

Recommender

Algorithms

Using Collaborative Filtering based on SVD to predict rate.

Collaborative Filtering is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating). The underlying assumption of the collaborative filtering approach is that if a person A has the same opinion as a person B on an issue, A is more likely to have B's opinion on a different issue than that of a randomly chosen person.

Simply SVD is to decompose a matrix of $m \times n$ into three matrices(U , Σ , V) as shown below. First, Use the SVD to create a matrix of user matrix(U), property matrix(Σ) and movie matrix(V) using given movie ratings.

$$\begin{matrix} \boxed{A} & = & \boxed{U} & \boxed{\Sigma} & \boxed{V^T} & \approx & \boxed{U_k} & \boxed{\Sigma_k} & \boxed{V_k^T} & = & \boxed{A_k} \\ t \times d & & t \times m & m \times m & m \times d & & t \times k & k \times k & k \times d & & t \times d \end{matrix}$$

If you do this, we can get diagonal matrix(Σ) which can called features. Using the computed U , V , and key features of Σ , can create approximate of original matrix. So, we can predict undefined values, using created U , Σ and V .

Implementations

Python is a simple language that is easy enough to understand directly. So it's not difficult to see and understand the code right away. But here are tips for Python beginners.

`lib.recommender.Recommender`

Default recommender class. You can choose which algorithm to use for the recommender system, but only SVD is currently implemented.

`lib.algorithms.factorization.SVD`

This class is basically written in `Cython`. This is because the matrix operation takes too long, so increase the calculation efficiency using `Cython`.

This class accepts parameters `factors`, `epochs`, `init_mean`, `init_derivation`, `learning_rate`, `regression_rate`.

And, there are two methods. Simply, `fit` create feature matrix and `predict` return predicted value using created feature matrix.

- `fit`

Train with train data. Create `bias` and `param` of each user, item. `unique` is indexer to compress given data. Caculate `dot` and `error` to update `bias` and `param`. `lr` and `reg` is rate of update values.

Calculate current errors

```
dot = sum(param_item[i, f] * param_user[u, f] for f in range(self.factors))
err = r - (mean + biase_user[u] + biase_item[i] + dot)
```

Update `bias` parts at line 64-65.

```
biase_user[u] += lr * (err - reg * biase_user[u])
```

Update `param` parts at line 68-70.

```
for f in range(self.factors):
    param_user[u, f] += lr * (err * param_item[i, f] - reg * param_user[u, f])
    param_item[i, f] += lr * (err * param_user[u, f] - reg * param_item[i, f])
```

- `predict`

Return prediction value which create using `bias` and `param` matrix.

Requirements

- NumPy: is the fundamental package for scientific computing with Python.
- Pandas: is providing high-performance, easy-to-use data structures and data analysis tools for the Python.
- Cython: support compiled language, generates CPython extension modules.

install packages using pip

```
pip3 install -r requirements.txt
```

Tested @ python3.5 in Ubuntu 16.04 LTS, macOS High Sierra and Windows 10

Run as below

```
// First of all, build Cython extensions to compile .pyx
// It's speeds up computation
python setup.py build_ext --inplace

// Run recommender, -h to show additional arguments
python recommender.py [train_data_path] [test_data_path]
```

Performance

	u1	u2	u3	u4	u5
RMSE	0.957	0.943	0.936	0.933	0.933

