UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ADRIANO CARNIEL BENIN

A Comparison of Recommender Systems for Crowdfunding Projects

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Science

Advisor: Prof. Dr. Bruno Castro da Silva

Porto Alegre
June 2018

“The question of whether computers can think
is like the question of whether submarines can swim.”
— EDSGER W. DIJKSTRA

ABSTRACT

Recommender systems have been a popular research topic in the field of Machine Learning and are of great commercial interest for many businesses. This work aims to implement and evaluate different recommendation strategies in the context of a crowdfunding platform. The main approaches used in modern commercial applications will be considered in this work and several performance metrics will be measured in order to evaluate each implemented recommendation system. The crowdfunding platform in which we evaluate these different recommendation algorithms provides many interesting challenges, such as the transient nature of projects and the low amount of information available for rating prediction. Results from production tests with real users will be provided in order to compare each approach with a chosen baseline.

Keywords: Machine learning. recommender systems. AI. crowdfunding.

Uma Comparação de Sistemas de Recomendação para Projetos de Financiamento Coletivo

RESUMO

Sistemas de recomendação são um tópico de grande interesse de pesquisa na área de aprendizado de máquina e são de grande interesse comercial para vários negócios. Esse trabalho propõe implementar e analisar diferentes estratégias de recomendação para uma plataforma de financiamento coletivo em particular. As principais abordagens empregadas em sistemas comerciais modernos serão analisadas, e várias métricas de performance serão avaliadas para cada uma delas. A plataforma de financiamento coletivo na qual avaliamos as técnicas de recomendação selecionadas apresenta diversas características que dificultam a implementação de sistemas de recomendação, tais como a natureza transiente dos projetos e a pequena quantidade de dados disponíveis para prever as avaliações dos usuários. No final, serão apresentados resultados comparando cada tipo de sistema de recomendação implementado em um sistema de produção com usuários reais.

Palavras-chave: aprendizado de máquina. sistemas de recomendação. IA. financiamento coletivo.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

RS	Recommender System
CB	Content-based
CF	Collaborative filtering
MSE	Mean-squared error
MAE	Mean absolute error

CONTENTS

1 INTRODUCTION

Since time immemorial, people have relied on recommendations from many sources to base their decisions on, be it spoken word, magazines, travel guides and so forth. The decision to buy a new book, for example, is often based on the opinions of a close group of friends; employers count on recommendation letters for recruiting; and when selecting a movie to watch, people rely on movie critics they have read. Recommender systems aim to augment this social process in order to assist people in sifting through an ever-growing list of movies, books and all sorts of items. Recommender Systems (RSs) are widely used in the industry today to provide useful suggestions to end-users in a completely automated manner. They are ubiquitous in modern e-commerce Web sites (SCHAFER; KONSTAN; RIEDL, 2001), where new products can be recommended based on a customer's interests and preferences, and in many other fields such as movies (Netflix) and music (Spotify). Its importance cannot be overstated: the effectiveness of targeted recommendations, as measured by click-through and conversion rates (i.e., the ratio of users who clicked on a specific link and the total number of users who viewed a page, and the ratio of users who performed the desired final action—such as buying the product—and the number of users who viewed the page, respectively), far exceed those of untargeted content (LINDEN; SMITH; YORK, 2003). By customizing recommendations for each user, search effort is greatly reduced, leading to greater customer loyalty, higher sales and advertising revenues, and better targeted promotions (ANSARI; ESSEGAIER; KOHLI, 2000).

A recommender system typically focuses on a specific type of "item"—the general term used to denote what the system recommends to users. The sort of items recommended by such systems can vary tremendously. Apart from the most well-known examples such as Amazon, which recommends products, and Netflix with movies, RSs are also used to recommend social connections in platforms such as Facebook, as well as news in news aggregators like Google News. Table 1.1 shows a list of recommender systems used in real-world applications. Some of these systems will be discussed in further details in Chapter 2.

The mathematical objective of the recommendation problem can be formulated in a number of ways. Aggarwal et al. (AGGARWAL, 2016) propose the following main models/objectives:

- *Prediction*: In this approach, we try to predict the rating value for a user-item combination; in particular, given a list J of items and a history of past ratings given by

Table 1.1: Example of recommended products by various real-world applications

System	*Product*
Amazon.com	Books and other products
Netflix	DVDs, Streaming Video
Jester	Jokes
GroupLens	News
MovieLens	Movies
last.fm	Music
Google News	News
Google Search	Advertisements
Facebook	Friends, Advertisements
Pandora	Music
YouTube	Online videos
Tripadvisor	Travel products
IMDb	Movies

Source: (AGGARWAL, 2016)

user i to various items $J_i \subset J$, we estimate the ratings that this user would give to any other items $j \notin J_i$. The training data, based on which the RS will be trained, is given by an incomplete R_{MxN} matrix, where M is the number of users and N the number of items. In this matrix, each entry (i, j) represents the rating given by user i to item j. This matrix is extremely sparse in most cases since most users only rate a very small subset of the available items. The observed values, which indicate user preferences for items, are used to train a model which will try to fill in the missing or unobserved ratings in the matrix. Since we have an incompletely specified matrix of values, this problem is sometimes referred to as the *matrix completion problem*.

- *Ranking*: For many applications, it is not necessary to obtain absolute numerical values for the predicted ratings. Instead, we are more interested in obtaining the top-k items to recommend to a user, or the top-k users to promote a particular item. Both methods are exactly analogous, with the former being much more common in practice. This problem is often referred to as the *top-k recommendation problem*.

The first formulation of the problem is more general since it is possible to derive the second one by predicting ratings for every item and then ranking the predictions. However, in many cases, it is easier and more practical to solve the ranking problem directly.

Along with the definition or objective of the problem, it is important to define the desired operational and technical characteristics of recommender systems. The main desired properties of any recommendation algorithm are as follows:

- *Relevance*: The main goal in recommender systems is to provide recommendations that are relevant to the user at hand. Users are much more likely to consume items which they find interesting and are tailored for them.
- *Novelty*: Users are much more engaged when confronting items they have not seen in the past. Recommending the same items over and over, however good the recommendations may be, is not a good strategy.
- *Serendipity*: Serendipity in recommender systems happens when recommendations are somewhat surprising or unexpected to the user, as opposed to obvious recommendations. It differs from the concept of Novelty in that it does not simply recommend items the user did not know about before but allows the user to discover entirely new areas of interest. For example, recommending a new spicy Mexican restaurant to someone who regularly visits Mexican restaurants may be novel, but it is not surprising at all. Recommending a Thai restaurant instead might allow the user to discover a whole new interest, increasing sales diversity and user satisfaction.
- *Diversity*: Recommender systems usually suggest a list of top-k items to the user. If all the recommended items are similar, there is an increased chance that the user will not like any of the items. On the other hand, if we show diversified items there is a higher chance that the user will like at least one of them.

Additionally, we can define the main *goals* of the system as a whole from the perspective of the service provider. There are many reasons why service providers may wish to invest in RSs (RICCI et al., 2010):

- *Increase sales*: Arguably the most important goal for a commercial RS, this can be achieved because recommended items are more likely to suit the user's needs than a generic non-personalized item list. In general, the primary goal of a RS is to increase the conversion rate, i.e., the number of items consumed in relation to the number of page views.
- *Diversify sales*: In a movie rental system such as Netflix, for instance, the service provider is interested in renting all of its catalog instead of just the most popular movies. However, advertising relatively unknown movies to all its user base could be risky. A RS can alleviate this problem by providing different recommendations for each user, thus increasing the effectiveness of the advertisement.
- *Increase user satisfaction*: By providing interesting and relevant recommendations,

RSs can improve the experience of the user with the application, therefore increasing the user's evaluation of the system.

- *Increase user fidelity*: The more a user interacts with a RS the more refined its user model becomes, giving them the feeling of being recognized as a valuable visitor.
- *Understand user needs*: The output of a RS can be used for many other purposes such as improving the management of an item's stock or creating promotions targeting specific user segments.

1.1 Motivation

The objective of this work is to develop, evaluate and compare different state-of-the-art approaches for building RSs in the context of a particular crowdfunding platform—Catarse—of which the author is a developer.

Launched in January 2011, Catarse was the first crowdfunding platform for creative projects in Brazil. With over 7000 successfully financed projects raising R$77m from 480.000 people, it is currently the largest national platform of its kind. It works similarly to most crowdfunding platforms: the project owner presents his or her idea for a project which they wish to be funded and specifies the required investment as well as the cutoff date for the project while offering rewards for those who back it. Projects are divided into 3 main categories: all-or-nothing, flexible, and recurrent. In the first type, projects are available for backing (i.e., to start receiving pledges from supporters) up to 60 days, and the project owner only receives the raised amount if the project's goal (total required amount of money for developing it) is met; otherwise all the money is returned to its original backers. On flexible projects, the owner receives the raised amount whether the goal was reached or not. Recurrent projects are subscription-based and the owner can collect the money monthly.

Catarse's business model relies on charging a fixed percentage of the total pledged amount from successful projects, even if this amount is greater than the original goal. Failed projects, i.e. projects that did not meet their goal by the deadline, represent a major financial cost to Catarse since transaction fees involved in the refund process are fully absorbed by the company. Therefore, we expect to benefit from RSs in two ways: first, by increasing the average pledged amount per project, which can be achieved by presenting appropriate projects that are tailored for the user; and secondly, by increasing project dis-

coverability we expect that more projects will be successfully funded and fewer projects will fail. As will be discussed in Chapter 4, we will evaluate different RS approaches according to how well they achieve these objectives.

As of April 2018, over 1500 projects in 18 different categories are online and available for backing in Catarse. It would be unreasonable to expect users to browse through all projects before deciding which ones to back. For this reason, some sort of ranking is fundamental when showing projects to users, and there is no doubt that the choice of this ranking method will greatly affect conversion rates and successful funding of projects. At the time of this writing, several ranking strategies are in use. First, on the home page projects are shown ordered by popularity, defined here simply by the pledged(contributed) amount in the last 48 hours. Another section of the website shows which projects were backed by particular Facebook friends of the user, for those users who opted to connect their Facebook account to their Catarse account. Finally, on the *explore page*, we also have filters for displaying projects expiring soon. It should be noted that none of these sections of the Catarse website, with the exception of the backed by friends section, display projects that are tailored for each specific user: every user will see the same projects in the same order.

Given the above properties and goals of Catarse, project recommendation can be considered a crucial aspect of Catarse's business model. We can expect any marginal improvement in this regard to be of great benefit to the company and its users. Therefore, a user-tailored recommendation system is fundamental to stay competitive in this market and is expected to greatly increase user satisfaction, both for backers who will be able to more easily find projects that are interesting to them, and for project owners who will receive more visits to their projects due to increased discoverability.

Catarse's dataset is unique in a variety of ways that makes collaborative filtering (CF) techniques (explained later in Chapter 3) hard to apply. First, unlike regular e-commerce Web sites such as Amazon, where products stay available for many years, crowdfunding projects have a predetermined cutoff date, after which it is no longer possible to make a pledge. It makes no sense to recommend expired projects, limiting our possible recommendations to online projects only (see Section **??** for more details on the terminology employed in this work). Another challenge is the fact that the majority of Catarse's users only back one project, thus making it hard to collect enough data for CF methods to work properly. One typical characteristic of classic CF systems is that they may have to be retrained after every new interaction; this imposes scalability challenges

when implemented on highly dynamic data, such as the training data of Catarse. On the other hand, content-based methods (discussed in Chapter 3) allows us to train our model with the whole dataset in order to extract backer-project features only once, and later use this model to search for online projects with the highest backing probability for the current user. However, such methods also have drawbacks, as discussed in Chapter 3. These challenges motivate us to search for methods that are adequate to our specific needs.

1.2 Goals

This work aims to implement and compare the main computational strategies for building RSs and apply them in the context of a particular large-scale Crowdfunding website. Several metrics will be used to determine the best approach, which will then be deployed in a production environment. Our primary goal is to find a strategy that is at least better than the current-used popularity ranking, which is usually hard to beat in realistic scenarios.

1.3 Definitions and Terminology

We present, below, some crowdfunding-specific definitions and terminology that will be used throughout this work:

- **Contribution, pledge or back**: financial endorsement by a user to a specific project;
- **Backer**: a platform user that contributed to a project;
- **Project Owner**: the user that created the project;
- **Successful Project**: a project that, by the end of its deadline, has reached its goal;
- **Failed Project**: a project that, by the end of its deadline, has not reached its goal;
- **Online Project**: a project that is available for backing and has not yet reached its deadline;
- **Reward**: something that is offered or promised to backers for contributing to the project;

1.4 Structure

This work is divided into 6 chapters. In the first chapter, the basic concepts of Recommender Systems are discussed, as well as the motivation and goals underlying this work. In the second chapter, we analyze a few existing applications that rely on Recommender Systems. On the third chapter, theoretical background for the employed methods is presented. The fourth chapter describes how we conducted our experiments. In the fifth chapter experimental results are presented and discussed. Finally, in the sixth chapter we summarize our conclusions and describe future work.

2 RELATED WORK

In this chapter, a few selected existing commercial applications of RSs will be discussed. For each application, the relevant underlying RS techniques (some of which will later be used in our own implementation) are highlighted and a brief explanation of how they were applied to that system (or how they differ from our use-case platform) is provided. Details on how each technique can be computationally implemented are discussed later on in Chapter 3.

2.1 GroupLens Recommender System

One of the first recommender systems to be developed, Grouplens (RICCI et al., 2010) was used for recommendations of Usenet news. It worked by collecting ratings from many Usenet readers in order to recommend new articles to users, a technique now known as Collaborative Filtering (CF). This technique will be discussed in more details in Section 3.1.

This basic approach was later extended to other settings such as ones for recommending books and movies; these were referred to as Booklens and Movielens, respectively. One important contribution of this research was the release of several datasets of user ratings to the research community, which were not easily available at the time. One of these datasets, Movielens, is still widely used in RS research to this day.

Collaborative filtering techniques such as the ones used by GroupLens remain one of the most popular approaches for building RSs to this day. In our implementation, this technique will be used to recommend projects to users by collecting implicit ratings from users; in the context of our crowdfunding project, the act of backing a project counts as a positive rating to that project.

2.2 Netflix Recommender System

Netflix recommendations are obtained through user ratings on a 5-point scale (recently changed to a thumbs up/down system) as well as implicit ratings, such as the action of watching a movie. One interesting aspect of their system is the quality of explanations provided to the recommendations. For instance, a list of recommendations can be pin-

pointed to having watched a specific movie. This approach can help users decide what to watch and also contribute to improving customer loyalty and retention.

The Netflix Prize contest was a competition organized by Netflix that ran from 2006 to 2009. Contestants were given a data set of (real-life anonymized) existing user ratings for different movies and were expected to outperform Netflix's own recommendation algorithm, Cinematch. This contest was responsible for many notable contributions to recommendation systems' research, such as advancements in latent factor models. Many of these contributions were applied in the algorithms used in this work.

2.3 Kickstarter Recommender System

Kickstarter is the world's largest crowdfunding platform and works very similarly to Catarse, making it a great reference for our implementation. Kickstarter's initial implementation of its recommender system used a Collaborative Filtering approach, written entirely in MySQL. Later, Content-Based recommendations (discussed in Section 3.2), which are based on the similarity of the text, were used by Kickstarter and implemented via a method called Latent Semantic Indexing (LSI). Both these approaches were later combined into the current Hybrid system. For the initial tests performed by Kickstarter, the "popular projects" ranking was used as a baseline for comparing new algorithms. This ranking sorts projects by their total pledged amount in the last 24 hours.

Due to the similarity between Kickstarter and Catarse's platform we can expect that a similar RS implementation could work equally well. In fact, in our work we evaluate many of the techniques used by Kickstarter, such as CF-based RSs, CB-based RSs, and a hybrid approach. Similarly to Kickstart, our performance evaluation will also be (partially) based on comparisons to the existing popularity metric. However, some key differences between the platforms must be taken into account when designing a RS for Catarse. Specifically, flexible projects, which do not exist on Kickstarter, must be taken into consideration due to their distinct funding and time characteristics and constraints. These considerations are further discussed in Chapter 4.

3 THEORETICAL BACKGROUND

In this chapter, the main concepts behind Recommender Systems will be reviewed. We will discuss the most common approaches used in the industry today and provide a detailed comparison of their general characteristics, as well as insights that are relevant to our particular application. For each general approach for constructing a RS, we give descriptions of the most used algorithms along with state-of-the-art techniques that were selected for use in the experiments that we conduct (see Section 4).

The basic idea behind any recommender system is to obtain a utility function to estimate user preferences towards an item. The meaning of this function will differ for each context; it could mean how likely a user will want to watch a specific movie or listen to a song, or the likelihood of buying a particular product. In our case, the goal is to find projects the customer is most likely to back given his backing history and other characteristics.

Methods for implementing Recommender Systems can be broadly divided into two categories: **Content-Based** (CB) and **Collaborative Filtering** (CF) methods (RAKESH; LEE; REDDY, 2016). The former utilizes user features and content features (describing items, such as relevant keywords) in order to recommend items to users. In collaborative filtering methods, user ratings are used in order to calculate similarities between users (user-based CF) or between items (item-based CF), which are then ranked in order to show the most relevant recommendations to each specific user by exploiting inter-user and inter-item relationships. How these similarities are computed and used to rank items is described in Section 3.1. Content-Based models also utilize user ratings in most cases; however, the model usually focuses on predicting the ratings of a single user, instead of attempting to discover inter-user relationships. These two methods are sometimes combined into what is known as Hybrid Recommender Systems. Hybrid systems can combine the strengths of various models in order to perform more robustly in a wide variety of settings.

In the next sections, mathematical models for these approaches will be provided, as well as a discussion of their main advantages and shortcomings.

3.1 Collaborative Filtering

Collaborative filtering (CF) is based on the principle that similar users will share similar interests. Assuming that ratings are highly correlated across various users and items, it is possible to predict a rating not yet given by the user. For example, if the algorithm identifies that user Alice and Bob have similar tastes, even if only Alice rated a certain item, we can surmise that Bob would give a similar rating as hers.

CF algorithms can be divided into two categories: **Memory-Based** and **Model-Based** approaches. Memory-Based CF algorithms, also referred to as *neighborhood-based collaborative filtering algorithms*, represent each customer by a N-dimensional vector of item ratings, where N stands for the number of available items, and where each vector component corresponds to the user rating of the corresponding item. These ratings can be obtained explicitly — e.g. star rating or "likes" — or implicitly — e.g. a user buying an item or listening to a song can be considered a positive rating. By collecting ratings from M users, we build a $R_{n \times m}$ matrix which is the starting point for Memory-Based CF. Each entry R_{ij} in this matrix corresponds to an observed rating by user i to item j, while missing ratings are represented with some sort of null value. For most applications, R will be extremely sparse. Our job is to try to predict the missing ratings; for this to be possible, most models focus on leveraging inter-item or inter-user correlations, that is, the similarities between items or between users. Memory-Based methods were among the earliest CF methods proposed. Although very simple to implement, these methods often have trouble with very sparse matrices, since they use the R matrix directly to compute similarities and make predictions, which is often very large. Neighborhood methods are discussed in Section 3.1.1.

In Model-Based CF methods, on the other hand, machine learning and data mining algorithms are used to develop a model based on the R ratings matrix that will be used to predict user ratings. Some examples include decision trees, rule-based models, Bayesian methods and latent factor models. These methods can solve some of the shortcomings of Memory-Based approaches, such as the need for large rating matrices since they use a pre-computed model instead of directly using the R matrix to make predictions, often by means of dimensionality reduction of the R matrix. Finally, both approaches can be combined into Hybrid Recommenders. An overview of these techniques is depicted in Figure 3.1.

3.1.1 Neighborhood Methods for CF

Neighborhood methods take rows or columns of R and compute a similarity value between them. If the rows correspond to users and the columns to items, we can obtain the similarity between two users u and v by computing the correlation between the rows u and v of R. This is usually referred to as *User-Based CF*. In the same fashion, the similarity between items i and j can be computed by taking columns i and j of R and calculating their distances, which is referred to as *Item-Based CF*.

For User-Based CF, given rows corresponding to users u and v in the R matrix, one common method to compute similarity between users is the cosine measure given by:

$$sim(u,v) = \cos(u,v) = \frac{u \cdot v}{||u|| \cdot ||v||} = \frac{\sum_{i \in I_{uv}} r_{ui} r_{vi}}{\sqrt{\sum_{i \in I_u} r_{ui}^2} \sqrt{\sum_{i \in I_v} r_{vi}^2}}$$

where I_u and I_v are the sets of items rated by user u and v respectively, $I_{uv} = I_u \cap I_v$ and r_{xi} is the observed rating of user x to item i. To get a rating prediction $\check{r}_{ui}$ given by the user u to item i, we aggregate ratings from the subset of users most similar to u, U'. One such aggregation function could be:

$$\check{r}_{ui} = n \sum_{u' \in U'} sim(u,u') r_{u'i}$$

where n is a normalizing factor. For large databases, User-based CF can be prohibitively computationally expensive. Its worst-case performance is $O(MN)$ where M is the number of customers and N is the number of items (LINDEN; SMITH; YORK, 2003); this problem can, sometimes, be partially alleviated by exploiting sparsity in the customer vector.

Item-Based CF works almost exactly the same as User-Based CF. One important distinction is that item-based CF retrieves recommendations directly from the similarity matrix and does not require R to be kept in memory. A major reason for the adoption of this approach is that in most systems users are much more numerous than items, leading to a significantly reduced similarity matrix when using item-based CF (SARWAR et al., 2001). We can redefine the cosine measure for this case like so: let U_i and U_j be the set of users who rated items i and j respectively, and $U_{ij} = U_i \cap U_j$ be the set of users who rated both items i and j, the similarity is then given by:

$$sim(i,j) = \cos(i,j) = \frac{i \cdot j}{||i|| \cdot ||j||} = \frac{\sum_{u \in U_{ij}} r_{ui} r_{uj}}{\sqrt{\sum_{u \in U_i} r_{ui}^2} \sqrt{\sum_{u \in U_j} r_{uj}^2}}$$

Similarly, we define the rating prediction for item-based CF, considering J to be the set of items j most similar to i, as:

$$\check{r}_{ui} = \frac{\sum_{j \in J} sim(i,j) r_{uj}}{\sum_{j \in J} sim(i,j)}$$

Item-based CF was first proposed by Amazon (LINDEN; SMITH; YORK, 2003). Each item purchased by the customer is compared to other items in the dataset in order to calculate a similarity metric. The original algorithm is shown below:

for *each item in product catalog, I1* **do**
  for *each customer C who purchased I1* **do**
    for *item I2 purchased by customer C* **do**
      Record that a customer purchased I1 and I2;
    end
  end
  for *each item I2* **do**
    Compute the similarity between I1 and I2;
  end
end

Algorithm 1: Item-to-item CF

In either case, for this to work large amounts of user data is required. This is known as the cold start problem: new users who have not rated many items yet will have a reduced recommendation quality. On the other hand, no information about the item itself is needed, making CF especially applicable to collections of hard-to-analyze items such as movies. Memory-based approaches also have the advantage of being able to add new items or users on the fly by simply adding a new row or column in R, without having to recalculate a model.

3.1.2 Model Based Methods for CF

Model based methods attempt to solve the scalability and sparsity problems of Memory-Based approaches by leveraging a latent factor model that captures the similarity between users and items. This is generally achieved by a matrix factorization method such

as Singular Value Decomposition (SVD), which is explained below.

The basic goal in SVD is to find a factorization for the user-item ratings matrix R that decreases its dimensions. In particular, we want to find item vectors q_i and user vectors p_u such that the dot product is the expected rating $\hat{r}_{ui}$:

$$\hat{r}_{ui} = q_i^T p_u$$

and we want to find, in particular, p and q such that the square error difference between their dot product and the known rating r_{ui} is minimized:

$$min \sum_{(u,i) \in R} (r_{ui} - q_i^T p_u)^2$$

In other words, for each row u in the R matrix (containing ratings of the corresponding user for every available item), we want to find a new vector p_u to describe this user such that $|p_u| < |u|$ and $q_i^T p_u$ gives a good approximation of the real ratings.

The above equation can be minimized using stochastic gradient descent algorithm (SGD), which works by assigning random initial values to the entries in the q_i and p_u vectors and iterating to reduce the error between each step. SGD is parametrized by a learning rate γ, which determines by how much the values should be updated between each step in the direction of the gradient of the objective equation.

Figure 3.1: Overview of collaborative filtering techniques.

CF categories	Representative techniques	Main advantages	Main shortcomings
Memory-based CF	*Neighbor-based CF (item-based/user-based CF algorithms with Pearson/vector cosine correlation) *Item-based/user-based top-*N* recommendations	*easy implementation *new data can be added easily and incrementally *need not consider the content of the items being recommended *scale well with co-rated items	*are dependent on human ratings *performance decrease when data are sparse *cannot recommend for new users and items *have limited scalability for large datasets
Model-based CF	*Bayesian belief nets CF *clustering CF *MDP-based CF *latent semantic CF *sparse factor analysis *CF using dimensionality reduction techniques, for example, *SVD*, *PCA*	*better address the sparsity, scalability and other problems *improve prediction performance *give an intuitive rationale for recommendations	*expensive model-building *have trade-off between prediction performance and scalability *lose useful information for dimensionality reduction techniques
Hybrid recommenders	*content-based CF recommender, for example, *Fab* *content-boosted CF *hybrid CF combining memory-based and model-based *CF* algorithms, for example, Personality Diagnosis	*overcome limitations of CF and content-based or other recommenders *improve prediction performance *overcome CF problems such as sparsity and gray sheep	*have increased complexity and expense for implementation *need external information that usually not available

Source: (SU; KHOSHGOFTAAR, 2009)

3.2 Content-based Filtering

In this approach, a vector description of each item is constructed using structured data or some sort of item presentation algorithms such as Latent Dirichlet Allocation (LDA) or TF-IDF, which are used to build structured representations of unstructured data such as unrestricted text fields. Along with the item descriptions, a user profile is created for each user. This profile can consist of many different types of information: a history of the user's interaction with the system such as page views, searches and purchases is often used to train the model without any explicit user input. Some systems may require the user to explicitly state their interests, however it may be very hard to get users to make this effort, rendering this approach very limited in practice. Item representations and the user profile are then combined to create a model that, given a user profile and a new, unlabeled item, predicts the rating for the new item. One simple approach, for instance, would be a nearest neighbor method that calculates the distance between this new item and items already rated by the user and classifies this new item according to the labels of its nearest neighbors.

As CB filtering focuses on item rather than user similarity, it avoids the cold start problem since little information about user preference is needed. For instance, if user Bob rated movie A but no other ratings are available for the movie, it is still possible to recommend other movies based on the attributes of movie A, such as genre keywords or release date. However, since only similar items to those already rated by the user will be considered, CB strategies tend to suffer from over-specialization (IAQUINTA et al., 2008). This is known as the serendipity problem. Another limitation of CB algorithms is that items are required to contain enough information in order to distinguish items the user likes from items the user does not like. For example, a dataset of songs where only the song name is available would not be enough to make good predictions based on this content only, however this would pose no problem for collaborative filtering methods which rely solely on user similarity. Finally, while CB methods excel in recommending new items, they are not effective when it comes to new users. This is because the training model requires a rating history in order to make robust predictions.

CB systems provide several advantages over CF approaches:

- *User independence*: CB systems do not rely on rating information from other users; this makes it possible to recommend items that were not yet rated by anyone else, based solely on the items characteristics.

- *Transparency*: Predictions can be explained by listing specific content features that caused an item to be recommended.

On the other hand, several shortcomings are present when compared to CF approaches:

- *User independence*: CB systems do not rely on rating information from other users, this makes it possible to recommend items that were not yet rated by anyone else, based solely on the items characteristics.
- *Limited content analysis*: feature extraction is limited in most cases, limiting the amount of information available to build the user model. For instance, performing a text analysis on a news article will often completely ignore multimedia information
- *Over-specialization*: Also called the serendipity problem, CB systems are, by design, only able to recommend similar items to those already rated by the user. This makes it impossible for the user to discover novel and unexpected items.

In the next section, we provide a detailed explanation of Gradient Boosting Trees, which can be used as a binary classifier to implement a CB system by considering a positive rating as belonging to the positive class and a negative rating as belonging to the negative class.

3.2.1 Gradient Boosting Trees

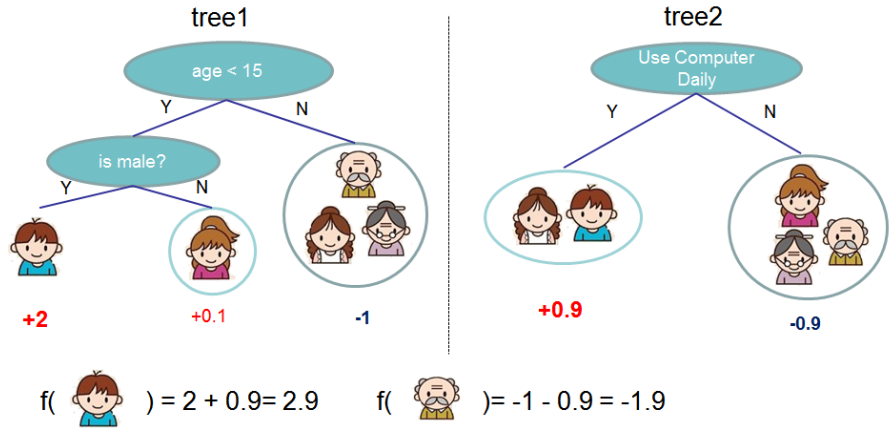
In recent years, tree boosting has become an increasingly popular method and been shown to give state-of-the-art results for many classification problems (LI, 2012). As in any supervised learning method, our objective is to train a model to predict a target variable y_i given features x_i. Boosting methods are characterized by combining many weak learners into a strong one in an iterative fashion. In this case, the model is an ensemble of trees, more specifically a set of classification and regression trees (CART). Unlike decision trees, where the leafs contain decision values, the leafs on gradient boosting trees contain a score $s \in \mathbb{R}$. Multiple simple trees are then constructed and the prediction of each tree is summed up to get the final score as explained on Figure 3.2. The model can then be defined as

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in \mathcal{F}$$

where K is the number of trees, f is a function in the functional space $\mathcal{F}$, and $\mathcal{F}$ is the set

of all possible CARTs (CHEN; GUESTRIN, 2016). The example in Figure 3.2 tries to predict whether someone will like computer games or not. We can use the above equation to compute the score for the boy, for example, by summing the corresponding leaf scores of each tree, i.e. the overall score associated with a given class is the sum of the scores (as predicted by each individual tree) for that class.

Figure 3.2: GbTree example



Source: (**??**)

The objective function can be written as

$$\text{obj}(\theta) = L(\theta) + \Omega(\theta)$$

$$\text{obj}(\theta) = \sum_{i}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

where L is the training loss function and Ω is the regularization term. L can also be written as the sum of the errors for all instances given by an error function l that takes as parameters the real class y_i and the predicted class $\hat{y}_i$.

One possible loss function would be the logistic loss function for logistic regression:

$$L(\theta) = \sum_{i} [y_i \ln(1 + e^{-\hat{y}_i}) + (1 - y_i) \ln(1 + e^{\hat{y}_i})]$$

To define the regularization function, we first define the tree function $f_t(x)$ for each tree t as

$$f_t(x) = w_{q(x)}, w \in R^T, q : R^d \to \{1, 2, \cdots, T\}.$$

where w is the vector of scores on leaves for some given input x, q is a function assigning each data point to the corresponding leaf, and T is the number of leaves. The regulariza-

tion function is then defined as follows:

$$\Omega(f) = \Omega(w_{q(x)}) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

The ensemble is trained by using an additive strategy. We need to learn the functions f that define the structure and leaf scores of each tree. This is done by fixing the trees we have already learned and, based on them, adding one new tree at a time. The prediction value at step t (t-th tree) for each instance i is given by:

$$\hat{y}_i^{(0)} = 0$$

$$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$$

$$\dots$$

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

We optimize the objective function to build the tree at each step:

$$\text{obj}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i)$$

$$= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant$$

After some algebra, the objective becomes:

$$\sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

where $g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$ and $h_i = \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$. By defining $G_j = \sum_{i \in I_j} g_i$ and $H_j = \sum_{i \in I_j} h_i$, with $I_j = \{i | q(x_i) = j\}$ being the set of indices of data points assigned to the j-th leaf, we can further simplify the objective function as:

$$\text{obj}^* = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$$

The above equation measures how good a tree structure is, with smaller scores representing better trees. Ideally, we could enumerate all possible trees and choose the one with the lowest score given by this definition. However, such a problem would be

intractable so in practice we optimize for one level of the tree at a time. This is done by splitting a leaf into 2 leaves using the Gain function below:

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma$$

the gain can be decomposed as the sum of the scores of the new left and right leafs minus the score of the original leaf. We can see that if the gain is smaller than γ we should not split the branch. This is the pruning step in the algorithm, after this, we can start building the next tree of the ensemble, up to the specified number of trees desired.

3.3 Hybrid systems

When used in isolation, all the previous methods present shortcomings that are hard to overcome. Hybrid systems are often used to solve these shortcomings by combining results from many distinct models. There are basically 3 ways of creating hybrid systems:

- *Ensemble design*: In this design, results from several algorithms are combined into a single output. This can be done in several ways, such as computing the weighted average of the various predictions or by multiplying every prediction. It's also possible to not directly combine the outputs, but instead, use the output from one algorithm as a feature for the next one.
- *Monolithic design*: In this approach, various data types are used as an input for a single generic algorithm. This can be done by, for example, modifying existing CB algorithms to take user rating matrices into account.
- *Mixed systems*: Like ensemble design, the output from multiple algorithms is computed to obtain a score for each item, however, instead of combining these scores, items from each algorithm are presented together side-by-side.

We can formalize the weighted average method from the ensemble design, which will be used in this work, by defining $R = [r_{uj}]$ as a $m \times n$ matrix of completely specified ratings, in which the unobserved entries of R are predicted by q different algorithms. For a set of weights $\alpha_1..\alpha_q$, the weighted hybrid method creates a combined prediction $\hat{R}$ as follows:

$$\hat{R} = \sum_{i=1}^{q} \alpha_i \hat{R}_i$$

Linear regression can be used to determine the optimal weights by minimizing metrics such as the *MSE* or *MAE*.

4 EXPERIMENTAL METHODOLOGY

The experiments we conducted consisted in implementing and evaluating three recommendation algorithms using the data available in Catarse's database. The chosen algorithms were: Collaborative Filtering with matrix factorization, a Content-Based approach using Gradient Boosting Trees, and a Hybrid system combining these two methods by using weighted averages. These algorithms were chosen due to their state-of-the-art performance and wide use in the industry, and also in order to test every major approach in RSs in the context of Catarse's environment. The existing popularity ranking was used as a baseline in our tests.

In the next few sections, we will describe how our training data was obtained and prepared for use in each RS implementation, as well as provide evaluation metrics that are commonly used in recommender systems in order to compare each approach. The main benefits and shortcomings of each metric will be presented and their applicability to our experiment will also be discussed.

4.1 Data Preparation

Two datasets were constructed from Catarses's production database: one for our CB model and one for the CF model. Each example in the dataset D for our CB model consists of a project-backer pair (p, b) with 12 dimensions presented below:

- *category count*: Number of projects backed by the user b in the same category as project p;
- *mode count*: Number of projects backed by the user b with the same funding mode (all-or-nothing, flexible or subscription) as project p;
- *same state*: True if the project geographic location is the same as the backer's;
- *recommended*: True if project p was manually recommended by an admin;
- *has video*: True if the project's creator has uploaded a video describing it;
- *budget*: Length of the textual description (prepared by the project's creator) of the project's budget;
- *description*: Length of the textual description (prepared by the project's creator) of the project's description;
- *pledged*: Amount pledged in the first 3 days since the launch of the project;

- *contributions*: Amount of contributions in the first 3 days
- *progress*: Percentage of goal reached in the first 3 days since the launch of the project;
- *owner projects*: Amount of projects from the same project owner as p;
- *reward count*: Number of offered rewards in project p;

In order to trim irrelevant data, we remove canceled and draft projects as well as projects with no backers from the dataset. The final cardinality of D was 300000 (p, b) pairs. Since D consisted of only positive samples (a user backing a project denotes a positive action), for the sake of balancing our dataset, we create 300000 more negative instances by randomly combining users with projects they have not backed. To further emphasize project quality, only successful projects are considered for the positive instances, while only failed projects are used for the negative instances. Finally, due to changes in the platform that occurred in 2015, we ignore data from earlier periods so as to maintain a consistent dataset.

The dataset used for the Collaborative Filtering model is a simple list of User ID and Project ID pairs, one for each contribution in the database. With this information, we build a $R_{n \times m}$ matrix where N is the number of users and M the number of projects. The rows indices represent the User ID, while the columns indexes represent the Project ID. A specific entry $R_{i,j}$ will be 1 if there exists a contribution made by the user with ID i to project with ID j, and 0 otherwise. At the time of writing, the cardinality of N was 900k and M was 75k, resulting in a matrix R with 67.5 billion entries. To achieve reasonable memory consumption, R is converted to a sparse representation.

4.2 Evaluation Metrics

There are basically 3 ways RSs can be evaluated: offline, online, and empirically. Offline measures are those computed based on a given static dataset, generally through techniques such as split validation and cross-validation. In these techniques, a subset of the data is withheld to be later used for testing; metrics are then computed for the test set only. According to Herlocker et al. (HERLOCKER et al., 2004), offline metrics can be broadly classified into the following categories: predictive accuracy metrics, such as Mean Absolute Error (MAE) and its variations; classification accuracy metrics, such as precision, recall and F1-measure; rank accuracy metrics, such as Pearson's product-moment

correlation, and normalized distance-based performance metric (NDPM). To define precision and recall, which will be used to measure performance in this work, we must first define the following terms: True Positives (**TP**): number of instances classified as belonging to its true class; True Negatives (**TN**): number of instances classified as not belonging to class A and that in fact do not belong to class A; False Positives (**FP**): number of instances classified as class A but that do not belong to class A; False Negatives (**FN**): instances not classified as belonging to class v but that in fact do belong to class A. We can then define precision as $P = TP/(TP + FP)$, recall as $R = TP/(TP + FN)$ and accuracy as $A = TP + TN/(TP + TN + FP + FN)$.

For ranked recommendations, we also define the Recall@N metric which evaluates whether an item that the user has rated in the past is in the top N recommendations from a random set of projects. It works as follows: a set of 100 random projects is augmented with a project that the user has interacted previously; we then measure if the interacted project is present in the top N recommendations.

In the experiments discussed in Chapter 5, we will present results for the most commonly used offline metrics for each implemented algorithm; however, it should be noted that they are not good indicators as to how well a RS will actually perform in production, and trying to optimize for them may actually degrade real-life performance (MCNEE; RIEDL; KONSTAN, 2006). To illustrate this point, let us consider a recommender system for movies. Suppose user Bob likes movies A, B and C. We split our dataset in such a way that movies A and B will be in the train set, while movie C is on the test set. Any kind of offline metric will test in a way or another if movie C was recommended to user Bob, resulting in a higher score if it is recommended and a lower score if other, unrelated movies, are recommended. The problem is that an algorithm that recommends movies D or E could actually be much better than one that recommends movie C, a fact that no offline metric could account for. This can be fundamentally attributed to the fact that, unlike most classification problems, recommender systems work with extremely sparse matrices; this sort of validation strategy would only be valid if we knew the ratings given by the user for every item beforehand. Evidently, in this imaginary scenario a RS would be pointless in the first place, consequently, other evaluation strategies are required.

Much more meaningful metrics can be obtained by online evaluation. While much more expensive and time-consuming to perform than offline measurements, live metrics are capable of testing directly whether the RS is achieving its intended purpose. The most common measures are the Click-Through Rate (CTR) and Conversion Rate (CR). CTR

is defined as the ratio of users who click on a specific link with respect to the number of users who viewed the page. In our case, if a set of N recommendations is displayed X times for distinct users, let Y be the number of times a user clicked on at least one of the recommended items in N. CTR is then defined as $\frac{Y}{X}$. CR is defined in the same way as CTR, except that users must not only click on a specific recommendation but also back the project. To compare two different RSs in an online setting, the most prominent approach today is A/B-testing. Since our experiment consists of 3 different approaches (in particular, CF, CB and Hybrid), an A/B/C test will be performed. For our online experiment, a recommendation algorithm was randomly selected for each user and click-through and conversion rates were measured by adding a reference header to projects that were accessed through the recommendations page.

Finally, empirical evaluation consists in having actually look at the results of an algorithm for one or more user profiles. This is equally important in any business setting. Consider a news portal that implemented a RS for its front page news. The RS might start recommending articles about cute cats, which might very well provide higher click-through rates than its more serious articles. This, however, is probably not what the company had in mind, and such a change in content might hurt the company's reputation in the long run. It is thus paramount to perform sanity checks on recommendation results manually. To achieve this in our experiment, a test page with each proposed algorithm was made available and a survey was sent to 14 members of the Catarse team. Participants were asked to analyze recommendations results for their own user on Catarse's platform for each proposed algorithm. The following questions were then asked of each algorithm:

1. : What is your level of satisfaction with this algorithm?
2. : How would you classify this algorithm with respect to the following properties (5 ratings, ranging from very unsatisfied to very satisfied)?
    - This algorithm showed me projects I would back;
    - This algorithm showed me projects I didn't know about;
    - This algorithm positively surprised me.

5 RESULTS

In this chapter, we discuss performance results for each of the three implemented algorithms, where (as discussed before) performance was measured according to the offline, online, and empirical metrics. The computed offline metrics were accuracy, recall, and precision, as defined in Section 4.2.

5.1 A Content-Based RS for Catarse

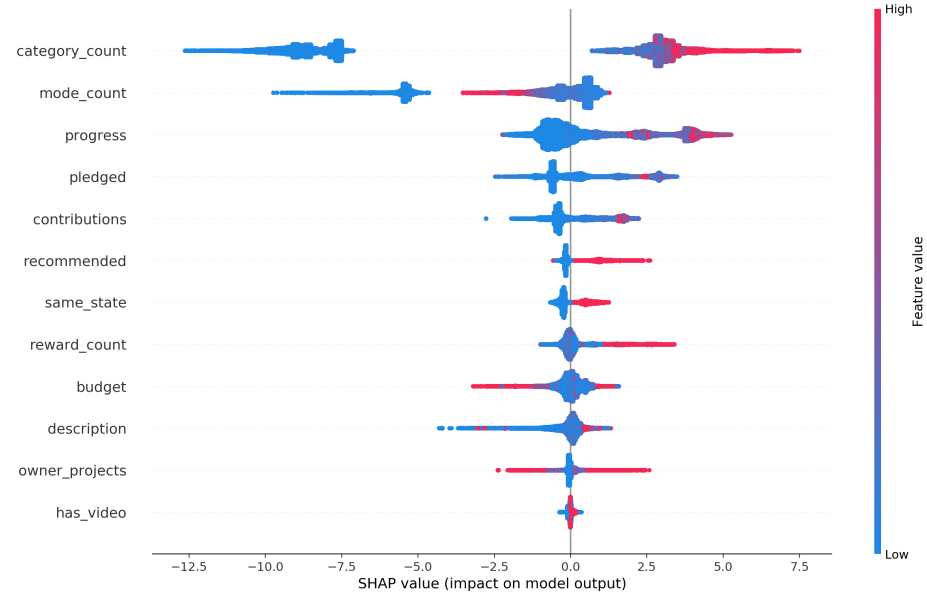
The Gradient Boosting tree method was chosen for our Content-Based system due to its state-of-the-art performance in many real-world applications (CHEN; GUESTRIN, 2016). The data described in Section 4.1 was used to train a model that predicts the backing probability of any user-project pair. The XGBoost Python library was used for this implementation. Hyperparameters were tuned by running 10-fold cross validation while optimizing for negative log-likelihood. No data standardization or normalization was necessary since the base learners are trees. In order to understand how each feature affects the outcome prediction in our model, we plot the SHAP[1] values of every feature for every sample in Figure 5.1. From this plot, we can draw some interesting conclusions: the most important feature in determining backing probability is the number of backed projects in the same category, followed closely by the project progress in the first 3 days. We can also see that having a low number of available rewards decreases backing probability.

To determine how individual features affect the output of the model, let's take our most significant feature, $category\_count$, and plot its value against the corresponding SHAP value for all examples in our dataset. Vertical dispersion shows the interaction between features, in this case how $category\_count$ relates to $reward\_count$. The result is shown in Figure 5.2. We can see that a higher $category\_count$ results in higher SHAP values, which in turn means that the backing probability is increased. This is easy to understand intuitively: the more projects in a certain category a user backs the more probable it is that they keep backing projects in the same category.

This approach gave exceptional results in every computed offline metric, as seen on Table 5.1. This, however, should be considered with care: our dataset is intrinsically

[1] SHAP is a unified approach to explain the output of any machine learning model. SHAP connects game theory with local explanations, uniting several previous methods and representing the only possible consistent and locally accurate additive feature attribution method based on expectations. SHAP values measure the impact of each feature on the model output. (LUNDBERG; LEE,)

Figure 5.1: Feature importance in Gradient Boosting Tree

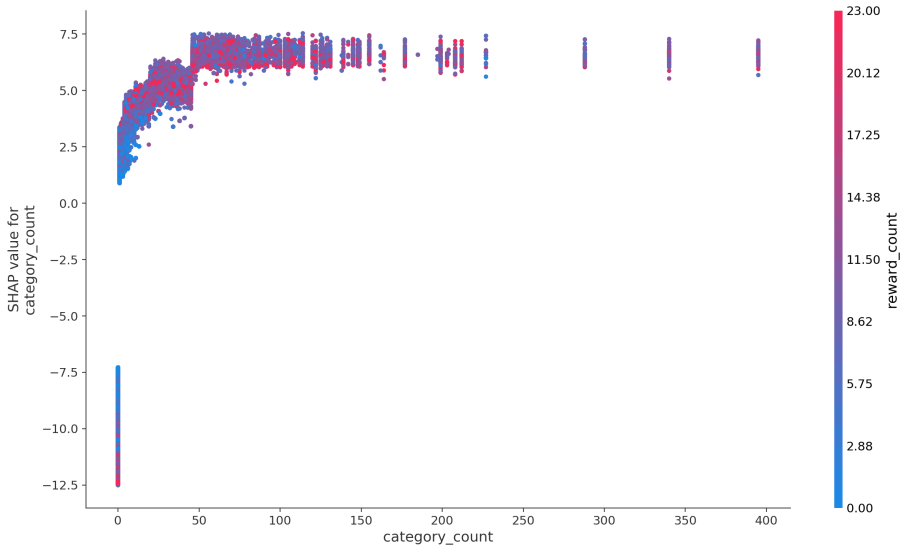


predictable to the point that any method should give great results; in fact, a simple test to check if the *category_count* feature is greater than zero already yields us an accuracy of 94.5%. This is due to the fact that most users only back one project in one category.

Table 5.1: Content-Based Offline metrics

Recall	*Accuracy*	*Precision*
0.990450	0.986260	0.982813

Figure 5.2: Dependence plot for category_count feature



5.2 A Collaborative Filtering RS for Catarse

Our Collaborative Filtering model was implemented using the LightFM library, which uses a Model-based matrix factorization algorithm. This algorithm was chosen for its simplicity and high performance in our extremely sparse rating matrix (approximately 1 rating for every 75k entries in the matrix). As depicted on Table 5.2, every offline metric for the CF approach, when applied to the Catarse system, performed much worse than its Content-Based counterpart. This was expected due to our dataset characteristics: since most users only back one project in their lifetime there is not enough rating data for CF to perform well. However, empirical and online tests (see Sections 5.6 and 5.5) using the learned CF model with real users provided much better results.

Calculating offline metrics poses a problem for CF algorithms that produce ranked recommendations. Given a user and a list of projects of size N, our model maps each project in the list to a real number and returns another list of size N with these numbers, which can later be sorted to show the most relevant recommendations in order. These numbers can assume arbitrary values which have no relation to the backing probability; they are used for ranking purposes only. Therefore it is impossible to compute an error metric since we do note have two sets of comparable values to test for equality. However, our hybrid approach requires that we have pairs of comparable numbers in order to compute the weighted average between CF and CB results. To make the CF model output comparable to our CB model output, which returns backing probability in the 0–1 range, we normalize the CF model output so it also stays in the 0–1 range. We then consider any normalized value greater than 0.5 to be in the positive class, and negative otherwise. These classes derived from the normalized values can then be used to compute offline metrics for our CF model. This gives us an accuracy of approximately 93%, or equivalently an error of 7%.

Table 5.2: Collaborative Filtering Offline metrics

Recall@5	*Recall@10*	*Precision@5*	*Precision@10*
0.06500372900657606	0.0925860086205232	0.015118373	0.010974493

5.3 A Hybrid RS for Catarse

Hybrid ratings were (as discussed before) computed by taking the weighted average of the ratings obtained by the CB method and the normalized CF ratings. By doing this, we hope to compensate any weaknesses in the original techniques and leverage its strengths. For example, a project highly rated by our CB algorithm but rated lowly by the CF algorithm could show that, although its features seem to indicate a good match for the user, similar users are not backing it in practice. By averaging both ratings such a project would rank much lower on the recommendation list, allowing better matching projects to appear in higher positions. Hybrid results are presented on Tables 5.3 and 5.4.

5.4 Discussion of Results

In this section, we will summarize and provide a comparative discussion of the performance of the different implemented algorithms according to each evaluation metric used: offline, online, and empiric.

5.4.1 System Performance According to Offline Metrics

Table 5.3 summarizes offline results for our three tested approaches. We can see that CB metrics provided much better results than the CF results, while the combined Hybrid approach resulted in slightly worse performance than the CB approach, however still presenting very good numbers. However, as noted on Section **??**, it is hard to compare offline metrics for systems that provide backing probability as is the case of our CB implementation with systems that simply provide an ordered list of recommendations, as is the case of the CF algorithm that we implemented. Therefore, these results should be taken with care and we should not assume that an algorithm with a better offline metric will result in better online metrics.

Table 5.3: Offline metrics of different RS approaches

Method	*Recall*	*Accuracy*	*Precision*
Content-Based	0.990450	0.986260	0.982813
Collaborative Filtering	0.089330	0.932910	0.010246
Hybrid	0.942722	0.964293	0.928434

5.4.2 System Performance According to Online Metrics

Online results were obtained by randomly assigning a recommendation algorithm for each user that visited the Explore Projects page. The existing popularity metric (i.e., the metric already implemented in the current Catarse system) was also added as a baseline for comparison, thus resulting in 4 test groups: CB, CF, Hybrid, and popular. For each group CTR and CR measurements were obtained. The experiment was run for 1 week on a live production environment with more than 20 thousand unique visitors per day.

It is known that the chance of obtaining a false positive increases with the number of variations tested. This probability is given by the formula $1 - (1 - a)^m$, with m being the total number of variations tested and a being the significance level, which gives us a false positive probability of 18% when $m = 4$ and $a = 0.05$. To correct for this, we compute a new required confidence level by applying the Bonferroni correction, which calculates the confidence level for a test with more than one variation. This correction simply divides our desired significance level by the number of variations, thus giving us a new required significance of $\frac{0.05}{4} = 0.0125$, or, equivalently, a required confidence level of 98.75%.

The performance results of the different algorithms according to the online metrics are described in Table 5.4. We can see from these results that Click-Through rates only improved for the CF algorithm in comparison to our popular control group baseline approach, while every other algorithm performed worse than the baseline. One possible cause for this is the intrinsic recommendation diversity provided by the CF approach; both CB and Hybrid recommendations are more likely to show projects similar to those already backed by the user; for example, a user that usually backs comic book projects would be shown mostly projects about comic books. However, such a user probably has already seen most available comic books projects and is, therefore, less likely to click on them again. On the other hand, diverse recommendations provided by the CF approach will present novel projects which are more likely to be clicked on. Nonetheless, simply improving CTR does not necessarily increase Conversion Rate, which is our real objective; in fact, our CR results show a quite different picture: not only did every recommendation approach performed significantly better than the baseline in this metric, but CF showed the least improvement while the Hybrid algorithm exhibited the best results, as expected. In fact, almost half of project views from the Hybrid ranking resulted in pledges to the

project, a particularly impressive result. The high amount of traffic that occurred during these experiments (approximately 20 thousand unique visitors per day) gives us the required confidence level thus validating the results.

Table 5.4: Online results of differente RS approaches

	CF	CB	Hybrid	Popular
CTR %	**42**	34	33	38
CR %	12	13	**15**	10

As discussed in Chapter 1.1, our two main goals from the business perspective were to increase the amount of money pledged to projects and to increase the percentage of successful projects. The first goal seems to have been achieved when looking at the conversion rates resulting from different RS systems: our hybrid approach resulted in 50% more pledges than the existing popularity ranking. Our second goal, however, cannot be measured directly on such a short experiment, and an indirect metric is therefore required for our comparison. In particular, it is reasonable to assume that more projects would be successful if pledges were distributed evenly among many projects instead of concentrating on just the most popular ones (many successful projects achieve significantly over a 100% of their goal mainly due to their exposure in the existing popularity ranking, creating a snowball effect; spreading this exposure to other high quality but unpopular projects should increase the number of successful projects); for this reason, we will measure the success of our second objective by looking at the project distribution of the pledges. The approach with the highest number of distinct projects pledged to will most likely meet our second goal. Table 5.5 show this analysis. We can see that every implemented algorithm increased this metric in comparison to our baseline, with the CF approach being the clear winner. Again, this can be explained by the diversity of results that is inherent to the CF approach.

From these results, it is clear that the CF and Hybrid approaches maximize different objectives, at least when evaluated via an online metric; in particular, CF approaches seem to maximize contribution spread, while Hybrid approaches seem to maximize CR. Balancing these possibly conflicting objectives corresponds to an optimization problem that is business-specific and beyond the scope of this work, but it is easy to see that this can be optimized by simply adjusting the weights in our Hybrid algorithm, giving more (or less) emphasis to the CF model.

Table 5.5: Number of distinct projects which received pledges

	CF	CB	Hybrid	Popular
Number of projects	**45**	29	26	24

5.4.3 System Performance According to Empirical Evaluation

The results on Table 5.6 were obtained by sending a survey to each member of Catarse's team. Respondents were asked to evaluate each available algorithm while logged in with their accounts in a production environment and answer several questions about each of them. To avoid bias, respondents were not given any details about the algorithms, which were labeled simply as 1, 2 and 3 to avoid identification. Five ratings were available for each question, from very unsatisfied to very satisfied. These rating were mapped to a 1-5 scale in order to obtain an average rating for each proposed question. It must be noted that, although this survey was aimed at people with vast knowledge and experience about crowdfunding projects, their recommendation results do not necessarily match those of a regular user. This is due to the fact that most site admins have backed hundreds of projects in order to test new features or while trying to reproduce bugs, which could greatly skew results since those projects are not necessarily the kind of projects they would normally back. To remove these outliers, only survey results of users with less than a 100 contributions were considered.

Table 5.6: Survey Results of different RS approaches

	CF	CB	Hybrid
What is your level of satisfaction with this algorithm?	3.8 ± 0.4	3.3 ± 0.5	$\mathbf{4.0} \pm 0.5$
This algorithm showed me projects I would back	$\mathbf{4.4} \pm 0.5$	3.6 ± 0.5	3.7 ± 1.3
This algorithm showed me projects I didn't know about	4.1 ± 1.0	3.5 ± 1.2	$\mathbf{4.7} \pm 0.2$
This algorithm positively surprised me	3.4 ± 1.1	3.5 ± 0.6	$\mathbf{4.0} \pm 1.7$

As expected, Hybrid recommendation results were the highest rated on average. Again as expected, the Collaborative Filtering approach gave more diversified results compared to the content-based approach, although, interestingly, Hybrid recommendations gave even more serendipitous recommendations.

One common complaint about the recommendation results was that "dead" projects, that is, projects which have not received any contributions for a long time, were being highly recommended in some cases. Indeed, since our dataset only includes data that is available in the first 3 days of the project's lifetime, it is possible that a project that received many contributions in its launch but flattened out later would be highly recommended. This is especially a problem for flexible projects, which can stay online for up

to a year. The Content-Based algorithm was also criticized for prioritizing highly popular projects over a list more tailored for the user. This is to be expected since the pledged amount was one of the most significant features in our ensemble model.

Collaborative Filtering results were much better than expected given the general characteristics of the dataset, but we must take into account the fact that most team members have backed on average much more projects than the average user, thus making them great targets for CF.

The last two questions, which were intended to measure novelty and serendipity respectively, showed the highest variation in the survey. This can be attributed to differences in familiarity with current projects amongst different team members. Members of the support team, for example, were more likely to be aware of the recommended projects than members of the financial team.

An additional question was added to the survey letting participants inform, in free form, general observations that they wished to make about each proposed algorithm. By reading these comments it was possible to notice that, even though users did not know which algorithm they were evaluating, their comments matched exactly the expected characteristics of each approach, further confirming the fact that these differences in characteristics are obvious even to non-technical users; that is, not necessarily programmers or personnel directly involved in designing the actual Catarse system, but people who might work, e.g., in the Human Resources office of the company. This ensures that the different subjective relevance metrics involved in this evaluation will be provided by users with a wider perspective than that provided by developers directly connected with the implementation details of the system.

6 CONCLUSION

After evaluating three selected state-of-the-art RS approaches (namely, CF, CB, and Hybrid) according to three different metrics, it is clear that personalized recommendations do provide better results than non-personalized in all measured metrics. This is to be expected since every user has distinct interests and a single list of recommended projects could not possibly interest every user. We can also see that offline metrics are indeed not good indicators as to how well a particular algorithm will behave in a productions environment in terms of CTR and CR; although CB showed almost perfect results according to our offline evaluation, it did not fare much better than other approaches in these online metrics. The CF approach presented surprisingly good online results considering the characteristics of our dataset, although it still could not beat other approaches when considering conversion rates. As expected from empirical results involving other commercial applications — such as Kickstarter's system, Hybrid recommendations performed best according our main objective of maximizing Conversion Rate. We can conclude that, based on our objectives (as described in Chapter 1), a Hybrid RS system would best serve our interests and should be the default recommender used in our production server. If the trends described in Chapter 5 continue to hold, we expect that by replacing the currently implemented *popular* recommendation strategy with a Hybrid RS, Catarse's Conversion Rate can improve by as much as 50% and the percentage of successful projects can increase as well.

The particular algorithms that we selected for representing each of the three types of evaluated RS approaches proved to be successful and a good match for our dataset. In particular, all of them performed better than the currently-deployed baseline algorithm according to the Conversion Rate metric. The different evaluated CF algorithms mainly differed in computational performance and further experimenting would not provide different recommendation results. As for CB approaches, we expect different sets of features and hyperparameters to produce quite different recommendations and they are therefore good targets for further experimenting, but these experiments are beyond the scope of this work. The transient nature of our projects turned out to be a quite manageable problem, even for the CF approach. The main difficulties observed in our implementation was related to choosing appropriate features for the CB model. We discovered that even small modifications in our training dataset could result in wildly different prediction performance; special care was taken, therefore, to make sure that every feature was meaningful

and independent in order for our model to work properly.

6.1 Future Work

A number of improvements can be made to the current implementation. First, for the Content-Based algorithm, new features can be added to improve its prediction power. More specifically, instead of only considering the length of the project's description we can analyze its content by using techniques such as Latent Dirichlet Allocation (LDA) or TF-IDF, and use the results from this analysis as features for our model. Secondly, time-aware features can be used to alleviate problems such as abandoned projects ranking highly, which was described in Section 5.6. Additionally, further rating information can be obtained by analyzing project visits and social media interactions to refine our models further. Finally, Hybrid recommendations can be further tuned by giving different weights to each algorithm, thus placing a higher emphasis in one of them.

REFERENCES

AGGARWAL, C. C. **Recommender Systems: The Textbook**. Springer, 2016. ISBN 978-3-319-29659-3. Available from Internet: <https://www.amazon.com/Recommender-Systems-Textbook-Charu-Aggarwal-ebook/dp/B01DK3GZDY?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=B01DK3GZDY>.

ANSARI, A.; ESSEGAIER, S.; KOHLI, R. Internet recommendation systems. **Journal of Marketing Research**, American Marketing Association (AMA), v. 37, n. 3, p. 363–375, aug 2000.

CHEN, T. **Introduction to Boosted Trees**. 2014. Available from Internet: <https://xgboost.readthedocs.io/en/latest/model.html>.

CHEN, T.; GUESTRIN, C. XGBoost. In: **Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD 16**. [S.l.]: ACM Press, 2016.

HERLOCKER, J. L. et al. Evaluating collaborative filtering recommender systems. **ACM Transactions on Information Systems**, Association for Computing Machinery (ACM), v. 22, n. 1, p. 5–53, jan 2004.

IAQUINTA, L. et al. Introducing serendipity in a content-based recommender system. In: **2008 Eighth International Conference on Hybrid Intelligent Systems**. [S.l.]: IEEE, 2008.

LI, P. Robust logitboost and adaptive base class (abc) logitboost. 2012.

LINDEN, G.; SMITH, B.; YORK, J. Amazon.com recommendations: item-to-item collaborative filtering. **IEEE Internet Computing**, Institute of Electrical and Electronics Engineers (IEEE), v. 7, n. 1, p. 76–80, jan 2003.

LUNDBERG, S.; LEE, S.-I. A unified approach to interpreting model predictions. 2017.

MCNEE, S. M.; RIEDL, J.; KONSTAN, J. A. Being accurate is not enough. In: **CHI 06 extended abstracts on Human factors in computing systems - CHI EA 06**. [S.l.]: ACM Press, 2006.

RAKESH, V.; LEE, W.-C.; REDDY, C. K. Probabilistic group recommendation model for crowdfunding domains. In: **Proceedings of the Ninth ACM International Conference on Web Search and Data Mining - WSDM 16**. [S.l.]: ACM Press, 2016.

RICCI, F. et al. **Recommender Systems Handbook**. Springer US, 2010. Available from Internet: <https://www.ebook.de/de/product/16201811/n_n_recommender_systems_handbook.html>.

SARWAR, B. et al. Item-based collaborative filtering recommendation algorithms. In: **Proceedings of the tenth international conference on World Wide Web - WWW 01**. [S.l.]: ACM Press, 2001.

SCHAFER, J. B.; KONSTAN, J. A.; RIEDL, J. E-commerce recommendation applications. In: **Applications of Data Mining to Electronic Commerce**. [S.l.]: Springer US, 2001. p. 115–153.

SU, X.; KHOSHGOFTAAR, T. M. A survey of collaborative filtering techniques. **Advances in Artificial Intelligence**, Hindawi Limited, v. 2009, p. 1–19, 2009.