RecSys Competition 2016

Extended Abstract

Matteo Pagliari
Politecnico di Milano
Piazza Leonardo Da Vinci
Milano, Italy 20133
matteo.pagliari@mail.polimi.it

ABSTRACT

The application domain is a professional social network, where users look for open job positions. The idea of this paper is to explain the methodologies used during the competition to understand on which job users will looking for. The main goal is to predict on which job a user will click, based on the record of past jobs looked by the user. I start introducing simple techniques found in the literature with their results and I will arrive to the final approach to the problem. The experiments are not all satisfactory but they have been useful to arrive at the ultimate result.

ACM Reference format:
Matteo Pagliari. 2017. RecSys Competition 2016. In *Proceedings of RecSys Competition, Milano, Italy, February 2017 (Milano 2017),* 3 pages.
DOI: 10.475/123_4

1 INTRODUCTION

The goal of the project is to recommend a list of 5 relevant items for each user starting from the interactions of the users with the application and the information about users and items. The recommendations provided to the users must not contain jobs that an user already clicked on and jobs not active during the test period. The recommendations has to be done only for a subset of all the users and for the evaluation it is used Mean Avarage Precision over 5 elements (MAP@5):

$$ap@n = \sum_{k=1}^{n} P(k)/min(m, n) \tag{1}$$

$$MAP@n = \sum_{i=1}^{N} ap@n_i/N \tag{2}$$

where *P(k)* means the precision at cut-off k in the item list.

2 IMPLEMENTATION AND EXPERIMENTS

During the development of the project, I follow different methodologies in order to provide every times better recommendations to the users. The data provided by the application are items and users profile and the interactions made by the users. With an initial analysis of data, I observed that the interactions represents implicit feedback and I used this assumption during the development of the project. There is a column in this table called interaction type that I decide to do not consider because it's not useful for my purpose.

2.1 Top Popular

Top popular method recommends to all the users the 5 most popular items based on the interactions. I decide to give the same weight to all the type of interactions and I sort them based on the number of times that they are present in the interactions file. Then I provide to the target users the top five jobs that are active during the test period. This method is a very simple and it gives a score something more than random recommendations.

2.2 Collaborative Filtering

The next approach was collaborative filtering, in particular I started implementing item-item collaborative filtering. It is based on the idea that items clicked by the same users are similar. The first step of this method is to compute the similarity between all the items using as similarity function the Jaccard similarity. The second step is the recommendation stage in which it uses the most similar items to a userfis already-rated items to generate a list of recommendations calculated with a weighted sum. For the computation of the score of the items, the best result of this method are obtained using a number of similar items between 50 and 200. After some attempts, I obtained the best evaluation using the 200 items. For the users with zero interactions I recommend the top popular items. The result obtained with this technique was better than the Top popular one. For the implementation of this algorithm, I used a library, *graphlab* that works very fast.

2.2.1 *CF with weights.*

Since some items have just one click on them, I modified the final score of the previous method using a weight. The idea of this weight is to give more importance on items with a good number of clicks and less importance on items with few clicks. For example, if an item similar to a chosen one has a certain score but it has few click on it, the effect of the weight is to reduce a lot its score. The consequence is that an item, less similar but more popular to a chosen one than another more similar, will can be clicked in the future by an user. I also add another weight based on the timestamps of each item giving more importance on items created recently. I applied a normalization between zero and one over the timestamp and I multiplied each item for its normalized timestamp. The idea is that a new job can have more appeal than an older one, so its score should be increase.

This is the basic formula:

$$score_{new}(i) = score_{cf}(i) * weight_{count}(i) * weight_{timestamp}(i) \tag{3}$$

where i is the recommenden item for a certain user.
From these new modifiec scores, I select the 5 top five elements. These intuition gives me a good improvement. I tried to add weights based on the country under the assumption that an user can probably prefers a job closer to his/her city but this idea didn't lead to me to a better result. I applied this idea also for the carrer level, I gave more importance to jobs near to the career level of the target users. Also in this case, I was not able to reach an improvement.

2.3 Content-based

Having a huge number of information for each items, I decide to use a Content-based approach on the items. I modified the data provided by an one-hot encoder that transforms categorical features in a format that works well with Machine Learning techniques. This process creates a large matrix of zeros and ones that I used to calculate the similarity between items. The first step of this method is to compute the similarity between all the items using as similarity function the Jaccard similarity. The second step is the recommendation stage in which it uses the most similar items to a userfis already-rated items to generate a list of recommendations calculated with a weighted sum. For the creation of the matrix a use a library for manage sparse matrix. I did a lot of attempts using different combination of information but this method works very slow and it gives to me worse result than the CF.

2.4 Alternative Least Square

ALS is used together with Matrix Factorization to give a better optimization on the computation of the loss function than the common Stochastic Gradient Descent. ALS works also better than SGD in case of implicit dataset. The parameter necessary to tune the algorithm are *alpha* that is useful to determine the importance of the interactions, the number of lusers and items latent *feature vectors*, a *regularization term* in order to avoid overfitting and the number of the *iterations* of the algorithm. Initially, I created the user-item matrix filling 1 cells if there is at least one interactions between an user *u* and a itemi. Later, I improved the algorithm filling the matrix with the number of interactions between the user and the item. I tried to implement this technique by myself but it spent a lot in time on the computation, so I decide to use a library called *implicit* that was very fast. Since the matrixes are very sparse, I decided to use a library called *scipy* to manage this type of data. Also in this case for the users that have no interactions, I recommend the Top Popular items. I did many attempts in order to find the best parameter and at the end, I have reached better result than the CF. Here there are the optimal values of the parameters:

$$alpha = 40$$
$$factors = 300$$
$$regularization = 0.01$$
$$iterations = 20$$

2.5 Hybrid

Since CF and ALS were the best technique that I have found for the problem, I wanted to use an hybrid approach to combine their results. The first attempts was very simple and it works picking at random items from the recommendations provided by these two methods giving more chances to the ALS that worked better. The results were not very satisfactory, so I decided to try something better. The first thing was to collect a bigger number of recommendations from these two methods and for each item recommended its score. I normalized the score provided and I combined the score of the common items between CF and ALS. I summed the scores of the common elements using different weights since I reached a good result. I also combined this idea with a content based approach on the items. From the items data, I used tags and titles and I separately used them for calculate the similarity between items. Not all the items have tags and titles and in these cases I didn't compute the similarity. The similarity function used for this purpose is the Jaccard similarity. If an item is only recommended by the ALS, its score remains the same provided by that algorithm. For each user I collect the most similar items based on tags and titles and their scores and I combined them with the recommendations provided by the CF and ALS. I sum up these new scores giving to each type of scores a different weight. Here I want to present the steps for the computation of the new scores. Since the ALS is the algorithm that works well, I decide to give a big importance, so the starting score has provided by this algorithm.

$$score_0(i) = score_{als}(i) \tag{4}$$

If in the CF recommendations there is a common element with ALS recommenadtion, I sum up the score using some weights.

$$score_1(i) = alpha * score_0(i) + beta * score_{cf}(i) \tag{5}$$

If also there is a common element in tags and title recommendations, I sum the scores using another weights.

$$score_2(i) = gamma * score_1(i) + delta * score_{tags}(i) \tag{6}$$

$$score_{final}(i) = omega * score_2(i) + theta * score_{titles}(i) \tag{7}$$

where *alpha, beta, gamma, delta, omega, theta* represent the importance of each type of score.
If an item is present in only one of the three last list of recommendation, its weights is adjusted following the equations in which is involved. The difficult part was to find the correct values in order to obtained good result:

$$alpha = 1$$
$$beta = 1$$
$$gamma = 1$$
$$delta = 0.9$$
$$omega = 1$$
$$theta = 0.5$$

3 RUN THE CODE

The source code is divided into some python files, each file is a part of the final algorithm and it represents one of the intermediate step presented during the implementation part that improves the result:

(1) The first file to run is *CF-basic.py* that generates recommendation based on a simple collaborative filtering.

(2) The next step is the file *CF-weighted.py* that brings the previuos CF and it indroduces the weights based on counts and timstamps.
(3) The hybrid model is composed also by the file *ALS.py* that produces recommendation based on ALS using the parameters previously introduced.
(4) *tags.py* computes the most similar items of all the items found in the interactions file.
(5) *tags-recommedantions.py* creates recommendations based on the similarity between tags.
(6) *titles.py* and *titles-recommendations.py* do the same operations done for the tags but for the titles.
(7) The file *Hybrid.py* contains the final model that computes the new scores using the result of all the previous file.

I also add an extra csv file called *zero-int-users.csv* that contains the list of all the target users with no interactions for retreving in a simple way this type of users.

4 CONCLUSIONS

The techniques that gives me the bette result is an hybrid approach using the recommendations provided by CF, ALS, tags and titles from items profile. All the result obtained before the final approach have been useful to arrive at this improvement. In the table below, you can find the diffent evalutation obtained using the *MAP@5* that gives to me an improvement at every step.

Table 1: Evaluation algorithms

Algorithm	MAP@5
Top Popular	0.000245
CF	0.00990
CF with weights	0.01245
ALS	0.01337
Hybrid	0.01456

A APPENDIX

A.1 Introduction

A.2 Implemetation

A.2.1 Top Popular.

A.2.2 Collaborative filtering.

CF with weights.

A.2.3 Content-based.

A.2.4 Alternative Least Square.

A.2.5 Hybrid.

A.3 Run the code

A.4 Conclusions