# Package 'FactoRizationMachines'

October 18, 2017

**Type** Package

**Title** Machine Learning with Higher-Order Factorization Machines

**Version** 0.35

**Date** 2017-10-11

**Author** Julian Knoll

**Maintainer** Julian Knoll <julian.knoll@th-nuernberg.de>

**Description** Implementation of different machine learning approaches: Support Vector Machine with a linear kernel, second-order Factorization Machine, higher-order Factorization Machine, and adaptive-order Factorization Machine.



**Imports** Rcpp (>= 0.12.1), methods, Matrix

**LinkingTo** Rcpp

**Suggests** MASS

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2017-10-18 20:34:55 UTC

## R topics documented:

FactoRizationMachines *Machine Learning with Higher-Order Factorization Machines*

## Description

Implementation of three factorization-based machine learning approaches:

- Support Vector Machines (SVM.train) with a linear kernel,

- second-order Factorization Machines [2] (FM.train),

- and higher-order Factorization Machines [1] (HoFM.train),

- and knowledge-extracting or adaptive-order Factorization Machines [3] (KnoFM.train).

Further informations about Factorization Machines are provided by the papers in the references.

## Details

This package includes the following methods:

- SVM.train: Method training a Support Vector Machine,

- FM.train: Method training a second-order Factorization Machine,

- HoFM.train: Method training a higher-order Factorization Machine,

- KnoFM.train: Method training a knowledge-extracting or adaptive-order Factorization Machine,

- predict.FMmodel: Predict Method for FMmodel Objects,

- summary.FMmodel and print.FMmodel: Summary and Print Method for FMmodel Objects.

Two learning methods are supported: coordinate descent (regularization suggested) and Markov Chain Monte Carlo (MCMC). To date the task regression ("r") is supported, the task classifiction ("c") will be supported in the future.

## Author(s)

Maintainer: Julian Knoll <julian.knoll@th-nuernberg.de>

## References

[1] J. Knoll, Recommending with Higer-Order Factorization Machines, Research and Development in Intelligent Systems XXXIII, 2016.

[2] S. Rendle, Factorization Machines with libFM, ACM Transactions on Intelligent Systems and Technology, 3, 2012.

[3] J. Knoll , J. Stuebinger, and M. Grottke, Exploiting social media with higher-order Factorization Machines: Statistical arbitrage on high-frequency data of the S&P 500. FAU Discussion Papers in Economics, University of Erlangen-Nuernberg, 2017.

## See Also

SVM.train, FM.train, HoFM.train, predict.FMmodel

## Examples

```
## Not run:

# Load libraries
library(FactoRizationMachines)
library(Matrix)

# Load MovieLens 100k data set
ml100k=as.matrix(read.table("http://files.grouplens.org/datasets/movielens/ml-100k/u.data"))
user=ml100k[,1]
items=ml100k[,2]+max(user)
wdays=(as.POSIXlt(ml100k[,4],origin="1970-01-01")$wday+1)+max(items)

# Transform MovieLens 100k to feature form
data=sparseMatrix(i=rep(1:nrow(ml100k),3),j=c(user,items,wdays),giveCsparse=F)
target=ml100k[,3]

# Subset data to training and test data
set.seed(123)
subset=sample.int(nrow(data),nrow(data)*.8)
data.train=data[subset,]
data.test=data[-subset,]
target.train=target[subset]
target.test=target[-subset]

# Predict ratings with Support Vector Machine with linear kernel
# using MCMC learning method
model=SVM.train(data.train,target.train)
# RMSE resulting from test data prediction
sqrt(mean((predict(model,data.test)-target.test)^2))

# Predict ratings with second-order Factorization Machine
# with second-order 10 factors (default)
# using coordinate descent learning method (regularization suggested)
model=FM.train(data.train,target.train,regular=0.1)
# RMSE resulting from test data prediction
sqrt(mean((predict(model,data.test)-target.test)^2))

# Predict ratings with second-order Factorization Machine
# with second-order 10 factors (default)
# using Markov Chain Monte Carlo learning method
model=FM.train(data.train,target.train)
# RMSE resulting from test data prediction
sqrt(mean((predict(model,data.test)-target.test)^2))

# Predict ratings with higher-order Factorization Machine
# with 3 second-order and 1 third-order factor and regularization
# using coordinate descent learning method (regularization suggested)
model=HoFM.train(data.train,target.train,c(1,3,1),regular=0.1)
# RMSE resulting from test data prediction
sqrt(mean((predict(model,data.test)-target.test)^2))
```

```
# Predict ratings with higher-order Factorization Machine
# with 3 second-order and 1 third-order factor and regularization
# using MCMC learning method
model=HoFM.train(data.train,target.train,c(1,3,1))
# RMSE resulting from test data prediction
sqrt(mean((predict(model,data.test)-target.test)^2))

# Predict ratings with adaptive-order Factorization Machine
model=KnoFM.train(data.train,target.train)
# RMSE resulting from test data prediction
sqrt(mean((predict(model,data.test)-target.test)^2))


## End(Not run)
```

---

`FM.train` *Method training a second-order Factorization Machine*

---

## Description

`FM.train` is a method training a second-order Factorization Machine.

`factors` specifies the model parameters of the Factorization Machine: the first element specifies whether linear weights are used (`1`) or not (`0`), the second element specifies the number of parameters factorizing the second-order.

## Usage

```
FM.train(data, target, factors = c(1, 10), intercept = T,
iter = 100, regular = NULL, stdev = 0.1)
```

## Arguments

| | |
|---|---|
| `data` | an object of class `dgTMatrix`, `matrix` or `data.frame` (or an object coercible to `dgTMatrix`): a matrix containing training data, each row representing a training example and each column representing a feature. |
| `target` | `numeric`: vector specifying the target value of each training example (length must match rows of object data). |
| `factors` | `numeric`: vector specifying the number of factors for each considered order: the first element specifies whether linear weights are used (`1`) or not (`0`), the second element specifies the number of parameters factorizing the second-order. |
| `intercept` | `logical`: specifying whether a global intercept is used (`TRUE`) or not (`FALSE`). |
| `iter` | `integer`: the number of iterations the learning method is applied. |
| `regular` | `numeric`: regularization value for each order corresponding to factors. If one value, each order is regularized with this value, otherwise the first element of the vector specifies the regularization value for the linear weights and the second the regularization value for the second-order factors. If regular is NULL, automatic regularization using Markov Chain Monte Carlo (MCMC) method is applied. |
| `stdev` | `numeric`: the standard deviation used to initialize the model parameters. |

## References

[1] J. Knoll, Recommending with Higer-Order Factorization Machines, Research and Development in Intelligent Systems XXXIII, 2016.

[2] S. Rendle, Factorization Machines with libFM, ACM Transactions onIntelligent Systems and Technology (TIST), 3, 2012.

## See Also

FactoRizationMachines

## Examples

```
## Not run:

### Example to illustrate the usage of the method
### Data set very small and not sparse, results not representative
### Please study major example in general help 'FactoRizationMachines'

# Load data set
library(FactoRizationMachines)
library(MASS)
data("Boston")

# Subset data to training and test data
set.seed(123)
subset=sample.int(nrow(Boston),nrow(trees)*.8)
data.train=Boston[subset,-ncol(Boston)]
target.train=Boston[subset,ncol(Boston)]
data.test=Boston[-subset,-ncol(Boston)]
target.test=Boston[-subset,ncol(Boston)]


# Predict with 3 second-order factors with MCMC regularization
model=FM.train(data.train,target.train,c(1,3))

# RMSE resulting from test data prediction
sqrt(mean((predict(model,data.test)-target.test)^2))


# Predict with 10 second-order factor with MCMC regularization
model=FM.train(data.train,target.train)

# RMSE resulting from test data prediction
sqrt(mean((predict(model,data.test)-target.test)^2))


# Predict with 10 second-order factor with manual regularization
model=FM.train(data.train,target.train,regular=0.1)

# RMSE resulting from test data prediction
sqrt(mean((predict(model,data.test)-target.test)^2))
```

```
## End(Not run)
```

---

HoFM.train *Method training a higher-order Factorization Machine*

---

## Description

HoFM.train is a method training a higher-order Factorization Machine.

factors specifies the model parameters of the Factorization Machine: the first element specifies whether linear weights are used (1) or not (0), the second element specifies the number of parameters factorizing the second-order, the third element specifies the number of parameters factorizing the third-order, ..., up to the tenth element specifying the number of parameters factorizing the tenth-order.

## Usage

```
HoFM.train(data, target, factors = c(1, 10, 5), intercept = T,
iter = 100, regular = NULL, stdev = 0.1)
```

## Arguments

| | |
|---|---|
| data | an object of class dgTMatrix, matrix or data.frame (or an object coercible to dgTMatrix): a matrix containing training data, each row representing a training example and each column representing a feature. |
| target | numeric: vector specifying the target value of each training example (length must match rows of object data). |
| factors | numeric: vector specifying the number of factors for each considered order: the first element specifies whether linear weights are used (1) or not (0), the second element specifies the number of parameters factorizing the second-order, the third element specifies the number of parameters factorizing the third-order, ..., up to the tenth element specifying the number of parameters factorizing the tenth-order. |
| intercept | logical: specifying whether a global intercept is used (TRUE) or not (FALSE). |
| iter | integer: the number of iterations the learning method is applied. |
| regular | numeric: regularization value for each order corresponding to factors. If one value, each order is regularized with this value, otherwise each element of the vector specifies the regularization value of the corresponding order. If regular is NULL, automatic regularization using Markov Chain Monte Carlo (MCMC) method is applied. |
| stdev | numeric: the standard deviation used to initialize the model parameters. |

## References

[1] J. Knoll, Recommending with Higer-Order Factorization Machines, Research and Development in Intelligent Systems XXXIII, 2016.

[2] S. Rendle, Factorization Machines with libFM, ACM Transactions onIntelligent Systems and Technology (TIST), 3, 2012.

## See Also

FactoRizationMachines

## Examples

```
## Not run:

### Example to illustrate the usage of the method
### Data set very small and not sparse, results not representative
### Please study major example in general help 'FactoRizationMachines'

# Load data set
library(FactoRizationMachines)
library(MASS)
data("Boston")

# Subset data to training and test data
set.seed(123)
subset=sample.int(nrow(Boston),nrow(trees)*.8)
data.train=Boston[subset,-ncol(Boston)]
target.train=Boston[subset,ncol(Boston)]
data.test=Boston[-subset,-ncol(Boston)]
target.test=Boston[-subset,ncol(Boston)]


# Predict with 7 second-order and 2 third-order factors
#  with MCMC regularization
model=HoFM.train(data.train,target.train,c(1,7,2))

# RMSE resulting from test data prediction
sqrt(mean((predict(model,data.test)-target.test)^2))


# Predict with 10 second-order and 5 third-order factor
#  with MCMC regularization
model=HoFM.train(data.train,target.train)

# RMSE resulting from test data prediction
sqrt(mean((predict(model,data.test)-target.test)^2))


# Predict with 10 second-order and 5 third-order factor
#  with manual regularization
model=HoFM.train(data.train,target.train,regular=0.1)
```

```
# RMSE resulting from test data prediction
sqrt(mean((predict(model,data.test)-target.test)^2))


## End(Not run)
```

---

KnoFM.train *Knowledge-extracting or adaptive-order Factorization Machine*

---

## Description

KnoFM.train is a method training a knowledge-extracting Factorization Machine.

## Usage

```
KnoFM.train(data, target, multicore = T, silent = F)
```

## Arguments

| | |
|---|---|
| data | an object of class dgTMatrix, matrix or data.frame (or an object coercible to dgTMatrix): a matrix containing training data, each row representing a training example and each column representing a feature. |
| target | numeric: vector specifying the target value of each training example (length must match rows of object data). |
| multicore | logical: specifying whether multiple cores should be used. |
| silent | logical: specifying whether progess should be printed. |

## References

[1] J. Knoll , J. Stuebinger, and M. Grottke, Exploiting social media with higher-order Factorization Machines: Statistical arbitrage on high-frequency data of the S&P 500. FAU Discussion Papers in Economics, University of Erlangen-Nuernberg, 2017.

[2] J. Knoll, Recommending with Higer-Order Factorization Machines, Research and Development in Intelligent Systems XXXIII, 2016.

## See Also

FactoRizationMachines

## Examples

```
## Not run:

### Example to illustrate the usage of the method
### Data set very small and not sparse, results not representative
### Please study major example in general help 'FactoRizationMachines'

# Load data set
library(FactoRizationMachines)
library(MASS)
data("Boston")

# Subset data to training and test data
set.seed(123)
subset=sample.int(nrow(Boston),nrow(trees)*.8)
data.train=Boston[subset,-ncol(Boston)]
target.train=Boston[subset,ncol(Boston)]
data.test=Boston[-subset,-ncol(Boston)]
target.test=Boston[-subset,ncol(Boston)]

# Predict with an adaptive-order Factorization Machine
# using one CPU core  and printing progress
model=KnoFM.train(data.train,target.train,FALSE,FALSE)

# RMSE resulting from test data prediction
sqrt(mean((predict(model,data.test)-target.test)^2))



## End(Not run)
```

---

`predict.FMmodel` *Predict Method for FMmodel Objects*

---

## Description

Function for predicting new data based on a FMmodel object

## Usage

```
   ## S3 method for class 'FMmodel'
predict(object, newdata, truncate = T, ...)
```

## Arguments

| | |
|---|---|
| `object` | a FMmodel object (output of `SVM.train`, `FM.train`, or `HoFM.train`) |
| `newdata` | new data for prediction based on the FMmodel object (number of features must match the features of the training data) |
| `truncate` | bool indicating whether the output should be trunceted (`T`) order not (`F`) |
| `...` | additional arguments |

## See Also

SVM.train, FM.train, HoFM.train

## Examples

```
### Example to illustrate the usage of the method
### Data set very small and not sparse, results not representative
### Please study major example in general help 'FactoRizationMachines'

# Load data set
library(FactoRizationMachines)
library(MASS)
data("Boston")

# Subset data to training and test data
set.seed(123)
subset=sample.int(nrow(Boston),nrow(trees)*.8)
data.train=Boston[subset,-ncol(Boston)]
target.train=Boston[subset,ncol(Boston)]
data.test=Boston[-subset,-ncol(Boston)]
target.test=Boston[-subset,ncol(Boston)]


# Predict with 10 second-order and 5 third-order factor
model=HoFM.train(data.train,target.train)

# RMSE resulting from test data prediction
sqrt(mean((predict(model,data.test)-target.test)^2))
```

---

# summary.FMmodel *Summary and Print Method for FMmodel Objects*

---

## Description

Function generating the summary of a FMmodel object.

## Usage

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

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

### Arguments

| | |
|---|---|
| object | a FMmodel object (output of SVM.train, FM.train, or HoFM.train) |
| x | a FMmodel object (output of SVM.train, FM.train, or HoFM.train) |
| ... | additional arguments |

### Details

The summary contains for instance:

- the number of training examples the model was build with,

- the number of variables (features) the model considers,

- the minimum value of the target vector elements (to truncate the prediction),

- the maximum value of the target vector elements (to truncate the prediction),

- the number of factors for each considered order: the first element specifies whether linear weights are used (1) or not (0), the second element specifies the number of parameters factorizing the second-order, the third element specifies the number of parameters factorizing the third-order.

### See Also

SVM.train, FM.train, HoFM.train

---

## SVM.train *Method training a Support Vector Machine*

---

### Description

SVM.train is a method training a Support Vector Machine with a linear kernel.

factors specifies whether linear weights are used (1) or not (0). If linear weights are not used intercept is set to TRUE.

### Usage

```
SVM.train(data, target, factors = 1, intercept = T,
iter = 100, regular = NULL, stdev = 0.1)
```

### Arguments

| | |
|---|---|
| data | an object of class dgTMatrix, matrix or data.frame (or an object coercible to dgTMatrix): a matrix containing training data, each row representing a training example and each column representing a feature. |
| target | numeric: vector specifying the target value of each training example (length must match rows of object data). |
| factors | either 0 or 1: specifying whether linear weights are used (1) or not (0). If linear weights are not used intercept is set to TRUE. |
| intercept | logical: specifying whether a global intercept is used (TRUE) or not (FALSE). |

| | |
|---|---|
| iter | integer: the number of iterations the learning method is applied. |
| regular | numeric: regularization value for the linear weights. If regular is NULL, automatic regularization using Markov Chain Monte Carlo (MCMC) method is applied. |
| stdev | numeric: the standard deviation used to initialize the model parameters. |

## See Also

FactoRizationMachines

## Examples

```
## Not run:

### Example to illustrate the usage of the method
### Data set very small and not sparse, results not representative
### Please study major example in general help 'FactoRizationMachines'

# Load data set
library(FactoRizationMachines)
library(MASS)
data("Boston")

# Subset data to training and test data
set.seed(123)
subset=sample.int(nrow(Boston),nrow(trees)*.8)
data.train=Boston[subset,-ncol(Boston)]
target.train=Boston[subset,ncol(Boston)]
data.test=Boston[-subset,-ncol(Boston)]
target.test=Boston[-subset,ncol(Boston)]


# Predict with linear weights and intercept with MCMC regularization
model=SVM.train(data.train,target.train)

# RMSE resulting from test data prediction
sqrt(mean((predict(model,data.test)-target.test)^2))


# Predict with linear weights but without intercept with MCMC regularization
model=SVM.train(data.train,target.train,intercept=FALSE)

# RMSE resulting from test data prediction
sqrt(mean((predict(model,data.test)-target.test)^2))


# Predict with linear weights and manual regularization
model=SVM.train(data.train,target.train,regular=0.1)

# RMSE resulting from test data prediction
sqrt(mean((predict(model,data.test)-target.test)^2))
```

```
## End(Not run)
```

# Index