Matching is Difficult!

We changed the algorithm for realistic matching

Nascenia

Table of Contents

OVERVIEW

The **Biyeta** web and mobile app contains a matching algorithm which matches the user's preference with other people's profile, and displays suitable results that corresponds to that individual's user preference. This product was launched in Bangladesh during 2016 with a view to solving problem of finding the right bride or groom for marriage. Being a predominantly Muslim country, Bangladesh has some unique characteristics for choosing partners. Unlike Western countries, having a premarital physical relationship in the Bangladeshi social setting is a rarity, and highly discouraged. Even having an affair is scorned upon.

It leaves almost no option for a potential marriage candidate to find the kind of partner (s)he dreams about. In most cases, they need to depend on their parents, relatives or neighbours, to find a right life partner. With a view to solve this social problem, Biyeta came to the aid by launching a state of the art platform for finding the right partner. Here, while registering in the system, a user is asked a specific set of questions.

According to the individual's preferences the web user inputs the type of girl or boy the user is looking for based on skin color, height, weight, his/her location, religion, caste and job. The user also answers certain miscellaneous questions; for example, a male user looking for a suitable girl may be asked whether the girl he is looking for should wear hijab or not.

Based on this input, the algorithm assigns some arbitrary points for these attributes, and performs direct matching to generate an overall score from which a percentage is calculated. For example: for direct religion matching, the highest score achievable was given 150 points, for skin color it was 60, for location it was 100, and so on. This way, other values were also assigned based on the psychology of Bangladeshi people. The points reflect the degree of importance given to certain aspects, such as religion and location, which holds higher priority for marriage prospects. If the user's preferences match perfectly with a person's profile, he/she is assigned full points by 100%. For religion and caste, if the preference provided by the user partially matches with the matching user's religion and caste he/she is assigned with 50% of the total point.

PROBLEM

1. Issues with Point System

The problem with the algorithm is that the arbitrary full points that was assigned for each attributes is randomly given without any statistical validity. The mindset of people from different locations of Bangladesh varies over time, and it also varies based on socio-economic status. The value assigned reflects what the developer and the product owner deem as the possible case scenario in Bangladesh, and thus introduces favoritism in the result set.

For example, I want a girl who should be educated and should belong to my religion. Other factors are not a priority for me. But my other preference attributes match perfectly with other girls who do not meet my uncompromisable requirements of education and religion category. Without meeting my uncompromisable requirements, they still ended up obtaining a cumulative higher score and came at top of my list according to the algorithm.

2. Rigidness

Second problem with this algorithm is that, once the result is calculated for a user, over time, it stays the same. Since we hard coded the assigned points, the user who sits on top of the list will be on top of my list forever unless I change my preference.

3. Slowness due to performance issue

Thirdly, this algorithm speed of generating results was slow because every time the user hits the matching page, the algorithm takes all the users of opposite gender and calculates the same result that was generated on previous sessions, over and over again.

SOLUTION

In this section, we will explain how biyeta's match-making algorithm works under the hood. Biyeta.com uses a variant of knowledge-based recommendation with Pearson's correlation coefficient. We wanted to leverage the power of machine learning to make our algorithm a success, and in the end decided to go with K-Mean clustering. In order to boost application's performance, we used batch processing techniques with distributed caching. We aim to apply K-Mean as a viable model for knowledge-based recommendation systems, exploring how K-mean coupled with batch processing and caching can improve the performance of match-making with superb speed-up and reasonable accuracy.

1. Batch Processing: A faster approach for designing data intensive application

We started thinking how we can use batch processing to speed-up our recommendation systems as our systems exhibit read-heavy patterns. The particular moment when we felt overjoyed was when we realized that people do not edit their profiles everyday. This is the precise point in time when we realized that we could sever recommendation for similar profiles from cache. When the cache would expire, we could recompute fresh recommendations using batch processing and update the caching layer accordingly. As a result, users would almost always be served up from cache. But the caching layer needs to have updated somehow. In order to populate the caching layer, we decided to run the 'K-Mean clustering' once every 24-hour.

2. Redis: Lighting fast in-memory cache

The recommendation systems for Biyeta is a read-heavy application, where most users spend their time trying to find new recommendations. As most databases may need to seek disks for retrieving blocks of data, one way to speed-up such read-heavy systems is to use caching. Our recommendation system uses Redis extensively to accelerate application response times. User related details, such as profiles, preferences, search results, etc., are kept in Redis so that we don't need to access database at run time while serving users requests. When a user updates their user details, caches are updated accordingly.

3. MongoDB: Document-Oriented NoSQL

MongoDB stores data in JSON-like documents that can vary in structure. Related information is stored together for fast query access through de-normalization. Our REST API is backed by MongoDB. The reason why we used MongoDB is that we need to de-normalize our schemas to make our application faster. For example, If you want to store a user details in MySQL, you need to either perform multiple queries or perform a join between the users table and its subordinate tables. In the JSON representation, only one query is sufficient.

4. Knowledge-based recommendation

Most commercial recommendation systems in practice are based on the user ratings as the only knowledge sources for generating recommended items for their users, and as a result they don't need to maintain any additional information about the items being recommended. This type of system is known as collaborative filtering. Although collaborative techniques are widespread in industry, these approaches are not to the best choice for our match-making recommendations. The problem is that typically we want to stay married for a lifetime-we don't get married every year. Also, time spans play an important role as our old ratings might be inappropriate for recommendation.

One way to tackle the aforementioned problems is to use Knowledge-based recommender systems as these systems don't depend on any rating data. Also, by using only similarity measures, such as Pearson's correlation coefficient, we will be able to get pretty decent recommendations.

5. Scoring: Pearson correlation coefficient

Our system uses Pearson's correlation coefficient in order to determine a set of similar users for a given user. It measures how well two sets of data fit on a straight line. Though the formula for this is more complicated , it tends to give better results in situations where data sets aren't well normalized .The Pearson correlation coefficient takes values from +1 (strong positive correlation) to -1 (strong negative correlation). In our system, each user profile is evaluated according to a predefined set of features, such as age, weight, height, profession, etc., that provide an aggregated score. A predefined scoring table is used to find the final aggregated score. we also introduced a decay function based on user account's creation date so that new users get higher ranking in search results.

6. K-Mean Clustering: Sifting out unnecessary Data

Our recommendation algorithm uses k-means clustering algorithm to partition user profiles into smaller groups such that there is a high similarity between users belonging to the same cluster. When a user query has to be calculated at run time in order to fetch a set of matching profiles, the system first determines which cluster is the closed one and then sends all members of that cluster to the Pearson scorer . The performance of such an algorithm depends on the number of clusters and the respective cluster size. Though smaller clusters exhibit better runtime performance, when clusters are too small the accuracy of the system may degrade. One interesting aspect of using K-Mean clustering is that we can scale our algorithm across clusters of machine using Spark or Hadoop.

CONCLUSION AND FUTURE WORK

The work performed in this project can be enhanced in many possible ways. The recommendation systems can be improved in the following ways:

- The current algorithm can be replaced with better ones that can accelerate
- The computing performance even further. One reasonable solution could be using something like Min-Hashing. Also, we can improve our algorithm by monitoring how users engage with other users, and then feed that activities into a neural network-may be something like deep learning.
- One way to boost revenue for Biyeta is to be able to predict the likelihood that a user will become a paying customer. We can use machine learning classifiers , such as Decision Tree to predict that likelihood.
- In order to make our recommendation systems more fault-tolerant and cloud-native, we could try introducing service discovery techniques. We also need to introduce message-queue to make it more reactive and responsive.
- Last but not least, we may need to shard our database and caching layer, and may even need to implement distributed K-Mean clustering using Hadoop, Spark or something like that in the future to cater our users.

.