# Recommendation System For Online Coding Platforms


## Team Members

| | |
|---|---|
| Manjunath B. B. | 01FB16ECS195 |
| Nakshatra Yalagach | 01FB16ECS218 |
| Hrishikesh S. | 01FB16ECS139 |
| Lakkireddy Sai Chaitanya Reddy | 01FB16ECS174 |


## Outline

Online coding platforms have gained wide popularity through the recent years. Our recommendation system aims to improve user experience when the user interacts with online coding platform. Everytime the user visits the platform user data is collected based on user activity. The engine recommends next problem to the user based on the user profile and user's history.

## Problem Statement

1. *Recommender system through collaborative filtering.* For each user, recommender system recommend the problem based on how similar users solved the problem. For each user, user activity data is collected when the user visits the platform and based on the user activity data of similar users the problem will be recommended.

2. *Recommender system through content based filtering.* For each user, recommender system recommend the problem based on the user profile and the user's history. The next problem will be recommended based on the user profile which contains preference (type of the problem or complexity of the problem the user wishes to solve) and the user history based on the previous activity of the user .

3. *Prediction of level type of new problem.* Build machine learning models to predict the level type of the new problem which enters the system. The new problem entering the platform will contain only the tags and not the level type. The models will predict the level type of the problem which will be used to recommend the new problem to the users.

4. Predicting the number of attempts taken by the user to solve the problem.

## Dataset Description

The dataset was obtained from kaggle. There are 3 datasets.

1. Problem_data.csv: This dataset consists of the problem id, level of the problem, points awarded and the problem approach(tags i.e. the types of approaches to the problem e.g. graphs, sorting etc). It has initially 6545 instances of data which drops down to 6411 after preprocessing.
2. User_data.csv: This dataset includes user id, the rank of the user, problems solved and user rating number of followers. It has 3572 instances of data.
3. Train_submissions.csv: This dataset includes user id, the rank of the user, problems solved and user rating number of followers. It has 155296 instances of data.

## Approach

## Recommendation system based on collaborative filtering

Collaborative filtering requires a large amount of information about a user to make accurate recommendations. Collaborative filtering methods are based on collecting and analyzing a large amount of information on users activity or preferences and predicting what users would like to solve based on their similarity to other users.

A key advantage of the collaborative filtering approach is that it does not rely on machine analyzable content and therefore it is capable of accurately recommending complex items without requiring an "understanding" of the item itself.

There are two collaborative filtering technique and will be explained in terms of the recommendation system that we are trying to build:

1) User-based Collaborative Filtering
This technique recommends problems based on similarity of users, by observing user behaviour on the problems they are solving and trying to find some correlation between user A and user B for solving N problems.
This method is done between each and every user and then recommending user A problems that he/she hasn't done but was done by user B, user C, user D etc. based on how similar they are to each other.
2) Item-based Collaborative Filtering
This technique recommends problems based on similarity of the problems based on problem similarity. We first compute problem similarity between each and every problem. Then we look at the problems solved by the user and provide the most similar problem to the one they previously solved.
For both methods, we are using cosine similarity in our approach.
#Add cosine similarity formula image and also add explanation for cosine similarity if it doesn't make it too long.
For a given user and the number of problems that user would like to solve, denoted by 'k', we will recommend 'k' problems using both approaches.
The big debate will be regarding which approach to follow.
From the perspective of a programmer, both methods are useful.
For example, if a user A wants to solve all problems related to one topic only, then item-based CF is more useful but if the user wants to improvise his coding skills and compete with people of the same skill as him, user-based CF has an edge over the former.
*Demerits of recommender system through collaborative filtering.*
Collaborative filtering suffers from
(a) Cold-Start : Systems need large amount of data on user to make accurate recommendations
(b) Scalability : Large computation is needed since most of the time, number of users are in million
(c) Sparsity : Most active users will have only rated small subset of overall database

## Recommendation system based on content filtering

Content based filtering, also referred to as cognitive filtering, recommends items based on a comparison between the content of the items and a user

profile. Now there are numerous ways available to build a user's profile we've selected a few of them pertaining to our dataset. We've a come up with 3 different approaches to this detailed below:

1. Recommending problems based on tags of problems and user's profile
   - Tags in the problems data are taken to form a bag of words.
   - Problem matrix created (shape: number of problem, number of tags) containing 1 at a position if a problem has that tag else 0.
   - Then for each user an unique profile is created (shape: number of problem, number of tags) by multiplying the number of attempts(which is zero if the user hasn't solved the problem) with the problem matrix, and columns of these profiles added and normalised resulting in a matrix(shape number of users, number of tags) indicating the probability of a user choosing a problem with a certain tag.
   - Whenever a user is queried with the system for recommendation, the system multiplies problem matrix with the user's profile whose row sums indicate the probability of user choosing that problem next
   - These rowsums are sorted to recommend the user the most probable problems and most k problems from this sorted probabilities are recommended to the user.

The problem with this method is most of the problems in our dataset have no tags so, a user who has solved problems with no tags will never be recommended the problems with tags. To increase this narrow probability we've come up with the method below

2. Recommending problems based on level_type of problems and user's profile
   - For each user a profile (shape: number of level_types) is created by adding the number of attempts taken for all level_types separately for all the problems previously solved by user
   - These profiles are normalised, to give a profile which indicate the probability of a user choosing the problem of certain level_type
   - Based on the sorted values of these profile values, the k most problems with highest probability are being recommended to the queried user

This algorithm recommends users based on level_type widening the range of possible recommendations. Bcoz most of problems in our dataset have level_type

The methods above recommends type of problems which user has felt to be difficult previously. To make them recommend the type of problems which were easier to user we make a new column in problem data which is inverse of number of attempts and use that in the above algorithms

Both of these above problems suffer with ‘cold start’ problem.
That a new user absolutely has no profile meaning that he won’t be recommended any problems, to overcome this we’ve come up with method below

3. Clustering users and recommending problems solved by other users in same clusters

- We take into account of user ratings of all users and we cluster them into 10 different groups based on their user rating.
- All problems solved by each user in all the groups are clustered together
- A queried user is recommended problems belonging to the same cluster

It solved the ‘cold start’ problem by recommending a new user with just his rating rather than trying to create the user profile

*Demerits of recommender system through content based filtering.*
The disadvantage of content based recommendation systems is they never recommend problem of different level_type or tags that the user has not tried out before, restricting the user recommendation to a very narrow set of problems.
To overcome this problem we’ve done collaborative filtering

## Prediction of level type of new problem

Machine learning models are built to predict the level type of the problem which will be used to recommend the next problem to the user and also to

predict the number of attempts taken by the user to solve the next problem. Prediction of the attempts is explained in the next section.

In order to make accurate recommendations we built various models to predict level type. We are mainly considering two parameters counts which is the number of people who have solved a particular problem and tags like greedy, dynamic programming etc. Tags in the problems data are taken to form a bag of words. Based on the similarity of tags and count the level type of the problem will be predicted.

Using count as the parameter
x train consists of count.
Y train consists of level type.

1. SVM: using count gives an accuracy of 16.87%
2. Logistic: using count gives an accuracy of 17.34%
3. Naive Bayes- Bernoulli classifier: using count gives an accuracy of 16.73%

Using count and tags as the parameter
x train consists of count, tags.
Y train consists of level type.

4. Logistic: using count and tags gives an accuracy of 24.06%
5. KNN: using count and tags gives an accuracy of 45.73%
6. Random forest classifier: using count and tags gives an accuracy of 48.47%

*Analysis*
The initial models built only using count (number of users who solved a particular problem) as parameter gave lesser accuracy. Hence we decided to use both the count and the tags(Brute force, hashing) as parameters.

SVM, Naive Bayes- Bernoulli classifier will give lesser accuracy because

- They give better accuracy for binary classifiers while our dataset contains multi class data.
- Naive Bayes has strong feature independence assumptions.

- It does not perform well for large dataset.
- When dataset has noise.
- It is computationally expensive.

To overcome the demerits of these models we built KNN and Random Forest Classifier which give higher accuracy for the following reasons.

KNN on the other hand requires no training prior to making real time predictions. This makes the KNN algorithm much faster than other algorithms that require training e.g SVM, linear regression, etc. When k=1 the accuracy is maximum as major clusters are not formed and prediction is based on similarity of points.

Random forests is a set of multiple decision trees . Random forests is considered as a highly accurate and robust method because of the number of decision trees participating in the process. It has lower classification error and better f-score. It does not suffer from the overfitting problem as it creates trees on random subset which prevents overfitting. The main reason is that it takes the average of all the predictions, which cancels out the biases.

## Predicting the number of attempts taken by the user to solve the problem.

- For each user a profile (shape: number of level_types) is created by adding the number of attempts taken for all level_types separately for all the problems previously solved by user
- For each of these profiles the mean of the columns is calculated, to give a profile which indicate the probability of a user choosing the problem of certain level_type
- Whenever a prediction is asked for a certain user and problem, the user's profile of user queried looked up for the value in it for the level_type of the problem queried. And that value is returned which
- is the predicted number of attempts to be taken by user to solve that problem

- We've done an in sample validation of all the users and problems present in train submissions dataset. The RMSE value turned out to be as less as 0.9869

## Conclusion

- Recommender systems have great value in recommending problem to users. It can be quite useful in finding novel and serendipitous recommendations.
- Both of these content information and item based collaborative filtering method are then used together in prediction process.
- In the beginning, current recommendation systems and main theoretical issues behind them are generally introduced.
- We can recommend two best models to predict level type of new problem.
    1. Random forest classifier which gives an accuracy of 48.47%
    2. KNN which gives an accuracy of 45.73%
- Model for predicting the number of attempts taken by the user to solve the problem. The RMSE value is 0.9869