Recommender Systems: Neighborhood-based Filtering

AAA-Python Edition

Plan

- 1- Introduction
- 2- Collaborative Filtering
- 3- Similarity scores
- 4- Cross-Validation
- 5- User-based Collaborative Filtering With Surprise
- 6- Item-based Collaborative Filtering With Surprise

1- Introduction

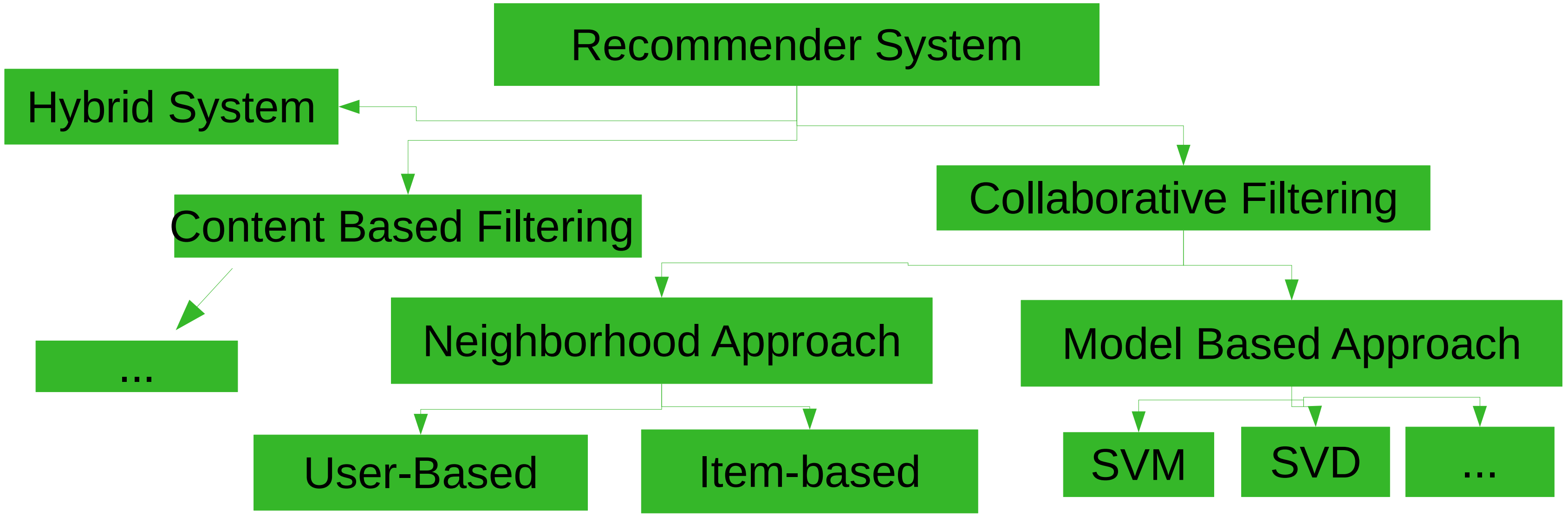
Definition

- From [Francesco et al., 2011] **recommender systems** are software tools and techniques that provide **suggestions** for **items** to be of use of **users**.
- The systems can suggest to the user things like what item to buy or what movie to watch.
- Recommender system can predict reviews of users on new items, as well as they can predict items properties.
- The **3** main **approaches** to build a Recommender System are: **Content Based** , **Collaborative filtering** and **Hybrid systems.**
- An hybrid system is the one that combine different approaches and techniques in order to eliminate some disadvantages of these approaches

[By Amina Delali]

Categories

- In collaborative filtering, we can **build** a recommender system following **2 other** approaches: **Neighborhood approach** and **model based** approach.
- Each of the approaches cited above, group a set of different techniques.



Libraries

- There are different libraries and frameworks (other than sklearn) that we can use to **build** our **recommender** system (definitions extracted from their respective websites):
  - Surprise Library: A Python scikit for recommender systems.
  - Crab Library: A Recommender System in python
  - Polylearn:
    A library for factorization machines and polynomial networks
  - Graphlab:
    Simple development of custom machine learning models

school of ai

2- Collaborative Filtering

Neighborhood Approach

- In collaborative filtering, to predict ratings of a **user u** on an **item j,** the **ratings** of that user are taken into account, as well as the ratings of **other users**.

- In neighborhood approach**(memory-based),** the ratings are used directly to make these predictions following **2** approaches:

  - **User-based** recommendation: the ratings of a user **u** for an item **j** are obtained from the ratings given by the **neighbors** of that user, to that item. The neighbors are those who had **similar** rating patterns as the user **u** for other items that they have rated in common.

  - **Item-based** recommendation: the ratings of a user **u** for an item **j** are obtained from the ratings of that user **u** for **other similar items** to the item **j.** Two items are similar, if they have been rated in the same way by **other users.**

Model based approach

- In this case, the ratings of users for items are not used directly. They are instead used to **create** a **model.**

- **The created model** will be used later to predict **ratings** for **new items.**

    - Some of them are **latent factor models.** They rely on the idea that there are **latent** (hidden) **characteristics** of the **users** and **items.** So, the interaction user-item will be modeled with **factors** that represent these characteristics.

        - Several techniques can be used as **matrix factorization** with **singular value decomposition.**
        - The models can also be created using **supervised learning techniques**. They are **trained** using the **user-item iteractions**. And then used to predict **new values**.
        - In this case, **Support vector machines (SVM)** can be used.

Knn algorithm

- **K nearest neighborhood** algorithm can be used for both **item-based** and **user-based** recommendation.

- The following steps, describe the application of this algorithm for **user-based** recommendation, to predict ratings of user **u** on item **j** :

  - Each user is represented by a **vector** of his ratings

  - A **similarity** measure is selected to identify **similar** users.

  - Find the **k most similar users** to the user **u**, that have rated the item **j.**

  - **Predict** the ratings of the user **u** for the item **j** by computing the **weighted average** of the ratings of the **k neighbors** of the user **u**, for the item **j.**

Introduction

- A **similarity score** is a value that describes the degree of similarity between users or items.

- This degree can be computed using different formula:

  - **Cosine similarity** : the users (items) are vectors, and the similarity is described by the **cosine** of the **angle** formed by a pair of these vectors.

  - **MSD**: the mean squared difference between pairs of users (items)

  - **Pearson score:** measures the **correlation (linear relationship)** between pairs of items (users).

  - **Pearson score** with a **shrinkage parameter:** computes the correlation between pairs using **baselines** and a **shrunk parameter** , to avoid overfitting.

3- Similarity scores

[By Amina Delali]

3- Similarity scores

Cosine Similarity

- **Cosine**:

$$\text{cosine\_sim}(u,v)=\frac{\sum_{i\in I_{uv}} r_{ui}\cdot r_{vi}}{\sqrt{\sum_{i\in I_{uv}} r_{ui}^2}\cdot\sqrt{\sum_{i\in I_{uv}} r_{vi}^2}}$$

Common **items** between user **u** and user **b**

Rating of the user **u** for the item **i**

$$\text{cosine\_sim}(i,j)=\frac{\sum_{u\in U_{ij}} r_{ui}\cdot r_{uj}}{\sqrt{\sum_{u\in U_{ij}} r_{ui}^2}\cdot\sqrt{\sum_{i\in U_{ij}} r_{uj}^2}}$$

Common **users** between item **i** and item **j**

Only the common users (items) are taken into account. The formula is used either to compute the **similarity score between users** or between **items.** The values range from **0** to **1**

[By Amina Delali]

3- Similarity scores

Pearson similarity

- **Pearson:**

$$\text{pearson\_sim}(u,v)=\frac{\sum_{i\in I_{uv}}(r_{ui}-\mu_u)\cdot(r_{vi}-\mu_v)}{\sqrt{\sum_{i\in I_{uv}}(r_{ui}-\mu_u)^2}\cdot\sqrt{\sum_{i\in I_{uv}}(r_{vi}-\mu_v)^2}}$$

The **mean** of the ratings made by the **user u**

$$\text{pearson\_sim}(i,j)=\frac{\sum_{u\in U_{ij}}(r_{ui}-\mu_i)\cdot(r_{uj}-\mu_j)}{\sqrt{\sum_{u\in U_{ij}}(r_{ui}-\mu_i)^2}\cdot\sqrt{\sum_{u\in U_{ij}}(r_{uj}-\mu_j)^2}}$$

The **mean** of the ratings of the **item j**

Only the common users (items) are taken into account. This score can be seen as the **mean centered cosine** similarity score. The values range from **-1** to **1**

4- Cross-Validation

Introduction

- To measure the performance of a model, we split the data into **training** and **testing** set. We train the data using only the training set. And, finally we test our model on the testing sets.

- But, when we want to identify the best parameters for ou estimators, we train and **test** our models several times. **Indirectly,** the test set values will interfere in the training. And the metrics used to estimate our models, will no longer reflect the actual performance of the model.

- To avoid this situation, another split is necessary: the **validation set**. We use it to test our parameters. And when we are done, we perform a final test on the test data.

- But when the data is too small, we use instead a **cross-validation** technique without using a validation set.

[By Amina Delali]

K-fold Cross validation

- The cross-validation technique can be applied with different approaches. A basic one is the **k-fold cross-validation**
  - The data set is split into **k** smaller **folds**
  - The model is trained on the **k-1 folds**
  - For each training, the model is tested on the **remaining fold.** And a **corresponding score** is computed.
  - **The performance score** of the model is the **average** of all the scores computed previously
- The cross validation technique is generally combined with an other tool: the **Grid Search** tool(already seen in **Ensemble Learning Lessons)**
- In fact, the GridSerchCVclass of scklearn can be parameterized by specifying the **cv** parameter (the cross validations strategy to use).

Cross-validation implementation

- In **sklearn,** we can use the **cross-validation** in different manners:
  - Using the cross validation indirectly by using the **GridSearchCV** class
  - Using the **cross_validate** and **cross_val_score** functions
  - Splitting the data using the corresponding fold strategy for a cross-validation approach as the **KFold** class
- In **Surprise** library, that we will use for our Recommender Systems examples, implements the cross-validation as well:
  - Using the **GridSearchCV** class and iterators as **Kfold**
  - Using the **cross_validate** function

5- User-based Collaborative Filtering With Surprise

Library and data

```
#installation of surprise library
!pip install surprise
```

Use all the data of the training

The items ratings and the users ratings

```
from surprise import Dataset

# load movielens-100k dataset
myData = Dataset.load_builtin('ml-100k')

# Retrieve the trainset.
trainset = myData.build_full_trainset()

print ("The number of items = ",trainset.n_items)
print ("The number of users = ",trainset.n_users)
print ("The number of ratings= ",trainset.n_ratings)
print("The rating for the item 0 by the user ",trainset.ir[0][2][0]," is:",trainset.ir[0][2][1])
print("The rating of the user 0 for the item",trainset.ur[0][1][0]," is:",trainset.ur[0][1][1])
```

User and item inner IDs,
As defined in
build_full_trainset method

```
The number of items =  1682
The number of users =  943
The number of ratings=  100000
The rating for the item 0 by the user  218  is: 5.0
The rating of the user 0 for the item 528  is: 4.0
```

The data contains 100000 ratings of 943 users for 1682 items

[By Amina Delali]

school of ai

5- User-based Collaborative Filtering With Surprise

Similarity matrix computation (fitting)

```
from surprise import KNNBasic


# We'll use the basic Knn algorithm.
# we will use a user based estimator, using cosine score
Recommender = KNNBasic(sim_options={"name":"cosine","user_based":True})

# fitting to the data ==> compute similarity scores between users
Recommender.fit(trainset)
#the similarity matrix
Recommender.sim
```

To have an **item based** collaborative filtering, all you have to do is to set the parameter “**user_base**” to **False.**

Similarity value between **user 0 and user 1**

```
array([[1.        , 0.87278605, 0.91226401, ..., 0.86717176, 0.84366149,
        0.9486833 ],
       [0.87278605, 1.        , 0.84761034, ..., 0.8782826 , 0.87552384,
        0.94252177],
       [0.91226401, 0.84761034, 1.        , ..., 0.88184244, 1.        ,
        0.90116647],
       ...,
       [0.86717176, 0.8782826 , 0.88184244, ..., 1.        , 0.89504128,
        0.93603858],
       [0.84366149, 0.87552384, 1.        , ..., 0.89504128, 1.        ,
        0.98994949],
       [0.9486833 , 0.94252177, 0.90116647, ..., 0.93603858, 0.98994949,
        1.        ]])
```

[By Amina Delali]

Prediction

- The used formulas:

User-based filtering

$$\hat{r}_{ui} = \frac{\sum_{v\in N_i^k(u)} \text{sim}(u,v)\cdot r_{vi}}{\sum_{v\in N_i^k(u)} \text{sim}(u,v)}$$

Item-based filtering

$$\hat{r}_{ui} = \frac{\sum_{j\in N_u^k(i)} \text{sim}(i,j)\cdot r_{uj}}{\sum_{j\in N_u^k(i)} \text{sim}(i,j)}$$

The prediction (estimation) of the **rating** of the **user u** for the **item i**

Are the **k** most similar **items** to item **i,** rated by **u**

Similarity score between **i** and **j**

```
user_raw_id = trainset.to_raw_uid(218)
item_raw_id=  trainset.to_raw_iid(528)
print(Recommender.predict(user_raw_id,item_raw_id))
```

```
user: 226          item: 393          r_ui = None    est = 3.50
```

The estimation is equal to 3.50

```
{'actual_k': 40, 'was_impossible': False}
```

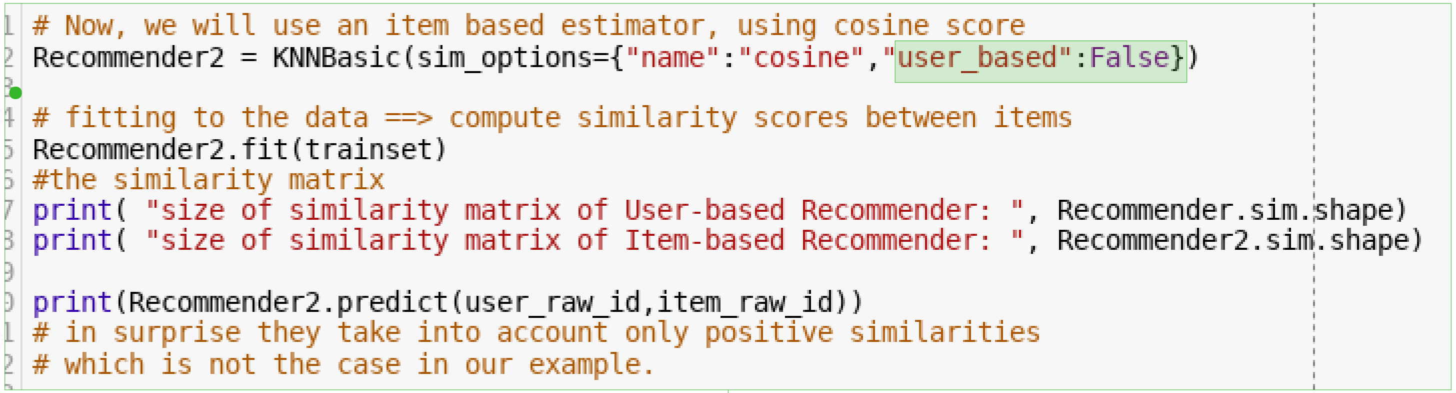
Actual number of neighbors (it could be less than 40: the default value)

The prediction was possible: minimum amount of neighbors was available

[By Amina Delali]

6- Item-based Collaborative Filtering With Surprise

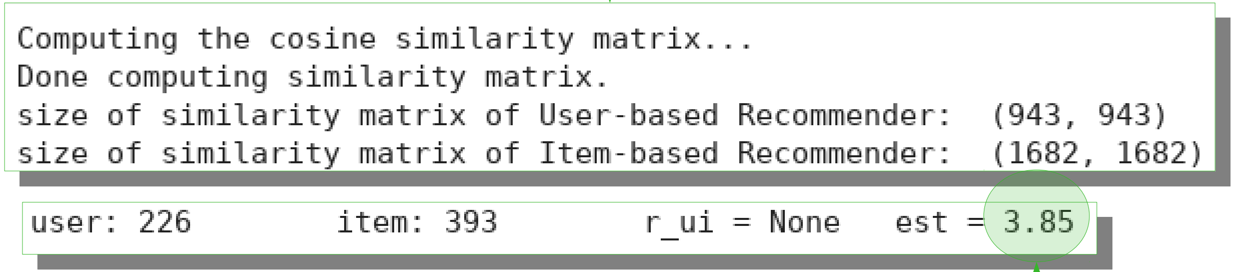
Prediction



```python
# Now, we will use an item based estimator, using cosine score
Recommender2 = KNNBasic(sim_options={"name":"cosine","user_based":False})

# fitting to the data ==> compute similarity scores between items
Recommender2.fit(trainset)
#the similarity matrix
print( "size of similarity matrix of User-based Recommender: ", Recommender.sim.shape)
print( "size of similarity matrix of Item-based Recommender: ", Recommender2.sim.shape)

print(Recommender2.predict(user_raw_id,item_raw_id))
# in surprise they take into account only positive similarities
# which is not the case in our example.
```



```
Computing the cosine similarity matrix...
Done computing similarity matrix.
size of similarity matrix of User-based Recommender:  (943, 943)
size of similarity matrix of Item-based Recommender:  (1682, 1682)
```

```
user: 226          item: 393          r_ui = None    est = 3.85
```

The estimation is different from the previous one **3.85** instead of 3.50

[By Amina Delali]

6- Item-based Collaborative Filtering With Surprise

More Details

```
#number of neighbors
k= Recommender2.k
print("Number of neighbors taken into account:",k)
```

```
Number of neighbors taken into account: 40
```

By default, the number of neighbors is **40**

1

```
import numpy as np
# RL number of items rated by user 218 (rawid ="226")
RL = len(trainset.ur[218])

# inner ids of items rates by the user 218
ratedByU =[trainset.ur[218][i][0] for i in range(RL)]
# the corresponding ratings by the user 218
ratesOfU =[trainset.ur[218][i][1] for i in range(RL)]
# similarities between the item 528 (rawid ="393")
similaritiesU = Recommender2.sim[528,ratedByU]
# the indices of ordred similarites (descending order)
# it was ascending, then [::-1]reversed the order
indSortU = np.argsort(similaritiesU)[::-1]
```

2

```
# select the k indices corresponding to
# the k greatest sorted similarities
if RL < k :
  indSortUk = indSortU[:RL]
else:
  indSortUk = indSortU[:k]
```

3

```
 # select the k sorted greatest similarities
similaritiesUk = similaritiesU[indSortUk]
ratesOfU = np.asarray(ratesOfU)
# selct the k corresponding ratings
ratesOfUk = ratesOfU[indSortUk]
# the sum of the k similarities
simSum = similaritiesUk.sum()
# the sum of the weighted similarities
# the weights are the corresponding ratings
weightedSum = (similaritiesUk * ratesOfUk).sum()
# the estimated rating
finalRating = weightedSum/simSum
```

In fact, the algorithm applied in **Surprise** library **doesn't** take into account the **negative similarities** even if they correspond to the **k** most near items (or users). Which is not reproduced in our example.

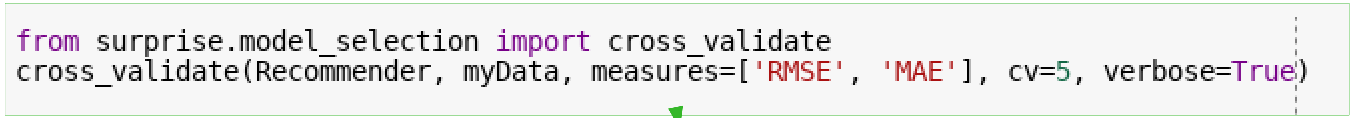
```
print("Estimated rating of the user 218 ('226'), for the item 528('393') is : %1.2f" % finalRating)
```

```
Estimated rating of the user 218 ('226'), for the item 528('393') is : 3.85
```

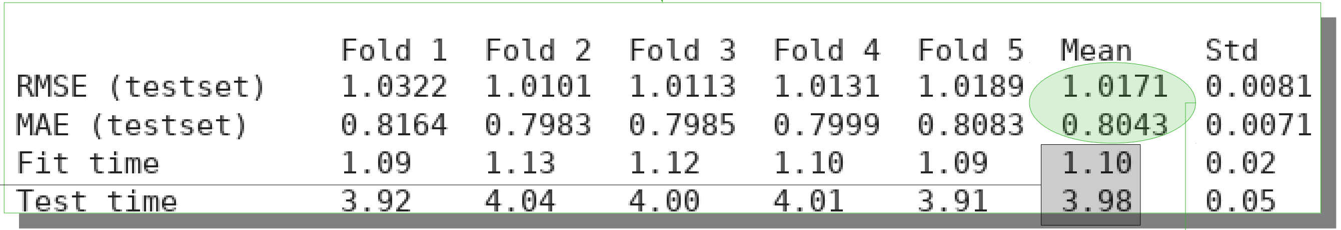
[By Amina Delali]

Comparison and cross-validation

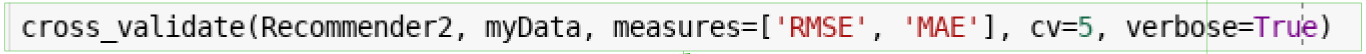
- We will use the “cross-validate” function from “Surprise.model_selection” package in order to compare the performances of the **user** and **item** based **Recommender Systems**



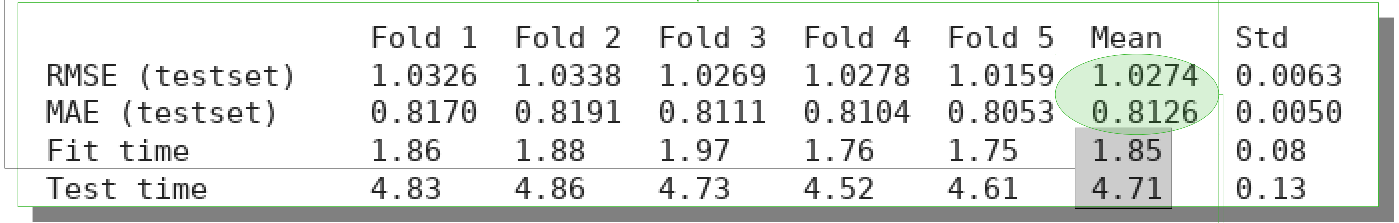
```
from surprise.model_selection import cross_validate
cross_validate(Recommender, myData, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```



```
                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.0322  1.0101  1.0113  1.0131  1.0189  1.0171  0.0081
MAE (testset)     0.8164  0.7983  0.7985  0.7999  0.8083  0.8043  0.0071
Fit time          1.09    1.13    1.12    1.10    1.09    1.10    0.02
Test time         3.92    4.04    4.00    4.01    3.91    3.98    0.05
```



```
cross_validate(Recommender2, myData, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```



```
                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.0326  1.0338  1.0269  1.0278  1.0159  1.0274  0.0063
MAE (testset)     0.8170  0.8191  0.8111  0.8104  0.8053  0.8126  0.0050
Fit time          1.86    1.88    1.97    1.76    1.75    1.85    0.08
Test time         4.83    4.86    4.73    4.52    4.61    4.71    0.13
```

The **fit** and **test** time are **bigger** for the **item-based** filtering (which is logic, since the number of items is much bigger than the number of users)

The errors are slightly more important for item-based filtering

[By Amina Delali]

References

- [Francesco et al., 2011] Francesco, R., Lior, R., Bracha, S., and Paul B., K., editors (2011). Recommender Systems Handbook. Springer Science+Business Media.

- [Grčar et al., 2006] Grčar, M., Fortuna, B., Mladenič, D., and Grobelnik, M. (2006). knn versus svm in the collaborative filtering framework. In Data Science and Classification, pages 251–260. Springer.

- [Hug, 2017] Hug, N. (2017). Surprise, a Python library for recommender systems. http://surpriselib.com.

- [kumar Bokde et al., 2015] kumar Bokde, D., Girase, S., and Mukhopadhyay,D. (2015). Role of matrix factorization model in collaborative filtering algorithm: A survey. CoRR,abs/1503.07475.

school of ai

Thank you!

FOR ALL YOUR TIME