",
                                                          "xlevels","y","x","model",
                                                          "df.residual","FLLinRegrStats",

"call","s","FLCoeffCorrelWithRes"
                                                         ,"FLCoeffNonZeroDensity",
                                                          "FLCoeffPValue","FLCoeffTStat",

"FLCoeffStdErr","fitted.values",
                                                          "residuals","coefficients" ))


#' @export
coefficients<-function(table){
         UseMethod("coefficients",table)
}
#' @export
coefficients.default <- stats::coefficients

## move to file lm.R
#' @export
coefficients.FLLinRegr<-function(object){
    parentObject <- unlist(strsplit(unlist(strsplit(
                             as.character(sys.call()),
                             "(",fixed=T))[2],")",fixed=T))[1]
    if(is.FLTableMD(object@table))
        coeffVector <- coefficients.FLLinRegrMD(object)
    else
         coeffVector <- coefficients.lmGeneric(object,

FLCoeffStats=c(object@results$mod["FLCoeffStdErr"],

object@results$mod["FLCoeffPValue"],
object@results$mod["FLCoeffTStat"],

object@results$mod["FLCoeffCorrelWithRes"],

object@results$mod["FLCoeffNonZeroDensity"]))
    assign(parentObject,object,envir=parent.frame())
    return(coeffVector)
}

## move to file lmGeneric.R
#' @export
coefficients.lmGeneric <-function(object,
                                  FLCoeffStats=c(),
                                  pIntercept=TRUE,
                                  ...){
    if(!is.null(object@results[["coefficients"]]))
         return(object@results[["coefficients"]])
    else
    {
        ## Since Currently only 1000 Columns are supported
        ## by FLLinRegr, fetch them.
                                        #browser()
                                        # vmapping <- NULL
        vID <- object@results$mod[["nID"]]
        vfcalls <- object@vfcalls
        vcoeffnames <- NULL
        vmodelnames <- NULL
        if(isDeep(object@table))
            coeffVector <- sqlQuery(getFLConnection(),
                                 paste0("SELECT * FROM ",vfcalls["coefftablename"],
                                        " where
AnalysisID=",fquote(object@AnalysisID),
                                        ifelse(length(object@results[["modelID"]])>0
&&
                                             object@vfcalls[["functionName"]] !=
"FLRobustRegr" &&
                                             object@vfcalls["functionName"] !=
"FLPLSRegr",
                                               paste0(" AND
ModelID=",object@results[["modelID"]]),""),
                                        " ORDER BY ",vID))
        else{
                                        #browser()
            vcoeffframe <- sqlQuery(getFLConnection(),
                                 paste0("SELECT a.*,b.* \n",
                                        " FROM
",getSystemTableMapping("fzzlRegrDataPrepMap")," AS a, \n ",
                                        vfcalls["coefftablename"]," AS b \n",
                                        " WHERE a.Final_VarID = b.",vID," \n",
                                        " AND a.AnalysisID =
",fquote(object@wideToDeepAnalysisID),
                                        "\n AND b.AnalysisID =
",fquote(object@AnalysisID),
                                        ifelse(length(object@results[["modelID"]])>0
&&
                                         object@vfcalls[["functionName"]] !=
"FLRobustRegr" &&
                                                object@vfcalls[["functionName"]] !=
"FLPLSRegr",
                                                      paste0("\n AND b.ModelID =
",object@results[["modelID"]]),""),
                                               "\n ORDER BY b.",vID))

            colnames(vcoeffframe) <- toupper(colnames(vcoeffframe))
            # vcolumnnames <- unique(as.character(vcoeffframe[["COLUMN_NAME"]]))
            # vcolumnnames <- vcolumnnames[2:length(vcolumnnames)]
            # vmodelnames <- c(all.vars(object@formula)[1],
            #                   vcolumnnames)
            coeffVector <- vcoeffframe
            vcolumnnames <- vcoeffframe[["COLUMN_NAME"]]
            vcolumnnames <- vcolumnnames[2:length(vcolumnnames)]
            want <- all.vars(object@formula)
            want <- want[2:length(want)]
            q <- unlist(sapply(want,
                               function(x){
                                   which(toupper(vcolumnnames) %in% toupper(x))}
                               ))+1
            coeffVector <- coeffVector[c(1,q), ]
            vmodelnames <- c(all.vars(object@formula)[1],
                             unique(as.character(coeffVector[["COLUMN_NAME"]])[-1]))
            vVarnames <- colnames(object@table)
            vmapNames <- function(t)
            {
                vindex <- match(tolower(t),tolower(vVarnames))
                if(is.na(vindex))
                    vnames <- t
                else vnames <- vVarnames[vindex]
                vnames
            }

            vcoeffnames <- as.vector(apply(coeffVector,1,
                                           function(x){
if(tolower(x["VAR_TYPE"])%in%c("category","varchar"))
return(paste0(vmapNames(x["COLUMN_NAME"]),x["CATVALUE"]))
                                               else
return(vmapNames(x["COLUMN_NAME"]))}))

            # vcoeffnames <- sapply(vcoeffnames,vmapNames)
        }

        colnames(coeffVector) <- toupper(colnames(coeffVector))
        coeffVector1 <- coeffVector[[object@results$mod[["nCoeffEstim"]]]]
                                         # vmapping <- as.FLVector(unique(c(-2,-
1,coeffVector[["COEFFID"]])))
        if(!is.null(vcoeffnames)){
            if(!pIntercept)
                names(coeffVector1) <- as.character(vcoeffnames)
            else
                names(coeffVector1) <- c("(Intercept)",
as.character(vcoeffnames)[2:length(vcoeffnames)])
        }
        else{
```

```r
                vallVars <- all.vars(genDeepFormula(coeffVector[[vID]]))
                names(coeffVector1) <- c("(Intercept)",vallVars[2:length(vallVars)])
                if(!pIntercept)
                    names(coeffVector1) <- vallVars
            }
                                            # to remove null values.
            FLCoeffStats <- FLCoeffStats[FLCoeffStats!= ""]
            FLCoeffStats  <- lapply(FLCoeffStats,
                                    function(x){
                                        t<-coeffVector[[x]]
                                        names(t)<-names(coeffVector1)
                                        t
                                    })

            vcolnames <- colnames(object@deeptable)
            droppedCols <- vcolnames[!vcolnames %in% c("-1",coeffVector[[vID]])]
            object@results <- c(object@results,
                                list(coefficients=coeffVector1,
                                     droppedCols=droppedCols),
                                FLCoeffStats)
            object@results[["modelColnames"]]<-vmodelnames
            object@results[[vID]] <- as.numeric(coeffVector[[vID]])
                                            # object@results[["varIDMapping"]] <- vmapping
            parentObject <- unlist(strsplit(unlist(strsplit(
                as.character(sys.call()),"(",fixed=T))[2],")",fixed=T))[1]
            assign(parentObject,object,envir=parent.frame())
            return(coeffVector1)
        }
}

## move to file lm.R
#' @export
residuals.FLLinRegr<-function(object)
{
    if(!is.null(object@results[["residuals"]]))
         return(object@results[["residuals"]])
    else
    {

        residualsvector <- calcResiduals(object=object)
        object@results <- c(object@results,list(residuals=residualsvector))
        parentObject <- unlist(strsplit(unlist(strsplit(
            as.character(sys.call()),"(",fixed=T))[2],")",fixed=T))[1]
        assign(parentObject,object,envir=parent.frame())
        return(residualsvector)
    }
}

## move to file lm.R
#' @export
model.FLLinRegr <- function(object,...)
{
         if(!is.null(object@results[["model"]]))
         return(object@results[["model"]])
         else
         {
                  coeffVector <- object$coefficients
                  vallVars <- all.vars(object@formula)
                  vcolnames <- NULL
                  if(!is.null(object@results[["modelColnames"]])){
                           vcolnames <- object@results[["modelColnames"]]
                           modelframe <- object@table
                           modelframe@Dimnames[[2]] <- vcolnames
                  }
                  else{
                           vdroppedCols <- object@results[["droppedCols"]]
                           modelframe <- object@deeptable
                           modelframe@select@whereconditions <-
c(modelframe@select@whereconditions,

                  paste0(getVarIdSQLExpression(object@deeptable),
                                                        " NOT IN ","(",paste0(c(0,-
2,vdroppedCols),
                                                            collapse=","),

                                    ")"))

                           if(is.matrix(coeffVector))
                                    vcolnames <- c(vallVars[1],

         colnames(coeffVector)[2:ncol(coeffVector)])
                           else
                                    vcolnames <- c(vallVars[1],

         names(coeffVector)[2:length(coeffVector)])
                  }

                  ## Have to implement names for FLTable
                  # modelframe@Dimnames <- list(modelframe@Dimnames[[1]],
                  #                                                     vcolnames)
                  # return(modelframe)

                  modelframe <- as.data.frame(modelframe)
                  if(!is.null(names(vcolnames)))
                           colnames(modelframe) <- names(vcolnames)
                  else colnames(modelframe) <- vcolnames

                  object@results <- c(object@results,list(model=modelframe))
                  parentObject <- unlist(strsplit(unlist(strsplit(
                           as.character(sys.call()),"(",fixed=T))[2],")",fixed=T))[1]
                  assign(parentObject,object,envir=parent.frame())
                  return(modelframe)
         }
}

#' @export
summary.FLRobustRegr <- function(object, ...){
    str <- paste0("SELECT a.StdDev, a.T_Val FROM ",object@vfcalls["coefftablename"]," a
                         WHERE a.AnalysisID = ",fquote(object@AnalysisID),"
                         ")
    df <- sqlQuery(connection, str)
    vcoeff <- data.frame(coefficients(object), df)
    names(vcoeff) <- c("Value", "Std.Error", "t Value")
    vresiduals <- as.vector(object$residuals)
    vdf <- c(length(object@deeptable@Dimnames[[2]]),
             length(object@deeptable@Dimnames[[1]]) + 1 -
length(object@deeptable@Dimnames[[2]]),
             length(object@deeptable@Dimnames[[2]]))

    reqList <- list(call = object$call,
                    residuals=vresiduals,
                    coefficients = vcoeff,
                    sigma = object$s,
                    stddev = NA,
                    df = vdf,
                    r.squared = NA,
                    cov.unscaled = NA,
                    terms = NA
                    )
                                        #print(reqList)
    class(reqList) <- "summary.rlm"
    parentObject <- unlist(strsplit(unlist(strsplit(as.character
    (sys.call()),"(",fixed=T))[2],")",fixed=T))[1]
    assign(parentObject,object,envir=parent.frame())
    return(reqList)
}
## move to file lm.R
#' @export
summary.FLLinRegr <- function(object,
                              calcResiduals=FALSE){
    if(is.FLTableMD(object@table))
        reqList <- summary.FLLinRegrMD(object)

    else
    {
        stat <- object$FLLinRegrStats
        colnames(stat) <- toupper(colnames(stat))
        coeffframe <- data.frame(object$coefficients,
                             object$FLCoeffStdErr,
                             object$FLCoeffTStat,
                             object$FLCoeffPValue)
        colnames(coeffframe)<-c("Estimate","Std. Error","t value","Pr(>|t|)")

                                        #put rowname
                                        # rname <- all.vars(object@formula)
                                        # rownames(coeffframe) <-
c(rownames(coeffframe)[1], rname[2:length(rname)])
        rownames(coeffframe) <- names(object$coefficients)

        if(calcResiduals)
            vresiduals <- as.vector(object$residuals)
        else vresiduals <- NULL
        reqList <- list(call = as.call(object@formula),
                        residuals  = vresiduals,
                        coefficients = as.matrix(coeffframe),
                        sigma = stat$STDERR,
                        df = as.vector(c((stat$DFREGRESSION + 1),stat$DFRESIDUAL,
(stat$DFREGRESSION + 1))),
                        r.squared = stat$RSQUARED,
                        adj.r.squared = stat$ADJRSQUARED,
                        fstatistic = c(stat$FSTAT, stat$DFREGRESSION, stat$DFRESIDUAL ),
                        aliased = FALSE
                        )
        class(reqList) <- "summary.lm"
        reqList
        }

    parentObject <- unlist(strsplit(unlist(strsplit(as.character
    (sys.call()),"(",fixed=T))[2],")",fixed=T))[1]
    assign(parentObject,object,envir=parent.frame())
    return(reqList)
}



## move to file lm.R
## Add deep statment, also problem can be of vobsid
## Use FLSUMPROD: usemethod dispatch, create new class.
#' @export
predict.FLLinRegr <- function(object,
                              newdata=object@table,
                              scoreTable="",
                              ...){

    return(predict.lmGeneric(object,newdata=newdata,
                             scoreTable=scoreTable,
                             ...))
}

predict.FLRobustRegr <- function(object,
                                 newdata=object@table,
                                 scoreTable="",
                                 ...)
{
    ObsID <- getVariables(object@deeptable)$obs_id_colname
    VarID <- getVariables(object@deeptable)$var_id_colname
    Num_Val <- getVariables(object@deeptable)$cell_val_colname

    str <- paste0(" SELECT  '%insertIDhere%' AS vectorIdColumn,
                                 b.",ObsID," AS VectorIndexColumn,
                                 FLSUMPROD(b.",Num_Val,",a.Est) AS vectorValueColumn FROM
",
                  object@vfcalls["coefftablename"]," a,",
                  getTableNameSlot(object@deeptable)," b
                         WHERE a.VarID  = b.",VarID," AND a.AnalysisID =
'",object@AnalysisID,"'
                         GROUP BY b.",ObsID,"")

    tblfunqueryobj <- new("FLTableFunctionQuery",
                          connectionName = getFLConnectionName(),
                          variables = list(
                              obs_id_colname = "vectorIndexColumn",
                              cell_val_colname = "vectorValueColumn"),
                          whereconditions="",
                          order = "",
                          SQLquery=str)
    flv <- newFLVector(
        select = tblfunqueryobj,
        Dimnames = list(rownames(object@table),
                        "vectorValueColumn"),
        dims = as.integer(c(newdata@dims[1],1)),
        isDeep = FALSE)
    return(flv)
}

## move to file lmGeneric.R
#' @export
predict.lmGeneric <- function(object,
                              newdata=object@table,
                              scoreTable="",
                              type="response",...){
    if(!is.FLTable(newdata) && class(newdata) != "FLpreparedData") stop("scoring allowed
on FLTable only")
    vfcalls <- object@vfcalls
    if(class(newdata) == "FLpreparedData"){
        newdata <- newdata$deepx
    }
    else{
        newdata <- prepareData(object,newdata,outDeepTableName="", ...) }

    newdata <- setAlias(newdata,"")

    if(scoreTable=="")
                                        # scoreTable <-
paste0(getOption("ResultDatabaseFL"),".",
                                        #
         gen_score_table_name(getTableNameSlot(object@table)))
         scoreTable <- gen_score_table_name(getTableNameSlot(object@table))
                                        # else if(!grep(".",scoreTable))
                                        #           scoreTable <-
paste0(getOption("ResultDatabaseFL"),".",scoreTable)

    vinputTable <- getTableNameSlot(newdata)
    vtable <- getTableNameSlot(newdata)
    vobsid <- getObsIdSQLExpression(newdata)
    vvarid <- getVarIdSQLExpression(newdata)
    vvalue <- getValueSQLExpression(newdata)

    vinputCols <- list()
    vinputCols <- c(vinputCols,
                    TableName=vtable,
                    ObsIDCol=vobsid,
                    VarIDCol=vvarid,
                    ValCol=vvalue
                    )
    if(!object@vfcalls["functionName"]=="FLPoissonRegr")
        vinputCols <- c(vinputCols,
                        WhereClause="NULL")
    vinputCols <- c(vinputCols,
                    RegrAnalysisID=object@AnalysisID,
                    ScoreTable=scoreTable)

         if(!is.Hadoop())
         vinputCols <- c(vinputCols,
                                             Note=genNote(paste0("Scoring
",vfcalls["note"])))

         AnalysisID <- sqlStoredProc(getFLConnection(),
         vfcalls["scoretablename"],

         outputParameter=c(AnalysisID="a"),

         pInputParams=vinputCols)
         AnalysisID <- checkSqlQueryOutput(AnalysisID)

    if(type %in% "link"){
         sqlQuery(getFLConnection(),paste0("alter table ",scoreTable," add logit float"))
         sqlQuery(getFLConnection(),paste0("update ",scoreTable," set logit = -ln(1/Y -
1) where Y<1"))
         object@vfcalls["valcolnamescoretable"]<-"logit"
    }
             sqlstr <- getFittedValuesLogRegrSQL(object,newdata,scoreTable)
                 # sqlstr <- paste0(" SELECT '%insertIDhere%' AS vectorIdColumn,",
                 #                                       vobsid," AS
vectorIndexColumn,",
                 #
         vfcalls["valcolnamescoretable"]," AS vectorValueColumn",
                 #                               " FROM ",scoreTable)

                 tblfunqueryobj <- new("FLTableFunctionQuery",
                                 connectionName = getFLConnectionName(),
                                 variables = list(
                                                   obs_id_colname = "vectorIndexColumn",
                                                   cell_val_colname =
"vectorValueColumn"),
                                 whereconditions="",
                                 order = "",
                                 SQLquery=sqlstr)

                 flv <- newFLVector(
                                            select = tblfunqueryobj,
                                            Dimnames = list(rownames(newdata),

         "vectorValueColumn"),
                         dims = as.integer(c(newdata@dims[1],1)),
                                            isDeep = FALSE)

                 return(flv)
}

#' Print FLLinRegr Object
#'
#' Printing of output from Linear Regression
#'
#' @title Print FLLinRegr output Info
#' @method print FLLinRegr
#' @param object prints results of FLLinRegr on FL objects
#' @method coefficients FLLinRegr
#' @param object returns coefficient vector of the object
#' @method residuals FLLinRegr
#' @param object the residuals, that is response minus fitted values.
#' @method influence FLLinRegr
#' @param object returns the basic quantities which are used in forming a wide variety of
diagnostics for checking the quality of regression fits.
#' @method lm.influence FLLinRegr
#' @param object returns the basic quantities which are used in forming a wide variety of
diagnostics for checking the quality of regression fits.
#' @method plot FLLinRegr
#' @param object plots the results of FLLinRegr on FL objects.
#' @method summary FLLinRegr
#' @method predict FLLinRegr
#' @export
print.FLLinRegr<-function(object){
         parentObject <- unlist(strsplit(unlist(strsplit(
                  as.character(sys.call()),"(",fixed=T))[2],")",fixed=T))[1]
         reqList <- list(call=object$call,
                                            coefficients=object$coefficients)

         class(reqList) <- "lm"
         assign(parentObject,object,envir=parent.frame())
         print(reqList)
}

## move to file lm.R
#' @export
setMethod("show","FLLinRegr",print.FLLinRegr)

## move to file lm.R
#' @export
plot.FLLinRegr <- function(object,method="R",limit=4000,...)
{
    parentObject <- unlist(strsplit(unlist(strsplit(
                         as.character(sys.call()),"(",fixed=T))[2],")",fixed=T))[1]
    if(method=="R"){
        vqr <- object$qr
        vqr <- list(qr=as.matrix(vqr$qr),
                    rank=as.integer(as.vector(vqr$rank)),
                    qraux=as.numeric(as.vector(vqr$qraux)),
                    pivot=as.integer(as.vector(vqr$pivot)))

        class(vqr)<-"qr"
        reqList <- list(residuals=as.vector(object$residuals),
                        coefficients=object$coefficients,
                        df.residual=object$df.residual,
                        qr=vqr,
                        rank=vqr$rank,
                        call=object$call,
                        xlevels=object$xlevels,
                        model=object$model,
                        terms=object$terms)
        class(reqList) <- "lm"
        assign(parentObject,object,envir=parent.frame())
        plot(reqList,...)
    }
    else{
        vObsIdColname <- getVariables(object@deeptable)[["obs_id_colname"]]
        vVarIdColname <- getVariables(object@deeptable)[["var_id_colname"]]
        vCellValColname <- getVariables(object@deeptable)[["cell_val_colname"]]
        p <- min(limit,length(object$fitted.values))/length(object$fitted.values)

        sqlstr <- paste0("SELECT \n ",
                                 " b.",vCellValColname," AS y, \n ",
                                 " a.y AS yhat, \n ",
                                        " (b.",vCellValColname," - a.y) AS residual \n ",
                                " FROM ",object@scoreTable," a, \n ",
                                        getTableNameSlot(object@deeptable)," b \n ",
                                " WHERE b.",vObsIdColname,"=a.",vObsIdColname,
                                        " AND b.",vVarIdColname,"=-1 ",
                                        " AND FLSimUniform(",
                                            getNativeRandFunction(pArg1=1,pArg2=10000),
                                            ", 0.0, 1.0) < ",p)

        vdf <- sqlQuery(getFLConnection(),sqlstr)
        vfit <- vdf[["yhat"]]
        vresid <- vdf[["residual"]]
        vactual <- vdf[["y"]]
        assign(parentObject,object,envir=parent.frame())
        plot(vfit,vresid,xlab="fitted.values",ylab="residuals",main="residual plot")
        readline("Hit <Return> to see next plot:")
        plot(vactual,vfit,xlab="actual values",ylab="fitted.values",main="Actual vs
Fitted")
    }
}

## move to file lm.R
#' @export
influence.FLLinRegr <- function(model,...){
         reqList <- list(residuals=as.vector(model$residuals),
                                            coefficients=model$coefficients,
                                            df.residual=model$df.residual,
                                            qr=model$qr,
                                            rank=model$rank,
                                            call=model$call,
                                            xlevels=model$xlevels,
                                            model=model$model,
                                            termsz=model$terms)
         class(reqList) <- "lm"
         parentObject <- unlist(strsplit(unlist(strsplit(as.character
                  (sys.call()),"(",fixed=T))[2],")",fixed=T))[1]
         assign(parentObject,model,envir=parent.frame())
         return(stats::influence(reqList,...))
}

#' @export
lm.influence <- function(model,do.coef=TRUE,...){
         UseMethod("lm.influence",model)
}

#' @export
lm.influence.default <- stats::lm.influence

#' @export
lm.influence.FLLinRegr <- function(model,do.coef=TRUE,...){
         reqList <- list(residuals=as.vector(model$residuals),
                                            coefficients=model$coefficients,
                                            df.residual=model$df.residual,
                                            qr=model$qr,
                                            rank=model$rank,
                                            call=model$call,
                                            xlevels=model$xlevels,
                                            model=model$model,
                                            terms=model$terms)
         class(reqList) <- "lm"
         parentObject <- unlist(strsplit(unlist(strsplit(as.character
                 (sys.call()),"(",fixed=T))[2],")",fixed=T))[1]
         assign(parentObject,model,envir=parent.frame())
         return(stats::lm.influence(reqList,do.coef,...))
}

##Return list of coefficients vectors
coefficients.FLLinRegrMD <- function(object){
         if(!is.null(object@results[["coefficients"]]))
         return(object@results[["coefficients"]])
         else
         {
                 if(isDeep(object@table)){
                         coeffVector <- sqlQuery(getFLConnection(),
                                                                         paste0("SELECT *
FROM ",object@vfcalls["coefftablename"],
                                                                                        "
where AnalysisID=",fquote(object@AnalysisID),
                                                                                        "
AND ModelID IN(",paste0(unlist(object@deeptable@Dimnames[[1]]),collapse=","),

         ") ORDER BY ModelID,CoeffID"))
            colnames(coeffVector) <- toupper(colnames(coeffVector))
                         vcoeffnames <- as.vector(apply(coeffVector,1,

         function(x){

         if(as.numeric(x[["COEFFID"]])=="0")

                 return("(Intercept)")

         else return(paste0("Var",x[["COEFFID"]]))

         }))
                 }
                 else{
                         vcoeffframe <- sqlQuery(getFLConnection(),

         paste0("SELECT a.*,b.* \n",

" FROM fzzlRegrDataPrepMDMap AS a, \n ",

object@vfcalls["coefftablename"]," AS b \n",

" WHERE a.Final_VarID = b.CoeffID \n",

         " AND a.AnalysisID = ",fquote(object@wideToDeepAnalysisID),

         "\n AND b.AnalysisID = ",fquote(object@AnalysisID),

         "\n AND a.groupID = b.modelID ",

         "\n AND b.ModelID IN(",

                 paste0(unlist(object@deeptable@Dimnames[[1]]),
                         collapse=","),

         ")\n ORDER BY ModelID,CoeffID"))
                         colnames(vcoeffframe) <- toupper(colnames(vcoeffframe))
                         want <- all.vars(object@formula)
                         want <- want[2:length(want)]
                         q <- dlply(vcoeffframe,"MODELID",function(x){
                                                      c(1,unlist(sapply(want,
                                                               function(y)
                                                                     which(

         as.character(x[["COLUMN_NAME"]][-1]) %in% y)

         ))+1)
                                                      })
                         for(i in 2:length(q)){
                                 q[[i]] <- q[[i]]+length(q[[i-1]])
                         }

                         coeffVector <- vcoeffframe[unlist(q),]
                         vcoeffnames <- as.vector(apply(coeffVector,1,

         function(x){

         if(x[["COLUMN_NAME"]]=="INTERCEPT")

                 return("(Intercept)")

         else if(tolower(x["VAR_TYPE"])=="category")

                 return(paste0(x["COLUMN_NAME"],x["CATVALUE"]))

         else return(x["COLUMN_NAME"])

         }))
                 }

                 coeffVector[["COEFFNAMES"]] <- vcoeffnames

                 vcoeffList <- dlply(coeffVector,"MODELID",
                                                     function(x){
                                                               vcoeff <-
x[["COEFFVALUE"]]
                                                               names(vcoeff) <-
x[["COEFFNAMES"]]
                                                               return(vcoeff)
                                                               })
                 names(vcoeffList) <-
paste0("Model",unlist(object@deeptable@Dimnames[[1]]))
                 parentObject <- unlist(strsplit(unlist(strsplit(as.character

         (sys.call()),"(",fixed=T))[2],")",fixed=T))[1]
                 object@results[["coefficients"]] <- vcoeffList
                 object@results[["coeffframe"]] <- coeffVector
                 assign(parentObject,object,envir=parent.frame())
                 return(vcoeffList)
         }
```

```
}

#' @export
summary.FLLinRegrMD <- function(object){
         vcoeffList <- object$coefficients
         coeffframe <- object@results[["coeffframe"]]
         if(is.null(object@results[["statsframe"]]))
                  statsframe <- sqlQuery(getFLConnection(),
                                                       paste0("SELECT * FROM
",object@vfcalls["statstablename"],
                                                                         " WHERE
AnalysisID=",fquote(object@AnalysisID),
                                                                         " ORDER BY MODELID
"))
         else statsframe <- object@results[["statsframe"]]
         colnames(statsframe) <- toupper(colnames(statsframe))
         vresList <- lapply(unlist(object@deeptable@Dimnames[[1]]),
                                              function(x){
                                                       vtemp <-
coeffframe[coeffframe[,"MODELID"]==x,]
                                                       vrownames <- vtemp[["COEFFNAMES"]]
                                                       vtemp <-
vtemp[,c("COEFFVALUE","STDERR","TSTAT","PVALUE","NONZERODENSITY","CORRELWITHRES")]
                                                       vcoeffmat <- as.matrix(vtemp)
                                                       rownames(vcoeffmat) <- vrownames
                                                       colnames(vcoeffmat) <-
c("Estimate","Std.Err","t-stat",

                  "p-value","non-zero Density","Correl With Residual")
                                                       vtemp <-
statsframe[statsframe[,"MODELID"]==x,]
                                                       vsummaryList <-
list(coeffframe=vcoeffmat,

         statsframe=vtemp,

         call=object$call)
                                                       class(vsummaryList) <-
"summary.FLLinRegrMD"
                                                       return(vsummaryList)
                                                       })
         names(vresList) <- paste0("Model",unlist(object@deeptable@Dimnames[[1]]))
         parentObject <- unlist(strsplit(unlist(strsplit(as.character

         (sys.call()),"(",fixed=T))[2],")",fixed=T))[1]
         object@results[["statsframe"]] <- statsframe
         assign(parentObject,object,envir=parent.frame())
         return(vresList)
}

#' @export
print.summary.FLLinRegrMD <- function(object){
         ret <- object$statsframe
         cat("Call:\n")
         cat(paste0(object$call),"\n")
         cat("\n\nCoefficients:\n")
         print(object$coeffframe)
         cat("\n---\n")
```

```
        cat("Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '' 1\n")
        cat("Residual standard error: ",ret[["MSRESIDUAL"]]," on ",ret[["DFRESIDUAL"]],"
degrees of freedom\n\n")
        cat("Multiple R-squared: ",ret[["RSQUARED"]]," , Adjusted R-squared:
",ret[["ADJRSQUARED"]],"\n")
        FStatPVal<-
pf(ret[["FSTAT"]],ret[["DFREGRESSION"]],ret[["DFRESIDUAL"]],lower.tail=FALSE)
        cat("F-statistic: ",ret[["FSTAT"]]," on ",ret[["DFREGRESSION"]]," and
",ret[["DFRESIDUAL"]]
                 ,"DF , p-value: ",FStatPVal,"\n")
        cat("MSRegression: ",ret[["MSREGRESSION"]]," , SigFStat:
",ret[["SIGFSTAT"]],"\n")
        cat("DWStat: ",ret[["DWSTAT"]]," , ResidualAutoCorrel:
",ret[["RESIDUALAUTOCORREL"]],"\n")
        cat("BPStat: ",ret[["BPSTAT"]]," , SigBPStat: ",ret[["SIGBPSTAT"]],"\n")
}

#' @export
print.FLLinRegrMD <- summary.FLLinRegrMD

#' @export
`[[.FLLinRegr`<-function(object,property){
    #parentObject <- deparse(substitute(object))
    parentObject <- unlist(strsplit(unlist(strsplit(as.character(sys.call()),
                        "(",fixed=T))[2],",",fixed=T))[1]
    vresult <- `$.FLLinRegr`(object=object,property=property)
    assign(parentObject,object,envir=parent.frame())
    return(vresult)

}
setDefaultsRegrDataPrepSpecs <- function(x,values){
    x <- as.list(x)
    for(i in c("depCol", "catToDummy","performNorm",
                "performVarReduc","minStdDev",
                "maxCorrel","makeDataSparse",
                "excludeCols","classSpec")){
        if(i %in% names(values))
            x[[i]] <- values
        else if(is.null(x[[i]])){
            if(i %in% c("maxCorrel","makeDataSparse"))
                x[[i]] <- 1
            else if(i %in% c("excludeCols","depCol",
                             "outDeepTableName","outObsIDCol",
                             "outVarIDCol","outValueCol",
                             "whereconditions"))
                x[[i]] <- ""
            else if(i %in% "classSpec")
                x[[i]] <- list()
            else x[[i]] <- 0
        }
    }
    x[["depCol"]] <- ""
    x
}

getFittedValuesLogRegrSQL <- function(object,newdata,scoreTable){
    UseMethod("getFittedValuesLogRegrSQL",newdata)
}
```

```
getFittedValuesLogRegrSQL.FLTable.Hadoop <- function(object,newdata,scoreTable){
    vobsid <- "ObsID"
    vfcalls <- object@vfcalls
    getFLVectorTableFunctionQuerySQL(indexColumn=vobsid,
                                     valueColumn=vfcalls["valcolnamescoretable"],
                                     FromTable=scoreTable)
}

getFittedValuesLogRegrSQL.FLTableDeep.Hadoop <- getFittedValuesLogRegrSQL.FLTable.Hadoop

getFittedValuesLogRegrSQL.FLTable.TDAster <- function(object,newdata,scoreTable){
    vobsid <- "obsid"
    vfcalls <- object@vfcalls
    getFLVectorTableFunctionQuerySQL(indexColumn=vobsid,
                                     valueColumn=vfcalls["valcolnamescoretable"],
                                     FromTable=scoreTable)
}

getFittedValuesLogRegrSQL.FLTableDeep.TDAster <-
getFittedValuesLogRegrSQL.FLTable.TDAster

getFittedValuesLogRegrSQL.default <- function(object,newdata,scoreTable){
    vobsid <- getObsIdSQLExpression(newdata)
    vfcalls <- object@vfcalls
    getFLVectorTableFunctionQuerySQL(indexColumn=vobsid,
                                     valueColumn=vfcalls["valcolnamescoretable"],
                                     FromTable=scoreTable)
}

getFittedValuesLogRegrSQL.FLpreparedData <- function(object,newdata,scoreTable){
    vobsid <- newdata$deepx@select@variables$obs_id_colname
    vfcalls <- object@vfcalls
    getFLVectorTableFunctionQuerySQL(indexColumn=vobsid,
                                     valueColumn=vfcalls["valcolnamescoretable"],
                                     FromTable=scoreTable)
}

#' @export
summary.FLLinRegrSF<-function(object,modelid=1){
         AnalysisID<-object@AnalysisID
         statstablename<-object@vfcalls["statstablename"]
         query<-paste0("Select * from ",statstablename, " Where AnalysisID = ",
                                     fquote(AnalysisID)," And modelid =",modelid)
         x<-sqlQuery(getFLConnection(),query)
         coeff<-sqlQuery(getFLConnection(),paste0("Select * from
",object@vfcalls["coefftablename"],

          " Where AnalysisID=",fquote(AnalysisID)," And modelid=coeffid"))
         coeffframe <- data.frame(coefficients(object),
                          t_stat=coeff$TSTAT,
                          p_value=coeff$PVALUE)
         reqList <- list(call = as.call(object@formula),
                                     residuals  = NULL,
                        coefficients = as.matrix(coeffframe),
                        sigma = x$STDERR,
                        df = as.vector(c((x$DFREGRESSION + 1),x$DFRESIDUAL,
(x$DFREGRESSION + 1))),
                        #r.squared = x$RSQUARED,
                        #adj.r.squared = x$ADJRSQUARED,
                        #fstatistic = c(x$FSTAT, x$DFREGRESSION, x$DFRESIDUAL ),
                        aliased = FALSE
                            )
    class(reqList) <- "summary.lm"
    reqList
}

#' @export
`$.FLLinRegrSF`<-function(object,property){
        if(property=="coefficients")
        return(coefficients(object))
}

#' @export
coefficients.FLLinRegrSF<-function(object){
        AnalysisID<-object@AnalysisID
        coefftablename<-object@vfcalls["coefftablename"]
        statstablename<-object@vfcalls["statstablename"]
        query1<-paste0("Select a.modelid, a.coeffvalue From ",coefftablename,
                                    " a Where AnalysisID= ",fquote(AnalysisID),"
And a.modelid=a.coeffid ORDER BY 1")
        query2<-paste0("Select a.modelid, a.coeffvalue From ",coefftablename,
                                    " a Where AnalysisID= ",fquote(AnalysisID),"
And a.modelid!=a.coeffid ORDER BY 1")
        query3<-paste0("Select a.ModelID, a.AdjRSquared, a.RSquared, a.StdErr, a.FStat
from ",statstablename,
                                " a Where AnalysisID=",fquote(AnalysisID)," Order
By 1")
        a<-sqlQuery(getFLConnection(),query1)
        b<-sqlQuery(getFLConnection(),query2)
        c<-sqlQuery(getFLConnection(),query3)
        ret<-data.frame(ModelID=a$MODELID,
                                    Intercept=b$COEFFVALUE,
                                    Coeff=a$COEFFVALUE,
                                    AdjRSquared=c$ADJRSQUARED,
                                    RSquared=c$RSQUARED,
                                    StdErr=c$STDERR,
                                    FStat=c$FSTAT)
        if(!isDotFormula(object@formula)) rownames(ret)<-
setdiff(all.vars(object@formula),all.vars(object@formula)[1])
        else rownames(ret)<- setdiff(colnames(object@table),all.vars(object@formula)[1])
        return(data.matrix(ret))
}


getReferenceCategories <- function(data,pExcludeCols="",
                                   classSpec=list(),
                                   ...){
    ## browser()
    vcolnames <- colnames(data)
    unused_cols <- c(pExcludeCols,
                     getObsIdSQLExpression(data),
                     getGroupIdSQLExpression(data))

    ## Detect factors and assign classSpec
    vfirstRow <- sqlQuery(getFLConnection(),
                            limitRowsSQL(paste0("SELECT * FROM (",
                                                constructSelect(data),") a "),1))
    vfactorCols <- list()
    ## apply(t,2,function(x){class(x[[1]])}) gives all character
    for(i in setdiff(colnames(vfirstRow),
                     c(unused_cols,names(classSpec),
                     list(...)[["doNotTransform"]],
                     "obs_id_colname",
                     "group_id_colname"))){
        if(length(i)==0) break;
        if(is.factor(vfirstRow[[i]])
            || is.character(vfirstRow[[i]])
            || is.logical(vfirstRow[[i]])){
               r<-as.character(vfirstRow[[i]])
               names(r) <- i
               vfactorCols <- c(vfactorCols,r)
        }
    }
    if(length(vfactorCols)>0){
        if(is.ODBC())
        vrefVars <- sqlQuery(getFLConnection(),
                         paste0("SELECT ",
                             paste0("MIN(",names(vfactorCols),
                                 ") AS ",names(vfactorCols),
                                 collapse=","),
                             " FROM (",constructSelect(data),") a "),
                             as.is=TRUE)
        else vrefVars <- sqlQuery(getFLConnection(),
                         paste0("SELECT ",
                             paste0("MIN(",names(vfactorCols),
                                 ") AS ",names(vfactorCols),
                                 collapse=","),
                             " FROM (",constructSelect(data),") a "))
        vtempList <- list()
        vrefVarNames <- names(vrefVars)
        for(i in colnames(vrefVars)){
            ## Remove variables with NA
            if(is.na(vrefVars[[i]]))
                vrefVarNames <- setdiff(vrefVarNames,
                                        i)
            else if(is.logical(vrefVars[[i]]))
                vtempList <- c(vtempList,
                               levels(sqlQuery(getFLConnection(),
                                           paste0("SELECT DISTINCT(",i,
                                           ") FROM(",constructSelect(data),") a
"))[[1]])[1])
            else vtempList <- c(vtempList,as.character(vrefVars[[i]]))
        }
        names(vtempList) <- vrefVarNames
        return(c(classSpec,vtempList))
    }
    return(classSpec)
}

setMethod("names",signature("FLLinRegr"),
          function(x) c("coefficients",
                        "residuals",
                        "fitted.values",
                        "FLCoeffStdErr",
                        "FLCoeffTStat",
                        "FLCoeffPValue",
                        "FLCoeffNonZeroDensity",
                        "FLCoeffCorrelWithRes",
                        "s",
                        "call",
                        "FLLinRegrStats",
                        "df.residual",
                        "model",
                        "x",
                        "y",
                        "qr",
                        "rank",
                        "terms",
                        "xlevels",
                        "assign",
                        "formula",
                        "anova"
                        ))
```

SVM:

```
function (x, y = NULL, scale = TRUE, type = NULL, kernel = "radial",
    degree = 3, gamma = if (is.vector(x)) 1 else 1/ncol(x), coef0 = 0,
    cost = 1, nu = 0.5, class.weights = NULL, cachesize = 40,
    tolerance = 0.001, epsilon = 0.1, shrinking = TRUE, cross = 0,
    probability = FALSE, fitted = TRUE, ..., subset, na.action = na.omit)
{
    yorig <- y
    if (inherits(x, "Matrix")) {
        loadNamespace("SparseM")
        loadNamespace("Matrix")
        x <- as(x, "matrix.csr")
    }
    if (inherits(x, "simple_triplet_matrix")) {
        loadNamespace("SparseM")
        ind <- order(x$i, x$j)
        x <- new("matrix.csr", ra = x$v[ind], ja = x$j[ind],
            ia = as.integer(cumsum(c(1, tabulate(x$i[ind])))),
            dimension = c(x$nrow, x$ncol))
    }
    if (sparse <- inherits(x, "matrix.csr"))
        loadNamespace("SparseM")
    if (is.null(degree))
        stop(sQuote("degree"), " must not be NULL!")
    if (is.null(gamma))
        stop(sQuote("gamma"), " must not be NULL!")
    if (is.null(coef0))
        stop(sQuote("coef0"), " must not be NULL!")
    if (is.null(cost))
        stop(sQuote("cost"), " must not be NULL!")
    if (is.null(nu))
        stop(sQuote("nu"), " must not be NULL!")
    if (is.null(epsilon))
        stop(sQuote("epsilon"), " must not be NULL!")
    if (is.null(tolerance))
        stop(sQuote("tolerance"), " must not be NULL!")
    xhold <- if (fitted)
        x
    else NULL
    x.scale <- y.scale <- NULL
    formula <- inherits(x, "svm.formula")
    if (is.null(type))
        type <- if (is.null(y))
            "one-classification"
        else if (is.factor(y))
            "C-classification"
        else "eps-regression"
    type <- pmatch(type, c("C-classification", "nu-classification",
        "one-classification", "eps-regression", "nu-regression"),
        99) - 1
    if (type > 10)
        stop("wrong type specification!")
    kernel <- pmatch(kernel, c("linear", "polynomial", "radial",
        "sigmoid"), 99) - 1
    if (kernel > 10)
        stop("wrong kernel specification!")
    nac <- attr(x, "na.action")
    if (sparse) {
        scale <- rep(FALSE, ncol(x))
        if (!is.null(y))
            na.fail(y)
        x <- SparseM::t(SparseM::t(x))
    }
    else {
        x <- as.matrix(x)
        if (!formula) {
            if (!missing(subset)) {
                x <- x[subset, ]
                y <- y[subset]
                if (!is.null(xhold))
                  xhold <- as.matrix(xhold)[subset, ]
            }
            if (is.null(y))
                x <- na.action(x)
            else {
                df <- na.action(data.frame(y, x))
                y <- df[, 1]
                x <- as.matrix(df[, -1], rownames.force = TRUE)
                nac <- attr(x, "na.action") <- attr(y, "na.action") <- attr(d
f,
                  "na.action")
            }
        }
        if (length(scale) == 1)
            scale <- rep(scale, ncol(x))
        if (any(scale)) {
            co <- !apply(x[, scale, drop = FALSE], 2, var)
            if (any(co)) {
                warning(paste("Variable(s)", paste(sQuote(colnames(x[,
                  scale, drop = FALSE])[co]), sep = "", collapse = " and "),
                  "constant. Cannot scale data."))
                scale <- rep(FALSE, ncol(x))
            }
            else {
                xtmp <- scale(x[, scale])
                x[, scale] <- xtmp
                x.scale <- attributes(xtmp)[c("scaled:center",
                  "scaled:scale")]
                if (is.numeric(y) && (type > 2)) {
                  yorig <- y
                  y <- scale(y)
                  y.scale <- attributes(y)[c("scaled:center",
                    "scaled:scale")]
                  y <- as.vector(y)
                }
            }
        }
    }
    nr <- nrow(x)
    if (cross > nr)
        stop(sQuote("cross"), " cannot exceed the number of observations!")
    ytmp <- y
    attributes(ytmp) <- NULL
    if (!is.vector(ytmp) && !is.factor(y) && type != 2)
        stop("y must be a vector or a factor.")
    if (type != 2 && length(y) != nr)
        stop("x and y don't match.")
    if (cachesize < 0.1)
        cachesize <- 0.1
    if (type > 2 && !is.numeric(y))
        stop("Need numeric dependent variable for regression.")
    lev <- NULL
    weightlabels <- NULL
    if (type == 2)
        y <- rep(1, nr)
    else if (is.factor(y)) {
        lev <- levels(y)
        y <- as.integer(y)
    }
    else {
        if (type < 3) {
            if (any(as.integer(y) != y))
                stop("dependent variable has to be of factor or integer type
for classification mode.")
            y <- as.factor(y)
            lev <- levels(y)
            y <- as.integer(y)
        }
        else lev <- unique(y)
    }
    if (type < 3 && !is.null(class.weights)) {
        if (is.character(class.weights) && class.weights == "inverse")
            class.weights <- 1/table(y)
        if (is.null(names(class.weights)))
            stop("Weights have to be specified along with their according lev
el names !")
        weightlabels <- match(names(class.weights), lev)
        if (any(is.na(weightlabels)))
            stop("At least one level name is missing or misspelled.")
    }
    nclass <- 2
    if (type < 2)
        nclass <- length(lev)
    if (type > 1 && length(class.weights) > 0) {
        class.weights <- NULL
        warning(sQuote("class.weights"), " are set to NULL for regression mod
e. For classification, use a _factor_ for ",
            sQuote("y"), ", or specify the correct ", sQuote("type"),
            " argument.")
    }
    err <- empty_string <- paste(rep(" ", 255), collapse = "")
    if (is.null(type))
        stop("type argument must not be NULL!")
    if (is.null(kernel))
        stop("kernel argument must not be NULL!")
    if (is.null(degree))
        stop("degree argument must not be NULL!")
    if (is.null(gamma))
        stop("gamma argument must not be NULL!")
    if (is.null(coef0))
        stop("coef0 seed argument must not be NULL!")
    if (is.null(cost))
        stop("cost argument must not be NULL!")
    if (is.null(nu))
        stop("nu argument must not be NULL!")
    if (is.null(cachesize))
        stop("cachesize argument must not be NULL!")
    if (is.null(tolerance))
        stop("tolerance argument must not be NULL!")
    if (is.null(epsilon))
        stop("epsilon argument must not be NULL!")
    if (is.null(shrinking))
        stop("shrinking argument must not be NULL!")
    if (is.null(cross))
        stop("cross argument must not be NULL!")
    if (is.null(sparse))
        stop("sparse argument must not be NULL!")
    if (is.null(probability))
        stop("probability argument must not be NULL!")
    cret <- .C(R_svmtrain, as.double(if (sparse) x@ra else t(x)),
        as.integer(nr), as.integer(ncol(x)), as.double(y), as.integer(if (spa
rse) x@ia else 0),
        as.integer(if (sparse) x@ja else 0), as.integer(type),
        as.integer(kernel), as.integer(degree), as.double(gamma),
        as.double(coef0), as.double(cost), as.double(nu), as.integer(weightla
bels),
        as.double(class.weights), as.integer(length(class.weights)),
        as.double(cachesize), as.double(tolerance), as.double(epsilon),
        as.integer(shrinking), as.integer(cross), as.integer(sparse),
        as.integer(probability), nclasses = integer(1), nr = integer(1),
        index = integer(nr), labels = integer(nclass), nSV = integer(nclass),
        rho = double(nclass * (nclass - 1)/2), coefs = double(nr *
            (nclass - 1)), sigma = double(1), probA = double(nclass *
            (nclass - 1)/2), probB = double(nclass * (nclass -
            1)/2), cresults = double(cross), ctotal1 = double(1),
        ctotal2 = double(1), error = err)
    if (cret$error != empty_string)
        stop(paste(cret$error, "!", sep = ""))
    cret$index <- cret$index[1:cret$nr]
    ret <- list(call = match.call(), type = type, kernel = kernel,
        cost = cost, degree = degree, gamma = gamma, coef0 = coef0,
        nu = nu, epsilon = epsilon, sparse = sparse, scaled = scale,
        x.scale = x.scale, y.scale = y.scale, nclasses = cret$nclasses,
        levels = lev, tot.nSV = cret$nr, nSV = cret$nSV[1:cret$nclasses],
        labels = cret$labels[1:cret$nclasses], SV = if (sparse) SparseM::t(Sp
arseM::t(x[cret$index])) else t(t(x[cret$index,
            , drop = FALSE])), index = cret$index, rho = cret$rho[1:(cret$ncl
asses *
            (cret$nclasses - 1)/2)], compprob = probability,
        probA = if (!probability) NULL else cret$probA[1:(cret$nclasses *
            (cret$nclasses - 1)/2)], probB = if (!probability) NULL else cret
$probB[1:(cret$nclasses *
            (cret$nclasses - 1)/2)], sigma = if (probability) cret$sigma else
NULL,
        coefs = if (cret$nr == 0) NULL else t(matrix(cret$coefs[1:((cret$ncla
sses -
            1) * cret$nr)], nrow = cret$nclasses - 1, byrow = TRUE)),
        na.action = nac)
    if (cross > 0)
        if (type > 2) {
            scale.factor <- if (any(scale))
                crossprod(y.scale$"scaled:scale")
            else 1
            ret$MSE <- cret$cresults * scale.factor
            ret$tot.MSE <- cret$ctotal1 * scale.factor
            ret$scorrcoeff <- cret$ctotal2
        }
        else {
            ret$accuracies <- cret$cresults
            ret$tot.accuracy <- cret$ctotal1
        }
    class(ret) <- "svm"
    if (fitted) {
        ret$fitted <- na.action(predict(ret, xhold, decision.values = TRUE))
        ret$decision.values <- attr(ret$fitted, "decision.values")
        attr(ret$fitted, "decision.values") <- NULL
        if (type > 1)
            ret$residuals <- yorig - ret$fitted
    }
    ret
}
<bytecode: 0x000000001b900180>
<environment: namespace:e1071>
```