INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact factor: 4.295
(Volume 5, Issue 2)
Available online at: www.ijariit.com

Test case recommendation system

Sudipto Nandan
sudipto.nandan@gmail.com
Oracle India Pvt. Ltd., Bengaluru, Karnataka

ABSTRACT

A code transaction modifies a set of code files and runs a set of tests to make sure that it does not break any of the existing features. But how do we choose the target set of tests? There could be 1000s of tests. Do we run all those tests? In that case, are we not wasting resources, and delaying the whole process of merging a code change? Can we choose a static set of tests? In that case, we have a chance of missing a set of tests which may be related to the code changes. How do we find out the optimal set of tests for a given code change – is the goal which we are trying to solve using the Recommendation System.

Keywords— *Test case selection, Singular value decomposition, SVD, Recommendation system, Latent factor collaborative filtering, Alternating least square*

1. INTRODUCTION

A code transaction making certain enhancements and changes always risk the existing code base. The way this risk is avoided is by running some of the existing tests against the code change. The constraints we have is – the number of tests is too many. Else, we could always run the whole set of tests. With too many tests and running all the tests will entail very high turnaround time for a code change to be merged and very high requirement of hardware resources. The goal of this study is to find an optimal group of tests to be executed with a code change, to make sure that the code change does not break any of the existing functionality. We will try to solve this by building a recommender system.

1.1 Recommender system

Recommender System is used for predicting the "rating" or "preference" that a user (identity -1) would give to an item (another identity -2). Given a user u_1, which likes "certain items/movies", can we predict what other items/movies will u_1 like? This problem is solved by Recommender Systems.
There are two types of Recommender Systems

- Content Based Filtering
- Collaborative Filtering

1.1.1 Content based filtering: We define a set of features/factors e.g. genre, box office collection, stars which can influence the ratings (relationship of both the identities). Then we define all the movies based on these factors/features – a movie belongs to which genre etc. Next, we create user profiles based on these factors – a user like which genre of movie – action, comedy etc. Based on these two mappings, we suggest movies to users. (Figure 1)

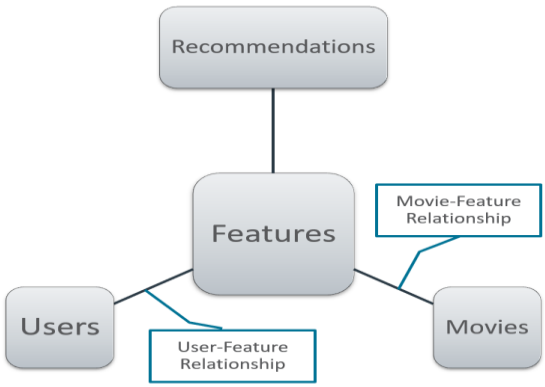

Fig. 1: Content based filtering

1.1.2 Collaborative Filtering: For Content Based Filtering to work, we need to have the user-feature mapping and movie/item-feature mapping done.

- What if – there are millions of such movies, users?
- What if – we do not know the features?
- What if – there are 1000's of features?

To solve these, we have Collaborative Filtering.

Collaborative filtering needs user history: how a user is influenced by some of the items/movies. Based on this, there are various ways using which we can determine what we should recommend to the user.

Collaborative Filtering – Nearest Neighbor Types: If we need to find a good recommendation for a user u_1, based on all user's history, we find what are the other users which are like u_1, i.e. all those users – who have a same/similar influence on a set of movies/items as u_1 has.

We find this using various similarity methods defined in algebra

(a) Euclidian distance

$$d = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

(b) Cosine Similarity

$$CosSim(x,y) = \frac{\sum_i x_i - y_i}{\sqrt{\sum_i x_i^2}_i \sqrt{\sum y_i^2}} = \frac{\langle x,y \rangle}{||x||||y||}$$

(c) Pearson Correlation

$$Corr(x,y) = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2}\sqrt{\sum (y_i - \bar{y})^2}}$$

$$Corr(x,y) = \frac{\langle x - \bar{x}, y - \bar{y} \rangle}{||x - \bar{x}||||y - \bar{y}}$$

$$Corr(x,y) = CosSim(x - \bar{x}, \quad y - \bar{y})$$

Once we have the determined "a set of" similar users (nearest neighbors), we predict the u_1's influence on movies (which it hasn't seen/rated) based on weighted average of those movies of nearest neighbors (figure2).

	Star Wars	Titanic	Rambo	Inception	Gravity
User 1	4			5	4
User 2		5			3
User 3	4			4	1
User 4	4	1			5
User 5	5		4	5	

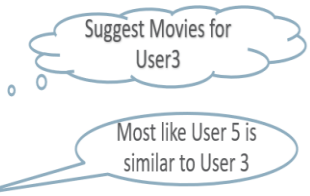

Fig. 2: Collaborative filtering – nearest neighbor types

Collaborative Filtering – Latent Factor Based Methods: This is a mathematical model of collaborative filtering which is analogous to "Singular Vector Decomposition" splits a matrix into two matrices.

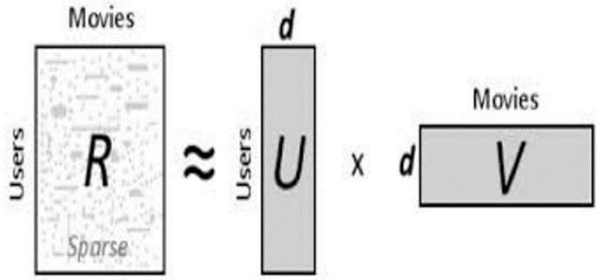

Fig. 3: Singular vector decomposition

We use a learning algorithm to determine the factors. The user's history consists of "users" and "items/movies" with its ratings on each item/movies as a matrix. We apply the learning algorithm on this matrix to split it into two separate matrices. The first matrix shows the relationship between "users" with "hidden factors" [also called latent factors]. The second matrix shows the relationship between each item and the hidden factors. This method is similar to Content-Based Filtering except that the latent factors are identified by the learning algorithm. Sometimes, these factors may map to real factors like Genre or Box office collections. In other cases, it may turn out to be an abstract factor which does not have a meaning in real life.

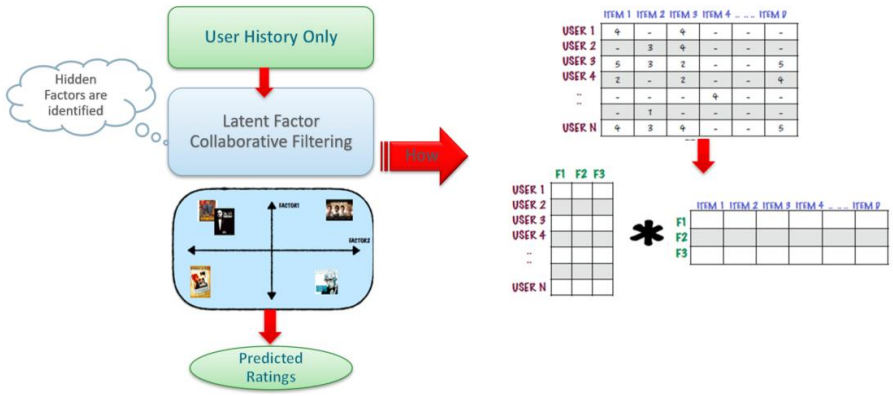

Fig. 4: Latent factor collaborative filtering

1.2 Sparsity

In most of the cases, the initial matrix will be very sparse, i.e. the ratings of movies from users are usually very rare. How do we do the decomposition of the matrix in such cases? In such cases, we perform the matrix factorization only for the values which are available. We solve the equations for the set of ratings which are available. Then we use the resulting matrices to find the rating of any other user and items which is not available. For a given rating r_{ui}, we perform the matrix factorization to get p_u (for each user) and q_i (for each item)

$$R = P * Q \qquad r_{ui} = p_u * q_i$$

This can be turned into a minimization problem.

$$min_{q,p} \sum_{(u,i)\epsilon k} (r_{ui} - q_i^T p_u)^2 \lambda(||q_i||^2 + ||p_u||^2)$$

The lambda term is the regularization term. We find the factor matrices which minimizes the above error. There are two common ways of matrix factorization – Stochastic Gradient Descent and Alternating Least Squares

1.3 Alternating Least Square

The equation to solve is

$$(r_{ui} - q_i^T p_u)^2 + \lambda(||q_i||^2 + ||p_u||^2) = 0$$

There are two variables – p_u and q_i. We first fix p_u and solve the equation and then we fix q_i and solve the equation. This is repeated till up and q_i converges.

1.4 Challenges

- **Cold Start:** Collaborative Filtering depends on user history. What if there is no user history, i.e. for new users, how do we recommend any item? One of the solutions could be to use a combination of content-based and collaborative filtering.
- **Size:** Size of the matrix is always a challenge in matrix factorization. Bigger the size, bigger the computation power, time and memory requirement. Using Alternating Least Square algorithm instead of a stochastic gradient algorithm can help to mitigate the issue to a certain extent.
- **The sparsity of data:** Usually both rows and columns (users and items) are huge in number but the ratings are quite less. We can use dimensions using Principal Component Analysis (PCA) and resolve the issue to some extent.
- **Synonyms:** There are always some products, items which are the same. The data needs to be preprocessed to take care of such cases.
- **Gray Ship:** Some users are not consistent in its ratings. Their ratings may not be easy to put into some mathematical formulae. This issue may be very specific to Movies or Items, but in the scenario where we are trying to deduct ratings, this is not a concern.
- **Shilling Attacks:** Some users can try to give some fake ratings. Again, this is not a concern where we are trying to deduce ratings from other activities.

1.5 How can these recommender system help us?

In our system, we categorize data as
Identity 1 "Code file" ~ "Users"
Identity 2 "test/failure file" ~ "Movies/Items"

We try to predict – given a Code File, a list of Test files with a strong relationship. We extract relationships from all the past instances of failures which we identified are caused by certain code changes. The strength of the relationships depends on how many such relationships we find. We also preprocess the data to remove all irrelevant data and clean it up. Once we have all the data, we standardize the data so that all the ratings/relationship values are in the same range of 0 to 1. We convert it into a matrix form and it looks something like figure 5.

cid	0	1	2	3	4	5	6	7	8	9	10	11
did												
0												
1												
2												
3												
4												
5												
6				0.125								
7												
8												
9												
10												
11				0.075	0.075		0.075					

Fig. 5: Code file ~ test File relationship matrix

We try to collect as much data as possible. In our case study, we used a dataset of 2.4 million data, consisting of 100K+ test files and ~9K codes and forming a matrix of size 100K*9000 = 900M elements.

We train our Machine Learning Model. The Hyper Parameters to be tuned are

- Number latent features
- Regularization term

We have used RMSE (Root Mean Square Error) as the error term calculation for our Case Study. We also used the ALS (Alternate Least Square) algorithm to converge up and q_i

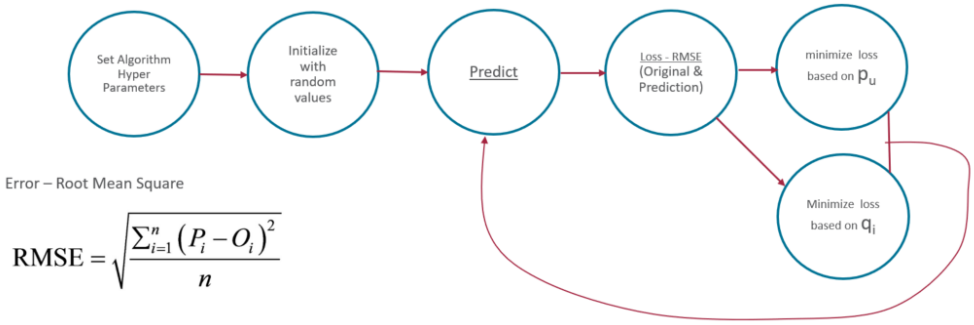

Fig. 6: Matrix decomposition algorithm using ALS

After training, the predicted "influence/relationship values/ratings" values look like figure 7. Once we have the predicted values, we can always predict/deduce, if a relationship between a code file and test file is strong enough or not. Based on these values, we can easily find out the list of top tests which have a strong relationship with a given code file. And thus, if the code file is modified in a transaction, we can help the developers with the list of such test files which it needs to execute.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
0	1.0	0.296618	0.233333	0.333333	0.35	0.372289	0.470370	0.433975	0.372289	0.470370	0.2	0.272967	0.801097	0.166667	0.5	0.8
1	1.0	0.288152	0.233333	0.333333	0.35	0.748275	0.814811	0.661023	0.748275	0.814811	0.2	0.236375	0.708180	0.166667	0.5	0.9
2	1.0	0.267314	0.233333	0.333333	0.35	0.237077	0.337655	0.374046	0.237077	0.337655	0.2	0.207363	0.733301	0.166667	0.5	0.8
3	1.0	0.319868	0.233333	0.333333	0.35	0.443991	0.550600	0.564011	0.443991	0.550600	0.2	0.258426	0.993025	0.166667	0.5	0.7
4	1.0	0.242232	0.233333	0.333333	0.35	0.331170	0.383546	0.359339	0.331170	0.383546	0.2	0.223781	0.700483	0.166667	0.5	0.8
5	1.0	0.242232	0.233333	0.333333	0.35	0.331170	0.383546	0.359339	0.331170	0.383546	0.2	0.223781	0.700483	0.166667	0.5	0.8
6	1.0	0.214460	0.233333	0.333333	0.35	0.148218	0.147092	0.189256	0.148218	0.147092	0.2	0.217337	0.678801	0.166667	0.5	0.8
7	1.0	0.239904	0.233333	0.333333	0.35	0.255215	0.330497	0.338961	0.255215	0.330497	0.2	0.211449	0.703843	0.166667	0.5	0.8
8	1.0	0.299572	0.233333	0.333333	0.35	0.528956	0.506668	0.395350	0.528956	0.506668	0.2	0.297054	0.704287	0.166667	0.5	0.9
9	1.0	0.312005	0.233333	0.333333	0.35	0.414128	0.484410	0.505836	0.414128	0.484410	0.2	0.286470	0.955837	0.166667	0.5	0.7
10	1.0	0.248131	0.233333	0.333333	0.35	0.458318	0.544233	0.477211	0.458318	0.544233	0.2	0.213432	0.707670	0.166667	0.5	0.8
11	1.0	0.320332	0.233333	0.333333	0.35	0.249643	0.228841	0.196457	0.249643	0.228841	0.2	0.307956	0.814098	0.166667	0.5	0.7
12	1.0	0.248131	0.233333	0.333333	0.35	0.458318	0.544233	0.477211	0.458318	0.544233	0.2	0.213432	0.707670	0.166667	0.5	0.8
13	1.0	0.273695	0.233333	0.333333	0.35	0.325191	0.319101	0.286454	0.325191	0.319101	0.2	0.267945	0.714950	0.166667	0.5	0.8
14	1.0	0.302538	0.233333	0.333333	0.35	0.481832	0.542328	0.461721	0.481832	0.542328	0.2	0.289451	0.785981	0.166667	0.5	0.8

Fig. 7: Code file ~ test file predicted relationship matrix

The Top 5 Recommended Test Files for the code – code1.c are

File Name	Relationship Score
Test1.java	1.35505
Test2.java	1.105023
Test99.java	1.099531
Test45.java	1.099451
Test11.java	1.098657

Fig. 8: A sample output based on the predicted matrix

2. CONCLUSION

Recommendation System is widely used in many commonly used applications. Recommendation System used in Test Development process will help us to improve the test development process too. It may turn out that, the recommendation from a single algorithm is not enough or is quite accurate. We use multiple algorithms for matrix factorization to get predicted values, we use other algorithms for prediction as well. The output from all of the algorithms are polled and the best group is selected.

3. REFERENCES

[1] Recommender Systems Handbook - Francesco Ricci, Lior Rokach and Bracha Shapira
[2] Recommender Systems An Introduction - Dietmar Jannach, Markus Zanker, Alexander Felfernig, Gerhard Friedrich
[3] Pattern Recognition and Machine Learning - Christopher M. Bishop
[4] Real World Machine Learning – Henrik Brink, Joseph W Richards, Mark Fetherwolf
[5] Foundations of Machine Learning – Mehryar Mohri, Afshin Rostamizadeh, Ameet Talwalkar
[6] Hands-On Machine Learning with Scikit-Learn & TensorFlow – Aurelien Geron
[7] Personalization Techniques and Recommender Systems - Uchyigit Gulden, Ma Matthew Y
[8] Recommender Systems for Information Providers - Neumann, Andreas W.

BIOGRAPHY

Sudipto Nandan
Consulting Member Technical Staff,
Oracle India Private Limited, Bengaluru, Karnataka, India

