Package: FeatureHashing (via r-universe)

January 19, 2025

Type Package

Title Creates a Model Matrix via Feature Hashing with a Formula Interface

Version 0.9.2

Date 2024-01-10

Maintainer Wush Wu <wush978@gmail.com>

Description Feature hashing, also called as the hashing trick, is a method to transform features of a instance to a vector. Thus, it is a method to transform a real dataset to a matrix. Without looking up the indices in an associative array, it applies a hash function to the features and uses their hash values as indices directly. The method of feature hashing in this package was proposed in Weinberger et al. (2009) <arXiv:0902.2206>. The hashing algorithm is the murmurhash3 from the 'digest' package. Please see the README in <https://github.com/wush978/FeatureHashing> for more information.

License GPL (>= 3) | file LICENSE

Depends R (>= 4.0), methods

Imports Rcpp (>= 0.11), Matrix, digest(>= 0.6.8), magrittr (>= 1.5)

LinkingTo Rcpp, digest(>= 0.6.8), BH(>= 1.54.0-1)

Suggests RUnit, glmnet, knitr, xgboost, rmarkdown, pROC

BugReports https://github.com/wush978/FeatureHashing/issues

URL https://github.com/wush978/FeatureHashing

VignetteBuilder knitr

RoxygenNote 7.2.3

NeedsCompilation yes

Author Wush Wu [aut, cre], Michael Benesty [aut, ctb]

Date/Publication 2024-01-10 15:33:03 UTC

Additional_repositories https://cranhaven.r-universe.dev

Repository https://cranhaven.r-universe.dev

RemoteUrl https://github.com/cranhaven/cranhaven.r-universe.dev

RemoteRef package/FeatureHashing

RemoteSha ea9524c273437ca61a2f17810f371a9d9905921e

RemoteSubdir FeatureHashing

Contents

`CSCMatrix-class` *CSCMatrix*

Description

The structure of `CSCMatrix` is the same as the structure of `dgCMatrix`. However, the `CSCMatrix` has weaker constraints compared to `dgCMatrix`.

`CSCMatrix` onlysupports limited operators. The users can convert it to `dgCMatrix` for compatibility of existed algorithms.

Details

The `CSCMatrix` violates two constraints used in `dgCMatrix`:

- The row indices should be sorted with columns.
- The row indices should be unique with columns.

The result of matrix-vector multiplication should be the same.

Methods

- `dim` The dimension of the matrix object `CSCMatrix`.
- `dim<-` The assignment of dimension of the matrix object `CSCMatrix`.
- `[` The subsetting operator of the matrix object `CSCMatrix`.
- `%*%` The matrix-vector multiplication of the matrix object `CSCMatrix`. The returned object is a numeric vector.

See Also

dgCMatrix-class

Examples

```
# construct a CSCMatrix
m <- hashed.model.matrix(~ ., CO2, 8)
# convert it to dgCMatrix
m2 <- as(m, "dgCMatrix")
```

`hash.mapping` *Extract mapping between hash and original values*

Description

Extract mapping between hash and original values

Usage

```
hash.mapping(matrix)
```

Arguments

`matrix`	Matrix returned by `hashed.model.matrix` function

Details

Generate a mapping between original values and hashes.

Option `create.mapping = T` needs to be used in function `hashed.model.matrix`.

Original values are stores in the names of the vector.

Value

a named `numeric` vector

Author(s)

Michael Benesty

Examples

```
data(ipinyou)

m <- hashed.model.matrix(~., ipinyou.train, 2^10, create.mapping = TRUE)
mapping <- hash.mapping(m)
```

hash.size *Compute minimum hash size to reduce collision rate*

Description

Compute minimum hash size to reduce collision rate

Usage

```
hash.size(df)
```

Arguments

df	data.frame with data to hash

Details

To reduce collision rate, the hash size should be equal or superior to the nearest power of two to the number of unique values in the input data.frame.

The value computed is a theorical minimum hash size. It just means that in the best situation it may be possible that all computed hash can be stored with this hash size.

Intuitively, if the distribution of hash generated by the algorithm was perfect, when the computed size is used, each permutation of bits of the hash vector would correspond to one unique original value of your data.frame.

Because a bit value is {0,1}, the computed size is 2^x with a x big enough to have a hash vector containing each original value.

In real life, there will be some collisions if the computed size is used because the distribution of hash is not perfect. However, the hashing algorithm Murmur3 is known to minimize the number of collisions and is also very performant.

The only known solution to have zero collision is to build a dictionnary of values, and for each new value to hash, check in the dictionnary if the hash value already exists. It is not performant at all.

If you increase the computed size (by multiplying it by 2^x, it is up to you to choose a x), you will reduce the collision rate. If you use a value under the computed size, there is a 100

There is a trade-off between collision rate and memory used to store hash. Machine learning algorithms usually deal well with collisions when the rate is reasonable.

Value

The hash size of feature hashing as a positive integer.

Author(s)

Michael Benesty

Examples

```
data(ipinyou)

#First try with a size of 2^10
mat1 <- hashed.model.matrix(~., ipinyou.train, 2^10, create.mapping = TRUE)

#Extract mapping
mapping1 <- hash.mapping(mat1)
#Rate of collision
mean(duplicated(mapping1))

#Second try, the size is computed
size <- hash.size(ipinyou.train)
mat2 <- hashed.model.matrix(~., ipinyou.train, size, create.mapping = TRUE)

#Extract mapping
mapping2 <- hash.mapping(mat2)
#Rate of collision
mean(duplicated(mapping2))
```

---

hashed.model.matrix *Create a model matrix with feature hashing*

Description

Create a model matrix with feature hashing

Usage

```
hashed.model.matrix(
  formula,
  data,
  hash.size = 2^18,
  transpose = FALSE,
  create.mapping = FALSE,
  is.dgCMatrix = TRUE,
  signed.hash = FALSE,
  progress = FALSE
)
```

Arguments

formula	formula or a character vector of column names (will be expanded to a formula)
data	data.frame. The original data.
hash.size	positive integer. The hash size of feature hashing.
transpose	logical value. Indicating if the transpose should be returned. It affects the space of the returned object when the dimension is imbalanced. Please see the details.

create.mapping	logical value. The indicator of whether storing the hash mapping or not. The mapping might miss some interaction terms which involves splited features. Please see the details.
is.dgCMatrix	logical value. Indicating if the result is dgCMatrix or CSCMatrix
signed.hash	logical value. Indicating if the hashed value is multipled by random sign. This will reduce the impact of collision. Disable it will enhance the speed.
progress	logical value. Indicating if the progress bar is displayed or not.

Details

The hashed.model.matrix hashes the feature during the construction of the model matrix. It uses the 32-bit variant of MurmurHash3 https://github.com/aappleby/smhasher. Weinberger et. al. (2009) used two separate hashing function h(hashed.value) and ξ(hash.sign) to determine the indices and the sign of the values respectively. Different seeds are used to implement the hashing function h and ξ with MurmurHash3.

The formula is parsed via terms.formula with "split" as special keyword. The interaction term is hashed (the reader can try to expl)in different ways. Please see example for the detailed implementation. We provide a helper function: hashed.interaction.value to show show the index after interaction. The "split" is used to expand the concatenated feature such as "10129,10024,13866,10111,10146,10120,10115,10 which represents the occurrence of multiple categorical variable: "10129", "10024", "13866", "10111", "10146", "10120", "10115", and "10063". The hashed.model.matrix will expand the concatenated feature and produce the related model matrix.

The "split" accepts two parameters:

- delim, character value to use as delimiter for splitting;
- type, one of existence, count or tf-idf.

If type is set to tf-idf, then signed.hash should be set to FALSE.

The user could explore the behavior via function simulate.split.

The argument transpose affects the size of the returned object in the following way. For a $m \times n$ matrix with k non-zero elements, the returned dgCMatrix requires $O(n) + O(k)$ space. For details, please check the documentation of the dgCMatrix-class. Note that the rownames of the returned dgCMatrix is character(0) so the space complexity does not contain the term $O(m)$.

The mapping created by enabling create.mapping might miss the interaction term which involves splited features. For example, suppose there are two columns a and b while the value are 1 and 1,2,3 respectively. The user marks the column b with split. If the hashed value of b1 and b2 are collided, then the interaction a1:b1 will not appear in the returned mapping table. Because this package is originally designed for predictive analysis and the mapping should not play an important role of predictive analysis. If you have a test case and want to ask us to fix this, please provide us a test case in https://github.com/wush978/FeatureHashing/issues/67.

References

H. B. McMahan, G. Holt, D. Sculley, et al. "Ad click prediction: a view from the trenches". In: _The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013_. Ed. by I. S. Dhillon, Y. Koren, R. Ghani, T. E. Senator, P. Bradley, R. Parekh, J. He, R. L. Grossman and R. Uthurusamy. ACM, 2013, pp.

1222-1230. DOI: 10.1145/2487575.2488200. <URL: https://doi.acm.org/10.1145/2487575.2488200>.

Kilian Q. Weinberger, Anirban Dasgupta, John Langford, Alexander J. Smola, and Josh Attenberg. ICML, volume 382 of ACM International Conference Proceeding Series, page 140. ACM, (2009)

W. Zhang, S. Yuan, J. Wang, et al. "Real-Time Bidding Benchmarking with iPinYou Dataset". In: _arXiv preprint arXiv:1407.7073_ (2014).

Examples

```
# The following scripts show how to fit a logistic regression
# after feature hashing
## Not run:
data(ipinyou)
f <- ~ IP + Region + City + AdExchange + Domain +
 URL + AdSlotId + AdSlotWidth + AdSlotHeight +
 AdSlotVisibility + AdSlotFormat + CreativeID +
 Adid + split(UserTag, delim = ",")
# if the version of FeatureHashing is 0.8, please use the following command:
# m.train <- as(hashed.model.matrix(f, ipinyou.train, 2^16, transpose = FALSE), "dgCMatrix")
m.train <- hashed.model.matrix(f, ipinyou.train, 2^16)
m.test <- hashed.model.matrix(f, ipinyou.test, 2^16)

# logistic regression with glmnet

library(glmnet)

cv.g.lr <- cv.glmnet(m.train, ipinyou.train$IsClick,
 family = "binomial")#, type.measure = "auc")
p.lr <- predict(cv.g.lr, m.test, s="lambda.min")
auc(ipinyou.test$IsClick, p.lr)

## Per-Coordinate FTRL-Proximal with $L_1$ and $L_2$ Regularization for Logistic Regression

# The following scripts use an implementation of the FTRL-Proximal for Logistic Regresion,
# which is published in McMahan, Holt and Sculley et al. (2013), to predict the probability
# (1-step prediction) and update the model simultaneously.


source(system.file("ftprl.R", package = "FeatureHashing"))
m.train <- hashed.model.matrix(f, ipinyou.train, 2^16, transpose = TRUE)
ftprl <- initialize.ftprl(0.1, 1, 0.1, 0.1, 2^16)
ftprl <- update.ftprl(ftprl, m.train, ipinyou.train$IsClick, predict = TRUE)
auc(ipinyou.train$IsClick, attr(ftprl, "predict"))

# If we use the same algorithm to predict the click through rate of the 3rd season of iPinYou,
# the overall AUC will be 0.77 which is comparable to the overall AUC of the
# 3rd season 0.76 reported in Zhang, Yuan, Wang, et al. (2014).

## End(Not run)

# The following scripts show the implementation of the FeatureHashing.
```

```
# Below the original values will be project in a space of 2^6 dimensions
m <- hashed.model.matrix(~ ., CO2, 2^6, create.mapping = TRUE,
 transpose = TRUE, is.dgCMatrix = FALSE)

# Print the matrix via dgCMatrix
as(m, "dgCMatrix")

# Extraction of the dictionary: values with their hash
mapping <- hash.mapping(m)

# To check the rate of collisions, we will extract the indices of the hash
# values through the modulo-division method, count how many duplicates
# we have (in best case it should be zero) and perform a mean.
mean(duplicated(mapping))

# The type of the result produced by the function `hashed.model.matrix`
# is a CSCMatrix. It supports simple subsetting
# and matrix-vector multiplication
rnorm(2^6) %*% m

# Detail of the hashing
# To hash one specific value, we can use the `hashed.value` function
# Below we will apply this function to the feature names
vectHash <- hashed.value(names(mapping))

# Now we will check that the result is the same than the one got with
# the more generation `hashed.model.matrix` function.
# We will use the Modulo-division method (that's the [%% 2^6] below)
# to find the address in hash table easily.
stopifnot(all(vectHash %% 2^6 + 1 == mapping))

# The sign is corrected by `hash.sign`
hash.sign(names(mapping))

## The interaction term is implemented as follow:
m2 <- hashed.model.matrix(~ .^2, CO2, 2^6, create.mapping = TRUE,
 transpose = TRUE, is.dgCMatrix = FALSE)
# The ^ operator indicates crossing to the specified degree.
# For example (a+b+c)^2 is identical to (a+b+c)*(a+b+c)
# which in turn expands to a formula containing the main effects
# for a, b and c together with their second-order interactions.

# Extract the mapping
mapping2 <- hash.mapping(m2)

# Get the hash of combination of two items, PlantQn2 and uptake
mapping2["PlantQn2:uptake"]

# Extract hash of each item
h1 <- hashed.value("PlantQn2")
h2 <- hashed.value("uptake")

# Computation of hash of both items combined
```

```
h3 <- hashed.value(rawToChar(c(intToRaw(h1), intToRaw(h2))))
stopifnot(h3 %% 2^6 + 1 == mapping2["PlantQn2:uptake"])

# The concatenated feature, i.e. the array<string> type in hive
data(test.tag)
df <- data.frame(a = test.tag, b = rnorm(length(test.tag)))
m <- hashed.model.matrix(~ split(a, delim = ",", type = "existence"):b, df, 2^6,
 create.mapping = TRUE)
# The column `a` is splitted by "," and have an interaction with "b":
mapping <- hash.mapping(m)
names(mapping)
```

`intToRaw` *Convert the integer to raw vector with endian correction*

Description

Convert the integer to raw vector with endian correction

Usage

```
intToRaw(src)
```

Arguments

`src`	integer value.

Value

raw vector with length 4

`ipinyou` *iPinYou Real-Time Bidding Dataset for Computational Advertising Research*

Description

This is a sample from the iPinYou Real-Time Bidding dataset. The data.frame named `ipinyou.train` is a sample from the data of 2013-10-19 and the data.frame named `ipinyou.test` is a sample from the data of 2013-10-20.

Usage

```
data(ipinyou)
```

Format

The column name of the data is the description of the data in Zhang, Yuan, Wang, et al. (2014). Most of the columns should be clearly described by their column names. For the details of the dataset, please read the Zhang, Yuan, Wang, et al. (2014).

`BidID`, the id of the RTB which is the unique identifier of the events.

`Adid`, the advertiser id.

`UserTag`, the user tags (segments) in iPinYou's proprietary audience database. This is also a real example of the concatenated feature.

Source

http://data.computational-advertising.org/

References

W. Zhang, S. Yuan, J. Wang, et al. "Real-Time Bidding Benchmarking with iPinYou Dataset". In: arXiv preprint arXiv:1407.7073 (2014).

`simulate.split` *Simulate how* `split` *work in* `hashed.model.matrix` *to split the string into tokens*

Description

Simulate how `split` work in `hashed.model.matrix` to split the string into tokens

Usage

```
simulate.split(x, delim = ",", type = c("existence", "count"))
```

Arguments

`x`	character vector or factor. The source of concatenated feature.
`delim`	character value. The string to use for splitting.
`type`	character value. Either `"count"` or `"existence"`. `"count"` indicates the number of occurrence of the token. `"existence"` indicates the boolean that whether the token exist or not.

Value

integer vector for `type = "count"` and logical vector for `type = "existence"`.

test.tag *test.tag*

Description

This is a vector to demo the concatenated feature.

Format

For each element, the string represents the occurrence of different tags. For example, the string "1,27,19,25,tp,tw" of the first instance represents that the feature ‘1‘ is TRUE, the feature ‘27‘ is TRUE, et. al. On the contrary, the missing feature such as ‘2‘ is FALSE.

Index