

Documentation: **recommendx**

Installation

recommendx has the following dependencies:

- Python (≥ 3.5)
- NumPy (≥ 1.10)
- Pandas (≥ 0.18)

The easiest way to install **recommendx** is using **pip** :

```
pip install recommendx
```

Alternatively, the package can be accessed from [Github](#).

The package contains two related prediction algorithms: **RWR** and **RWT** . These are discussed below.

Recommendation with Regressors (RWR)

RWR implements a slightly modified version of what we might call the “classic” SVD algorithm. This is often attributed to [Simon Funk](#), who famously used it during the [Netflix Prize](#) competition. The classic SVD approach relies upon latent item attributes. **RWR** extends this framework by allowing the researcher to specify observed item attributes, as well.

We can define \hat{r}_{ui} as user u 's predicted rating for item i :

$$\hat{r}_{ui} = \mu + b_u + X_i\beta_u + Z_i\alpha_u$$

In this specification,

- μ is the average rating in the data

- b_u is the bias for user u
- X_i is a vector of **observed** attributes for item i
- Z_i is a vector of **latent** attributes for item i
- β_u and α_u are user u 's preferences for observed and latent item attributes, respectively

This specification is similar to the usual matrix factorization set-up, with the standard item bias term (b_i) replaced by observed attributes.

Defining R as the set of all observed user-item ratings and imposing L2-regularization on our parameters, we seek to minimize the following objective function:

$$\sum_{r_{ui} \in R} (r_{ui} - \hat{r}_{ui})^2 + \lambda(b_u^2 + \|\beta_u\|^2 + \|Z_i\|^2 + \|\alpha_u\|^2)$$

The minimization is done using stochastic gradient descent (SGD). The relevant gradients, which can easily be obtained by hand, lead to the following update rules:

- $b_u \leftarrow b_u + \gamma(e_{ui} - \lambda b_u)$
- $\beta_u \leftarrow \beta_u + \gamma(e_{ui} X_i - \lambda \beta_u)$
- $\alpha_u \leftarrow \alpha_u + \gamma(e_{ui} Z_i - \lambda \alpha_u)$
- $Z_i \leftarrow Z_i + \gamma(e_{ui} \alpha_u - \lambda Z_i)$

where $e_{ui} = r_{ui} - \hat{r}_{ui}$, λ is the regularization penalty term, and γ is the SGD learning rate. The learning rate determines how large of a “step” to take when we update parameters.

Parameters

Note: I have purposely chosen the parameter names to be similar to this in [Surprise](#) in order to facilitate easy movement between packages.

- **n_factors** - The number of latent factors in Z . Default is **50**.
- **n_epochs** - The number of iterations of the SGD procedure. Default is **50**.
- **init_mean** - The mean of the normal distribution used to initialize parameter values. Default value is **0**.

- **init_std_dev** - The standard deviation of the normal distribution used to initialize parameter values. Default is **0.1**.
- **reg** - The regularization penalty term used for all parameters (λ). Default is **0.02**.
- **lr** - The learning rate for all parameters (γ). Default is **0.005**.

Attributes

Once an **RWR** instance is **fit()**, the resulting parameter values are returned as attributes of the instance.

- **intercept_** (μ)
 - Scalar intercept term
- **bu** (b_u)
 - NumPy array with shape (**n_users** ,1)
- **B** (β_u)
 - If X_i is provided to **fit()** (see below), **B** is a NumPy array with shape (**n_users** , **n_Xs**)
- **alpha_** (α_u)
 - NumPy array with shape (**n_users** , **n_factors**)
- **z** (Z_i)
 - NumPy array with shape (**u_items** , **n_factors**)

Methods

- **fit(self,df,Xi=None)**
 - Fits the recommender system model
 - **df** must be a NumPy array
 - Each row corresponds to a rating (r_{ui})
 - Columns **must** be ordered: [**user** , **item** , **rating**]
 - **user** and **item** may be strings or integers
 - **Xi** (if supplied) must be a NumPy array
 - If no observed item attributes are supplied, **fit()** returns the same results as SVD
 - First column of **Xi** must be item identifier that corresponds with item labels used in **df**

- Shape of array is (`n_items` , 1+ `n_Xs`)
- `accuracy(self,df,Xi=None)`
 - Returns the mean squared prediction error
 - Requires the recommender system be fit first
 - All provided values must be in the same format as supplied to `fit()`
- `predict(self,u_p,i_p)`
 - Returns predicted ratings
 - `u_p` is a user value
 - `i_p` is an item value
 - Both `u_p` and `i_p` must be provided in the same format as `fit()`

Sample Syntax

If we assume that `dat` is a NumPy array of ratings data and `att` is a NumPy array of observed item attributes, we can use the following code:

```
from recommendx import RWR
rwr = RWR(n_factors = 5)
rwr.fit(dat,att)
rwr.accuracy(dat,att)
rwr.predict('userA','item1')
```

Recommendation with Time (RWT)

`RWT` implements the same basic model as `RWR` but allows for time-varying taste parameters.

Our main ratings prediction equation becomes:

$$\hat{r}_{uit} = \mu + b_u + X_i\beta_{u,t} + Z_i\alpha_{u,t}$$

Neither observed (X_i) nor latent (Z_i) item attributes vary with time, although one could “trick” the model into allowing that by creating items that are time-specific.

User taste parameters $\beta_{u,t}$ and $\alpha_{u,t}$ are assumed to vary by time period. This allows for the possibility, for example, that a Netflix viewer might be more inclined to enjoy a horror movie at night. Or a coffee drinker may prefer espresso drinks more in the morning than in the evening.

RWT requires that time be defined categorically. A simple example might be that time takes the values [“Morning”, “Afternoon”, “Night”]. These categorical labels must be assigned by the researcher prior to fitting the recommender.

The model is fit using SGD. The equations are identical to those for **RWR** with the exception that the β and α parameters are now subscripted with time, as well.

Parameters

RWT has the same model parameters as **RWR**. Parameter arrays $\beta_{u,t}$ and $\alpha_{u,t}$ are identified only using ratings observations for each specific time period. To account for this, the default value of **n_epochs** has been increased to **100**.

Attributes

Once an **RWT** instance is **fit()**, the resulting parameter values are returned as attributes of the instance.

- **intercept_** (μ)
 - Scalar intercept term
- **bu** (b_u)
 - NumPy array with shape (**n_users** ,1)
- **B** ($\beta_{u,t}$)
 - If X_i is provided to **fit()** (see below), **B** is a 3-dimensional NumPy array with shape (**n_times** , **n_users** , **n_Xs**)
- **alpha_** ($\alpha_{u,t}$)
 - A 3-dimensional NumPy array with shape (**n_times** , **n_users** , **n_factors**)
- **z** (Z_i)
 - NumPy array with shape (**u_items** , **n_factors**)

Methods

- `fit(self, df, Xi=None)`
 - Fits the recommender system model
 - `df` must be a NumPy array
 - Each row corresponds to a rating (r_{ui})
 - Columns **must** be ordered: [`user` , `item` , `rating` , `time`]
 - `user` and `item` may be strings or integers
 - `time` should be the time label for r_{uit} . Can be string or integer but is treated as categorical
 - `Xi` (if supplied) must be a NumPy array
 - If no observed item attributes are supplied, `fit()` returns the results for SVD with time-varying parameters
 - First column of `Xi` must be item identifier that corresponds with item labels used in `df`
 - Shape of array is (`n_items` , 1+ `n_Xs`)
 - `accuracy(self, df, Xi=None)`
 - Returns the mean squared prediction error
 - Requires the recommender system be fit first
 - All provided values must be in the same format as supplied to `fit()`
 - `predict(self, u_p, i_p, tee)`
 - Returns predicted ratings
 - `u_p` is a user value
 - `i_p` is an item value
 - `tee` is a time value
 - Both `u_p` , `i_p` , and `tee` must be provided in the same format as `fit()`

Sample Syntax

If we assume that `dat` is a NumPy array of ratings data (with time label) and `att` is a NumPy array of observed item attributes, we can use the following code:

```
from recommendx import RWT
rwt = RWT(n_factors = 3)
rwt.fit(dat,att)
rwt.accuracy(dat,att)
rwt.predict('userA','item1','AM')
rwt.predict('userA','item1','PM')
```