Package 'SAR'

May 7, 2026

Title Smart Adaptive Recommendations

Version 1.0.4

Description 'Smart Adaptive Recommendations' (SAR) is the name of a fast, scalable, adaptive algorithm for personalized recommendations based on user transactions and item descriptions. It produces easily explainable/interpretable recommendations and handles ``cold item'' and ``semi-cold user'' scenarios. This package provides two implementations of 'SAR': a standalone implementation, and an interface to a web service in Microsoft's 'Azure' cloud: <https://github.com/Microsoft/Product-Recommendations/blob/master/doc/sar.md>. The former allows fast and easy experimentation, and the latter provides robust scalability and extra features for production use.

URL https://github.com/hongooi73/SAR

BugReports https://github.com/hongooi73/SAR/issues

License MIT + file LICENSE

Depends R (>= 3.3)

Imports AzureRMR, AzureStor, dplyr (>= 0.7), httr, jsonlite, Matrix, R6, parallel, Rcpp (>= 0.12), RcppParallel

Suggests testthat

LinkingTo Rcpp, RcppArmadillo, RcppParallel

SystemRequirements GNU make

LazyData true

RoxygenNote 7.3.1

NeedsCompilation yes

Author Hong Ooi [aut, cre],
Microsoft Product Recommendations team [ctb] (source for MS sample datasets),
Microsoft [cph]

Maintainer Hong Ooi <hongooi73@gmail.com>

Repository CRAN

Date/Publication 2025-04-12 23:30:05 UTC

Contents

`az_rec_service` *Azure product recommendations service class*

Description

Class representing an Azure product recommendations service.

Format

An R6 object of class `az_rec_service`, inheriting from `AzureRMR::az_template`.

Methods

- `new(token, subscription, resource_group, name, ...)`: Initialize a recommendations service object. See 'Initialization' for more details.
- `start()`: Start the service.
- `stop()`: Stop the service.
- `get_rec_endpoint()`: Return an object representing the client endpoint for the service.
- `set_data_container(data_container="inputdata")`: sets the name of the blob container to use for storing datasets.
- `delete(confirm=TRUE)`: Delete the service, after checking for confirmation.

Initialization

Generally, the easiest way to initialize a new recommendations service object is via the `create_rec_service` or `get_rec_service` methods of the AzureRMR::az_subscription or AzureRMR::az_resource_group classes.

To create a new recommendations service, supply the following additional arguments to `new()`:

- `hosting_plan`: The name of the hosting plan (essentially the size of the virtual machine on which to run the service). See below for the plans that are available.

- `storage_type`: The type of storage account to use. Can be `"Standard_LRS"` or `"Standard_GRS"`.
- `insights_location`: The location for the application insights service. Defaults to `"East US"`.
- `data_container`: The default blob storage container to use for saving input datasets. Defaults to `"inputdata"`.
- `wait`: Whether to wait until the service has finished provisioning. Defaults to TRUE.

See Also

rec_endpoint, for the client interface to the recommendations service

List of Azure hosting plans

Deployment instructions at the Product Recommendations API repo on GitHub

Examples

```
## Not run:

# recommended way of retrieving a resource: via a resource group object
svc <- resgroup$get_rec_service("myrec")

# start the service backend
svc$start()

# get the service endpoint
rec_endp <- svc$get_rec_endpoint()


## End(Not run)
```

`create_rec_service` *Create Azure recommender service*

Description

Method for the AzureRMR::az_resource_group and AzureRMR::az_subscription classes.

Usage

```
## R6 method for class 'az_subscription'
create_rec_service(name, location, hosting_plan, storage_type = c("Standard_LRS", "Standard_GRS"),
             insights_location = c("East US", "North Europe", "West Europe", "South Central US"),
                   data_container = "inputdata", ..., wait = TRUE

## R6 method for class 'az_resource_group'
create_rec_service(name, hosting_plan, storage_type = c("Standard_LRS", "Standard_GRS"),
             insights_location = c("East US", "North Europe", "West Europe", "South Central US"),
                   data_container = "inputdata", ..., wait = TRUE
```

Arguments

- `name`: The name of the recommender service.
- `location`: For the subscription method, the location/region for the service. For the resource group method, this is taken from the location of the resource group.
- `storage_type`: The replication strategy for the storage account for the service.
- `insights_location`: Location for the application insights service giving you details on the webapp usage.
- `data_container`: The name of the blob container within the storage account to use for storing datasets.
- `wait`: Whether to wait until the service has finished provisioning.
- `...` : Other named arguments to pass to the AzureRMR::az_template initialization function.

Details

This method deploys a new recommender service. The individual resources created are an Azure webapp, a storage account, and an application insights service for monitoring. Within the storage account, a blob container is created with name given by the `data_container` argument for storing input datasets.

For the az_subscription method, a resource group is also created to hold the resources. The name of the resource group will be the same as the name of the service.

Value

An object of class `az_rec_service` representing the deployed recommender service.

See Also

get_rec_service, delete_rec_service.

The architecture for the web service is documented here, and the specific template deployed by this method is here.

Examples

```
## Not run:

rg <- AzureRMR::az_rm$
    new(tenant="myaadtenant.onmicrosoft.com", app="app_id", password="password")$
    get_subscription("subscription_id")$
    get_resource_group("rgname")

# create a new recommender service
rg$create_rec_service("myrec", hosting_plan="S2")


## End(Not run)
```

`delete_rec_service` *Delete an Azure recommender service*

Description

Method for the AzureRMR::az_resource_group and AzureRMR::az_subscription classes.

Usage

```
delete_rec_service(name, confirm = TRUE, free_resources = TRUE)
```

Arguments

- `name`: The name of the recommender service.
- `confirm`: Whether to ask for confirmation before deleting.
- `free_resources`: Whether to delete the individual resources as well as the recommender template.

Value

NULL on successful deletion.

See Also

create_rec_service, delete_rec_service

Examples

```
## Not run:

rg <- AzureRMR::az_rm$
    new(tenant="myaadtenant.onmicrosoft.com", app="app_id", password="password")$
    get_subscription("subscription_id")$
    get_resource_group("rgname")

# delete a recommender service
rg$delete_rec_service("myrec")


## End(Not run)
```

`get_rec_service` *Get existing Azure recommender service*

Description

Method for the AzureRMR::az_resource_group and AzureRMR::az_subscription classes.

Usage

```
get_rec_service(name, data_container = "inputdata")
```

Arguments

- `name`: The name of the recommender service.
- `data_container`: The name of the blob container within the storage account to use for storing datasets.

Value

An object of class `az_rec_service` representing the deployed recommender service.

See Also

create_rec_service, delete_rec_service

Examples

```
## Not run:

rg <- AzureRMR::az_rm$
    new(tenant="myaadtenant.onmicrosoft.com", app="app_id", password="password")$
    get_subscription("subscription_id")$
    get_resource_group("rgname")

# get a recommender service
rg$get_rec_service("myrec")


## End(Not run)
```

item_predict *Get item-to-item recommendations from a SAR model*

Description

Get item-to-item recommendations from a SAR model

Usage

```
item_predict(object, items, k = 10)
```

Arguments

object	A SAR model object.
items	A vector of item IDs.
k	The number of recommendations to obtain.

Value

A data frame containing one row per item ID supplied.

Examples

```
data(ms_usage)
mod <- sar(ms_usage)

# item recomendations for a set of item IDs
items <- unique(ms_usage$item)[1:5]
item_predict(mod, items=items)
```

ms_catalog *Sample catalog dataset*

Description

Dataset of item descriptions from the Microsoft online store.

Usage

```
ms_catalog
```

Format

A data frame with 101 rows and 3 columns.

item The item ID, corresponding to the items in the ms_usage dataset.

name A short description of the item.

category The item category.

Source

Microsoft.

See Also

ms_usage

`ms_usage` *Sample usage dataset*

Description

Dataset of anonymised transaction records from the Microsoft online store.

Usage

```
ms_usage
```

Format

A data frame with 118383 rows and 3 columns.

user The user ID.

item The item ID, corresponding to the items in the ms_catalog dataset.

time The date and time of the transaction, in POSIXct format.

Source

Microsoft.

See Also

ms_catalog

rec_endpoint *Azure product recommendations endpoint class*

Description

Class representing the client endpoint to the product recommendations service.

Format

An R6 object of class `rec_endpoint`.

Methods

- `new(...)`: Initialize a client endpoint object. See 'Initialization' for more details.
- `train_model(...)`: Train a new product recommendations model; return an object of class `rec_model`. See `Training` for more details.
- `get_model(description, id)`: Get an existing product recommendations model from either its description or ID; return an object of class `rec_model`.
- `delete_model(description, id)`: Delete the specified model.
- `upload_data(data, destfile)`: Upload a data frame to the endpoint, as a CSV file. By default, the name of the uploaded file will be the name of the data frame with a ".csv" extension.
- `upload_csv(srcfile, destfile)`: Upload a CSV file to the endpoint. By default, the name of the uploaded file will be the same as the source file.
- `sync_model_list()`: Update the stored list of models for this service.
- `get_swagger_url()`: Get the Swagger URL for this service.
- `get_service_url()`: Get the service URL, which is used to train models and obtain recommendations.

Initialization

The following arguments are used to initialize a new client endpoint:

- `name`: The name of the endpoint; see below. Alternatively, this can also be the full URL of the endpoint.
- `admin_key`: The administration key for the endpoint. Use this to retrieve, train, and delete models.
- `rec_key`: The recommender key for the endpoint. Use this to get recommendations.
- `service_host`: The hostname for the endpoint. For the public Azure cloud, this is `azurewebsites.net`.
- `storage_key`: The access key for the storage account associated with the service.
- `storage_sas`: A shared access signature (SAS) for the storage account associated with the service. You must provide either `storage_key` or `storage_sas` if you want to upload new datasets to the backend.

- `storage_host`: The hostname for the storage account. For the public Azure cloud, this is `core.windows.net`.
- `storage_endpoint`: The storage account endpoint for the service. By default, uses the account that was created at service creation.
- `data_container`: The default blob container for input datasets. Defaults to `"inputdata"`.

Note that the name of the client endpoint for a product recommendations service is *not* the name that was supplied when deploying the service. Instead, it is a randomly generated unique string that starts with the service name. For example, if you deployed a service called "myrec", the name of the endpoint is "myrecusacvjwpk4raost".

Training

To train a new model, supply the following arguments to the `train_model` method:

- `description`: A character string describing the model.
- `usage_data`: The training dataset. This is required.
- `catalog_data`: An optional dataset giving features for each item. Only used for imputing cold items.
- `eval_data`: An optional dataset to use for evaluating model performance.
- `support_threshold`: The minimum support for an item to be considered warm.
- `cooccurrence`: How to measure cooccurrence: either user ID, or user-by-time.
- `similarity`: The similarity metric to use; defaults to "Jaccard".
- `cold_items`: Whether recommendations should include cold items.
- `cold_to_cold`: Whether similarities between cold items should be computed.
- `user_affinity`: Whether event type and time should be considered.
- `include_seed_items`: Whether seed items (those already seen by a user) should be allowed as recommendations.
- `half_life`: The time decay parameter for computing user-item affinities.
- `user_to_items`: Whether user ID is used when computing personalised recommendations.
- `wait`: Whether to wait until the model has finished training.
- `container`: The container where the input datasets are stored. Defaults to the input container for the endpoint, usually `"inputdata"`.

For detailed information on these arguments see the API reference.

See Also

az_rec_service for the service itself, rec_model for an individual recommmendations model

API reference and SAR model description at the Product Recommendations API repo on GitHub

Examples

```
## Not run:

# creating a recommendations service endpoint from an Azure resource
svc <- resgroup$get_rec_service("myrec")
rec_endp <- svc$get_rec_endpoint()

# creating the endpoint from scratch -- must supply admin, recommender and storage keys
rec_endp <- rec_endpoint$new("myrecusacvjwpk4raost",
    admin_key="key1", rec_key="key2", storage_key="key3")

# upload the Microsoft store data
data(ms_usage)
rec_endp$upload_data(ms_usage)

# train a recommender
rec_model <- rec_endp$train_model("model1", usage="ms_usage.csv", support_threshold=10,
    similarity="Jaccard", user_affinity=TRUE, user_to_items=TRUE,
    backfill=TRUE, include_seed_items=FALSE)

# list of trained models
rec_endp$sync_model_list()

# delete the trained model (will ask for confirmation)
rec_endp$delete_model("model1")


## End(Not run)
```

---

rec_model *Azure product recommendations model class*

Description

Class representing an individual product recommendations (SAR) model.

Format

An R6 object of class `rec_model`.

Methods

- `new(...)`: Initialize a model object. See 'Initialization' for more details.
- `delete(confirm=TRUE)`: Delete the model.
- `user_predict(userdata, k=10)`: Get personalised recommendations from the model. See 'Recommendations' for more details.
- `item_predict(item, k=10)`: Get item-to-item recommendations from the model. See 'Recommendations' for more details.
- `get_model_url()`: Get the individual service URL for this model.

Initialization

Generally, the easiest way to initialize a new model object is via the `get_model()` and `train_model()` methods of the `rec_endpoint` class, which will handle all the gory details.

Recommendations

These arguments are used for obtaining personalised and item-to-item recommendations.

- `userdata`: The input data on users for which to obtain personalised recommendations. This can be:
  1. A character vector of user IDs. In this case, personalised recommendations will be computed based on the transactions in the training data, *ignoring* any transaction event IDs or weights.
  2. A data frame containing transaction item IDs, event types and/or weights, plus timestamps. In this case, all the transactions are assumed to be for a single (new) user. If the event types/weights are absent, all transactions are assigned equal weight.
  3. A data frame containing user IDs and transaction details as in (2). In this case, the recommendations are based on both the training data for the given user(s), plus the new transaction details.
- `item`: A vector of item IDs for which to obtain item-to-item recommendations.
- `k`: The number of recommendations to return. Defaults to 10.

Both the `user_predict()` and `item_predict()` methods return a data frame with the top-K recommendations and scores.

See Also

az_rec_service for the service backend, rec_endpoint for the client endpoint

API reference and SAR model description at the Product Recommendations API repo on GitHub

Examples

```
## Not run:

# get a recommender endpoint and previously-trained model
rec_endp <- rec_endpoint$new("myrecusacvjwpk4raost", admin_key="key1", rec_key="key2")
rec_model <- rec_endp$get_model("model1")

data(ms_usage)

# item recommendations for a set of user IDs
users <- unique(ms_usage$user)[1:5]
rec_model$user_predict(users)

# item recommendations for a set of user IDs and transactions (assumed to be new)
user_df <- subset(ms_usage, user %in% users)
rec_model$user_predict(user_df)

# item recomendations for a set of item IDs
```

```
items <- unique(ms_usage$item)[1:5]
rec_model$item_predict(items)


## End(Not run)
```

---

sar *Fit a SAR model*

Description

Fit a SAR model

Usage

```
sar(...)

## S3 method for class 'data.frame'
sar(
  x,
  user = "user",
  item = "item",
  time = "time",
  event = "event",
  weight = "weight",
  ...
)

## Default S3 method:
sar(
  user,
  item,
  time,
  event = NULL,
  weight = NULL,
  support_threshold = 1,
  allowed_items = NULL,
 allowed_events = c(Click = 1, RecommendationClick = 2, AddShopCart = 3, RemoveShopCart
    = -1, Purchase = 4),
  by_user = TRUE,
  similarity = c("jaccard", "lift", "count"),
  half_life = 30,
  catalog_data = NULL,
  catalog_formula = item ~ .,
  cold_to_cold = FALSE,
  cold_item_model = NULL,
  ...
```

```
)

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

Arguments

`...`	For `sar()`, further arguments to pass to the cold-items feature model.
`x`	A data frame. For the `print` method, a SAR model object.
`user`, `item`, `time`, `event`, `weight`	For the default method, vectors to use as the user IDs, item IDs, timestamps, event types, and transaction weights for SAR. For the `data.frame` method, the names of the columns in the data frame `x` to use for these variables.
`support_threshold`	The SAR support threshold. Items that do not occur at least this many times in the data will be considered "cold".
`allowed_items`	A character or factor vector of allowed item IDs to use in the SAR model. If supplied, this will be used to categorise the item IDs in the data.
`allowed_events`	The allowed values for `events`, if that argument is supplied. Other values will be discarded.
`by_user`	Should the analysis be by user ID, or by user ID and timestamp? Defaults to userID only.
`similarity`	Similarity metric to use; defaults to Jaccard.
`half_life`	The decay period to use when weighting transactions by age.
`catalog_data`	A dataset to use for building the cold-items feature model.
`catalog_formula`	A formula for the feature model used to compute similarities for cold items.
`cold_to_cold`	Whether the cold-items feature model should include the cold items themselves in the training data, or only warm items.
`cold_item_model`	The type of model to use for cold item features.

Details

Smart Adaptive Recommendations (SAR) is a fast, scalable, adaptive algorithm for personalized recommendations based on user transaction history and item descriptions. It produces easily explainable/interpretable recommendations and handles "cold item" and "semi-cold user" scenarios.

Central to how SAR works is an item-to-item *co-occurrence matrix*, which is based on how many times two items occur for the same users. For example, if a given user buys items i_1 and i_2, then the cell (i_1, i_2) is incremented by 1. From this, an item *similarity matrix* can be obtained by rescaling the co-occurrences according to a given metric. Options for the metric include Jaccard (the default), lift, and counts (which means no rescaling).

Note that the similarity matrix in SAR thus only includes information on which users transacted which items. It does not include any other information such as item ratings or features, which may be used by other recommender algorithms.

#' The SAR implementation in R should be usable on datasets with up to a few million rows and several thousand items. The main constraint is the size of the similarity matrix, which in turn depends (quadratically) on the number of unique items. The implementation has been successfully tested on the MovieLens 20M dataset, which contains about 138,000 users and 27,000 items. For larger datasets, it is recommended to use the Azure web service API.

Value

An S3 object representing the SAR model. This is essentially the item-to-item similarity matrix in sparse format, along with the original transaction data used to fit the model.

Cold items

SAR has the ability to handle cold items, meaning those which have not been seen by any user, or which have only been seen by a number of users less than `support_threshold`. This is done by using item features to predict similarities. The method used for this is set by the `cold_items_model` argument:

- If this is `NULL` (the default), a manual algorithm is used that correlates each feature in turn with similarity, and produces a predicted similarity based on which features two items have in common.
- If this is the name of a modelling function, such as `"lm"` or `"randomForest"`, a model of that type is fit on the features and used to predict similarity. In particular, use `"lm"` to get a model that is (approximately) equivalent to that used by the Azure web service API.

The data frame and features used for cold items are given by the `catalog_data` and `catalog_formula` arguments. `catalog_data` should be a data frame whose first column is item ID. `catalog_formula` should be a one-sided formula (no LHS).

This feature is currently experimental, and subject to change.

See Also

Description of SAR at the Product Recommendations API repo on GitHub

Examples

```
data(ms_usage)

## all of these fit the same model:

# fit a SAR model from a series of vectors
mod1 <- sar(user=ms_usage$user, item=ms_usage$item, time=ms_usage$time)

# fit a model from a data frame, naming the variables to use
mod2 <- sar(ms_usage, user="user", item="item", time="time")

# fit a model from a data frame, using default variable names
mod3 <- sar(ms_usage)
```

`user_predict` *Get personalised recommendations from a SAR model*

Description

Get personalised recommendations from a SAR model

Usage

```
user_predict(
  object,
  userdata = NULL,
  k = 10,
  include_seed_items = FALSE,
  backfill = FALSE,
  reftime
)

set_sar_threads(n_threads)
```

Arguments

`object`	A SAR model object.
`userdata`	A vector of user IDs, or a data frame containing user IDs and/or transactions. See below for the various ways to supply user information for predicting, and how they affect the results.
`k`	The number of recommendations to obtain.
`include_seed_items`	Whether items a user has already seen should be considered for recommendations.
`backfill`	Whether to backfill recommendations with popular items.
`reftime`	The reference time for discounting timestamps. If not supplied, defaults to the latest date in the training data and any new transactions supplied.
`n_threads`	For `set_sar_threads`, the number of threads to use. Defaults to half the number of logical cores.

Details

The SAR model can produce personalised recommendations for a user, given a history of their transactions. This history can be based on either the original training data, or new events, based on the contents of `userdata` argument:

1. A character vector of user IDs. In this case, personalised recommendations will be computed based on the transactions in the training data, *ignoring* any transaction event IDs or weights.

2. A data frame containing transaction item IDs, event types and/or weights, plus timestamps. In this case, all the transactions are assumed to be for a single (new) user. If the event types/weights are absent, all transactions are assigned equal weight.
3. A data frame containing user IDs and transaction details as in (2). In this case, the recommendations are based on both the training data for the given user(s), plus the new transaction details.

In SAR, the first step in obtaining personalised recommendations is to compute a user-to-item affinity matrix A. This is essentially a weighted crosstabulation with one row per unique user ID and one column per item ID. The cells in the crosstab are given by the formula

$$sum(wt * 2^{(} - (t0 - time)/half_life))$$

where `wt` is obtained from the `weight` and `event` columns in the data.

The product of this matrix with the item similarity matrix S then gives a matrix of recommendation scores. The recommendation scores are sorted, any items that the user has previously seen are optionally removed, and the top-N items are returned as the recommendations.

The latter step is the most computationally expensive part of the algorithm. SAR can execute this in multithreaded fashion, with the default number of threads being half the number of (logical) cores. Use the `set_sar_threads` function to set the number of threads to use.

Value

For `user_predict`, a data frame containing one row per user ID supplied (or if no IDs are supplied, exactly one row).

See Also

Making recommendations at the Product Recommendations API repo on GitHub

Examples

```
data(ms_usage)
mod <- sar(ms_usage)

# item recommendations given a vector of user IDs
users <- unique(ms_usage$user)[1:5]
user_predict(mod, userdata=users)

# item recommendations given a set of user IDs and transactions (assumed to be new)
user_df <- subset(ms_usage, user %in% users)
user_predict(mod, userdata=user_df)

# item recomendations for a set of item IDs
items <- unique(ms_usage$item)[1:5]
item_predict(mod, items=items)

# setting the number of threads to use when computing recommendations
set_sar_threads(2)
```

Index