

Exploring Collaborative Filtering Recommender System for Scratch

Roman Shaikh, B.Tech.

A Dissertation

Presented to the University of Dublin, Trinity College
in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science (Data Science)

Supervisor: Assistant Prof. Glenn Strong

August 2019

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Roman Shaikh

August 14, 2019

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Roman Shaikh

August 14, 2019

Acknowledgements

I would like to thank my supervisor Prof. Glenn Strong for his continuous guidance and the help he has given me throughout my thesis. I am also thankful to the School of Computer Science and Statics, Trinity College for providing a solid infrastructure and an excellent environment for working on my thesis. I would also like to extend my sincere gratitude to my second reader Prof. Tim Savage for his valuable input and suggestions on improving the thesis during the presentation.

Finally, I would like to thank my parents for supporting me and my each and every decision in life.

ROMAN SHAIKH

*University of Dublin, Trinity College
August 2019*

Exploring Collaborative Filtering Recommender System for Scratch

Roman Shaikh, Master of Science in Computer Science
University of Dublin, Trinity College, 2019

Supervisor: Assistant Prof. Glenn Strong

Scratch is an online learning platform developed by Lifelong Kindergarten Group at the MIT Media Labs. Scratch helps young and inexperienced students to develop programming skills and think creatively. With Scratch, one can program interactive stories, games, and animation. It also provides a collaborative platform through which users can share their own code and also see other peoples work. It has often been observed that some users' get demotivated easily because either they are unsure of where to go further after they have started or because the programming exercises are not up to their individual expectations. Thus, the concern arises of how do we keep users motivated on Scratch and improve the user experience. An effective solution is to recommend users with projects from other users according to their level of knowledge and previous experience. This intuitively is known as Recommender Systems (RSs), a system that recommends users with contents based upon their previous activities. Recommender system in an educational environment is proven to be significantly beneficial. We analyse the data made available by Scratch community and try to suggest an effective recommendation method. We explore the traditional recommendation techniques such as content-based filtering and collaborative filtering methods on dataset and compare the recommendation results. We explore the different methods for finding the recommendations from the dataset and the weighted average and multiple linear regression to evaluate the predictions.

Contents

Acknowledgements	iii
Abstract	iv
List of Tables	viii
List of Figures	ix
Chapter 1 Introduction	1
1.1 Motivation	3
1.2 Research Objective	5
1.3 Research Challenges	5
1.4 Dissertation Overview	5
1.5 Dissertation Structure	6
Chapter 2 State of the Art	7
2.1 Recommender system overview	7
2.1.1 What is a recommender system?	7
2.1.2 Information filtering techniques	7
2.1.3 Challenges in user-based collaborative filtering algorithms	9
2.2 Recommender systems in e-learning	10
2.2.1 E-learning environment and recommendation	10
2.2.2 Recommendations in Scratch	11
2.3 Evaluation of a recommendation system	11
2.4 Other techniques in recommendation systems	12

Chapter 3 Methodology	14
3.1 Summary of the Collaborative Filtering process	14
3.2 The item-based collaborative filtering algorithm	16
3.2.1 Similarity computation	16
3.2.2 Cosine based similarity	17
3.2.3 Correlation-based similarity	18
3.2.4 Adjusted-cosine based closeness	18
3.3 Prediction Calculation	19
3.3.1 Assigning the rating score for user activities	19
3.3.2 Weighted sum method	20
3.3.3 Regression model	20
Chapter 4 Implementation	22
4.1 Collecting the data	22
4.1.1 Data description	22
4.1.2 Ethical implication	23
4.1.3 Identification of the relevant dataset	23
4.2 Data Analysis	25
4.3 Calculating similarity matrix	29
4.3.1 Preparing the data	29
4.3.2 Imputing missing data	30
4.3.3 Correlation-based similarity	31
4.3.4 Cosine-based similarity	31
4.3.5 Adjusted-cosine based similarity	32
4.3.6 Selecting top-N recommendations	33
4.4 Prediction Calculation	34
4.4.1 Weighted Sum method	34
4.4.2 Comparing the recommended items	35
Chapter 5 Method of Evaluation	37
5.1 Experiment	37
5.2 Evaluation methods	37
5.2.1 Statical accuracy metrics	38

5.2.2	Decision support accuracy metric	38
5.3	Results	40
5.3.1	Comparing the similarity algorithms	40
Chapter 6	Discussion and Future work	42
6.1	Limitations	42
6.1.1	Over specialisation problem	42
6.1.2	Concentration bias	43
6.1.3	Cold Start problem	43
6.1.4	Dataset limitation	43
6.2	Future work	44
6.2.1	Qualitative testing	44
6.2.2	Mining text from the dataset	45
6.2.3	Social network analysis	46
6.2.4	Recommendation based on heuristics	46
6.2.5	A better Rating mechanism	47
Chapter 7	Conclusion	48
7.1	Summary	48
Bibliography		51
Appendices		56

List of Tables

4.1	Description of tables in dataset from Scratch online community [37]. . .	24
4.2	Mean, Variance and Standard deviation for Friends and Projects data .	29
4.3	Projects recommended	36

List of Figures

1.1	Projects by Scratch users	2
3.1	The process of a Collaborative Filtering Algorithm	15
3.2	Computation of item-item similarity	16
4.1	The distribution of projects in dataset over the period of 5 years	26
4.2	The frequency of the use of project blocks	27
4.3	Distribution of the user accounts created by creation date	28
4.4	Distribution of the friend's data over time	28
4.5	Pearson correlation output	31
4.6	Cosine similarity computation output	32
4.7	Adjusted cosine similarity output	33
4.8	Top 5 project ids according to prediction	34
5.1	Performance metrics calculations from Confusion metrics.	39
5.2	Performance comparison of the similarity calculation method	41

Chapter 1

Introduction

With the development of technology more and more focus is shifting on teaching kids the concepts of programming. Online learning is becoming an important part of our modern world. Development of Creative thinking is one of the fundamentals of early education. While kids usually get excited about learning new things, they lose interest pretty quickly. As difficult and challenging as the subject of programming can be, keeping the audience interested is even more so. Scratch by LLK group at MIT media labs is one such platform that aims at developing programming skills in kids while also keeping them interested. Scratch is not only aimed at developing programming skills but also enabling creative thinking and collaboration skills in children [1].

Scratch is a graphical programming environment that enables the user to create rich interactive projects including various media [2]. We can see various examples where users have developed a comprehensive range of programs like animations, storylines, games, music videos, online games, science projects, and sensor-driven projects (Figure 1.1).

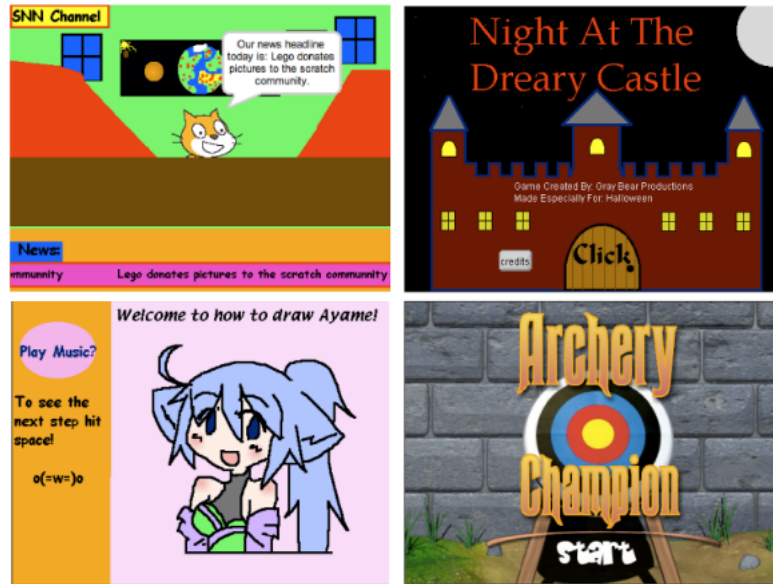


Figure 1.1: Projects by Scratch users

The Scratch platform lets users import various media elements like sounds and images into a project or even create internally using the inbuilt editor. These objects (called Sprites) can be made to interact with each other via programming. Programming is done using Lego-like blocks that snap together to form a logical command. These projects can then be saved on the computer or can be shared online on the scratch website.

A key objective of Scratch is to help self-coordinated learning through tinkering and joint effort with companions [2]. Commonly users learn Scratch as they go along exploring, trying and attempting commands from the command palette or investigating code from existing undertakings. To encourage such self-directed learning, the Scratch programming environment was designed to encourage scripting, give prompt feedback to content execution, and make information obvious [2]. When a project is shared online on the scratch website, it can be seen by other peers/users. Users can comment on the project to give feedback, give a like to the project or add the project to their favourite. Peers can also use the existing project and build upon the top of it. This is known as remixing. Scratch encourages users to explore the work by other users and collaborate to build on it. At the beginning some Scratchers (Scratch users) complained about the projects remix, claiming that others were stealing their work [3]. This led to a

discussion on the website forum about the idea of sharing in an open-source community. Scratch's goal is to make the users feel proud when their projects are remixed. Scratch consistently adds new features to the website to support this attitude. Whenever a project or task is remixed Scratch adds a link to the project where it originally came from, this gives the original author full credit. Also the website's landing page features top-remixed projects.

We have seen some quite interesting use cases of Scratch. One particular is of the users forming online companies. LLk observed that some of the online users have started their own "company" with the help of Scratch. A girl of 15 years of age from a town in England who goes by the alias BeeBop, set up a project with animations and sounds sprites and encourage users to use them in their own project or give an individual order on request for a custom build sprite. She was starting up the no-fee consulting company. Later, a girl with alias MusicalMoon, who was also from the UK, really liked the project by BeeBop and inquired if she would be interested in making a custom background for her own personal project. That is how Mesh Inc. started a self-declared mini-company which provided remarkable games in Scratch. After some time the company was joined by a boy from US of 14, who said he could help with debugging since he was good at Scratch programming [3].

Scratch is also being used in formal learning environments at universities to introduce high-school students and children to coding concepts. Studies have shown that visual programming languages have a notable influence on the understanding of computer programming concepts [4]. Some of the well-known visual programming languages like Alice [5], RAPTOR [6] and Etoys [7] have been utilised by various educational establishments. Scratch is one of the most popular and well-known visual programming language today [8],[2]. Although originally designed for kids, it was accepted for introductory programming lessons by a few higher education organisations [9].

1.1 Motivation

One of the main objectives of Scratch is to encourage collaboration amongst peers [2]. This is done by sharing the projects online with the community, getting feedback, tinkering with and mixing others work with their own ideas. If a user likes the project

scratch provides options to FAVOURITE, LOVE or REMIX it. Users can also leave comments on the project. Users can add other users (for example the author of the project they like) as friends i.e. follow them. Note that "Love" and "Like" are used interchangeably in context on Scratch

The development of Scratch is always tightly associated with the development of Scratch online website [10]. For Scratch to be successful, it must be connected to a community which supports collaboration and feedback from one another and improving on others work [11]. Hence Scratch is inherently built with the idea of "Sharing" with the community. Scratch gives a "Share" option on the project page. Once the project is shared on the website anyone can run it, tinker with it comment on it and vote using the Love button.

Following the launch of Scratch, within just 27 months users have shared more than 500,000 projects on the community website [3]. For many Scratch users, the concept of sharing their work with others and getting feedback and advice from peers is the source of strong encouragement. The huge amount of work that is shared on Scratch serves as a library of resources for exploring new ideas and learning new techniques in programming for other users.

Scratch website also serves as a platform for collaboration. Scratchers continuously build and adapt ideas from one another's projects. Around 15% of the total projects shared on scratch are remixed projects meaning they were built upon some other project that some other scratcher has previously shared [3]. For instance there are a number of different versions of Tetris, where different scratchers have added new functionality and features to the game to try and improve the experience. Similarly many doll based projects that exists on Scratch are adaptations from some original project [3].

There exist various different ways in which projects shared are discovered by other users. On the main home page are the sections like *Featured Projects*, *Featured Studios*, *Projects Curated by*, *Scratch Design Studios*, *What the Community is Remixing*, *What the Community is Loving*, this sections feature projects that are either hand-picked by curators, or are most trending projects in the community or the projects that are getting most attention(likes, favourites, remixes etc).

However, when it comes to personalised recommendation there is no option which looks at what is the user's interest or is based on users activities. Like in any online platform, self-directed learning plays an important part in Scratch. For enabling suit-

able self-directed learning it is important to guide the user through proper steps and resources[12]. By recommending projects based on users development and interest we can better enable user with a self-directed learning attitude.

Thus, this study focuses on developing an effective recommendation method for Scratch which would recommend users with the projects based on their preference and activities i.e. a personalised recommendation.

1.2 Research Objective

With the research goal in mind for Studying recommender system in the Scratch platform, this dissertation aims at achieving following objectives.

This dissertation focuses on the various content filtering techniques primarily collaborative and content-based methods. We will study and apply some of these collaborative techniques on the Scratch data. The dataset that is made publicly available by the LLK Scratch team will be used for analysis and model building. We will try to fit an item-item collaborative model on the project datasets and compare the recommendation outputs using standard metrics. We will also demonstrate how a recommendation would work on a scratch website by building a JS client/server model. Finally, we will discuss the risks and limitations of the proposed recommendations method.

1.3 Research Challenges

- Filter the huge dataset available for finding relevant and useful information.
- Find the best collaborative filtering algorithm that fits the available dataset.
- Finding the best metrics for evaluating the generated recommendations for quality and usefulness.

1.4 Dissertation Overview

Recommender systems are all around us, from the product recommendation you get in Amazon to Spotify generating a customised playlists. This dissertation discusses the

Collaborative filtering technique which is amongst one of the most common ways in recommender system research for calculating the predictions of items for a given user. There are many different techniques in collaborative filtering which can be applied depending on the requirements and the data. For predicting the Scratch projects a user might find interesting in Scratch, item-item collaborative filtering techniques are best suited. In item-item collaborative filtering, we first calculate the closeness of the project to other projects from our dataset and then check what projects from this list the user is most likely to like. This is done by looking at the past activities of the user.

The data is provided by the Scratch community website. It contains details of users, projects and the project contents that have been shared on the website. The data is collected over a period of 5 years from 2007 to 2012. The data is then filtered to search for the attributes that will be useful in our task for building a recommender system. We find that the most useful from the dataset is the users and the projects table, which contains information for our recommendation system analysis.

After fitting the item-based collaborative filtering model, we statistically evaluate the results and compare the output. For the purpose of demonstrating the recommendations system, a client/server model is built. JavaScript and Python have been used for the client-server interaction and Pandas library for data analysis and calculating the similarity matrix (See Appendix A for code reference).

1.5 Dissertation Structure

The rest of this document is orchestrated as following: Chapter 2 discusses the current state of the art in the field of recommender systems and explores the topic of recommendation system in education. It also looks at the previous studies done on Scratch with respect to recommendation methods. Chapter 3 describes the methodology for collaborative filtering techniques and different prediction and recommendation generation techniques. In Chapter 4 we apply the techniques discussed in the previous chapter on the Scratch data. Chapter 5 analyses different evaluation methods for a recommendation system and also examine the recommendation results obtained in the previous chapter. In Chapter 6 we discuss our recommendation approach, its limitations and future work that is required. Chapter 7 concludes the study.

Chapter 2

State of the Art

2.1 Recommender system overview

2.1.1 What is a recommender system?

Often termed Recommender systems, they are simply a kind of content filtering techniques and tools used to provide users with the most accurate and relevant product suggestions filtered from a huge database of information. Recommendation system works by discovering information patterns from the dataset, learning user choices and yield results that are co-related with their requirement and likings.

The most common use of recommender system can be seen in the commercial applications. From recommending movies on Netflix to predicting user ratings on amazon, the use of recommender systems can be seen in various fields. Fields like financial services [13] and research articles exploration [10] and to find collaborators [14] have seen increased used recently.

Recommender systems are extensively being used in e-learning environments under the context of TEL (Technology Enhanced Learning) [15] for improving the self-directed learning capabilities of students.

2.1.2 Information filtering techniques

Recommender system strategies generally use two types of techniques for generating recommendations one is content-based filtering approach and the other being collabora-

tive filtering approach. Some times knowledge-based systems are also used (also known as rule-based). A combination of the above three techniques is also used commonly known as the hybrid approach. In [16] Ricci categorises the recommender systems into 5 categories: Knowledge-based, Content-based, Demographics based, Collaborative filtering, community-based and Hybrid approach.

Content-based method sometimes, also known as item-based collaborative approach look at the individual items and their attributes to compute the similarity between the given item i and a list of items and then selects k most similar items [17].

Contrastively, Collaborative filtering techniques look at the users and their attributes. It calculates the closeness between all the users and then produces a recommendation based upon it. Basically, the idea of a CF-based algorithm is to look at the opinions of the like-minded users from the available pool and then recommend items based on it. The data of a users behaviour can be obtained in two ways explicitly or implicitly. Example of explicit data collection is the ratings and review a user gives to an item. Implicit data like the previous purchase history, browsing pattern of the user can also be useful in making a meaningful recommendation.

Knowledge-based techniques for recommendation takes into account the explicit knowledge about different things like the product, the user and some predefined rules to generate a recommendation. This is also known as rule-based recommendations [18]. The rule-based technique implements the relationship-rule discovery algorithm for finding the relationship amongst similar items and then produces a recommendation list based on the strength of the relationship [19].

For the purpose of this study, the focus is on the most widely used and perhaps the most effective technique [16] which is collaborative filtering. Collaborative filtering can be commonly categorised into two *1. User-based collaborative filtering* and *2. Item-based collaborative filtering*.

In user-based collaborative filtering, the behaviour of other users is incorporated in the system to give more weights to the items which users similar to the target user have purchased or expressed and opinion about. The intuition being the more similar a user is to our target user, the more likely it is that the target user will like the items that other users with similar interests have liked.

2.1.3 Challenges in user-based collaborative filtering algorithms

In spite of being amongst the most widely used algorithms in both small scale and large commercial-scale applications, the widespread use of user-based collaborative algorithms has shown some possible challenges with their use like:

- **Scalability:** Computational requirement for closest neighbour calculations increases with both the quantity of items and the users.
- **Sparsity:** In a large commercial use (like Amazon recommending books from large DB), even if a user has purchased less than 1% of the items (from millions of books) the nearest neighbour method won't be able to generate a recommendation of items for that user. Resulting in poor accuracy.

In conventional collaborative filtering recommender system, the quantity of work grows as the quantity of users increases. For producing accurate results quickly on a large dataset we will explore the item-based collaborative filtering technique.

The item-based method first looks at the user vs item matrix to find out the relationship between the various items present in the system and then utilize this information to implicitly calculate the recommendation for the user. In [17] the author discussed various techniques to compute the item-item similarity matrix (e.g. cosine closeness, Pearson correlation etc.) and also different methods (E.g., regression model vs weighted sum) to obtain recommendations from them. Because the association amongst the item is relatively steady, item-based algorithms may be able to produce equivalent or sometimes superior results [17].

In [17] the author studies collaborative filtering algorithms and their bottlenecks. Tapestry [20] shows the earliest applications of item-based collaborative recommender systems. The research in [20] relied on the direct inputs of users from the closed group, like a college working group. However, recommender systems for large groups like Scratch cannot rely upon every individual knowing each other one. In the course of time, many rating based automated recommenders were developed. The GroupLens research group at the University of Minnesota [21],[22] gives pseudonymous collaborative filtering solutions for the movies and Usenet. Video Recommender [23] and Ringo [24] are web-based music recommendation systems.

2.2 Recommender systems in e-learning

E-learning is a broad term that outlines an environment in which a person can learn at any time and anywhere using a computer, generally connected to a computer network. It is well established that e-learning can be as huge and valuable as the typical classroom experience or significantly more than it [25]. Using e-learning, one can learn and master the skills and knowledge just like he would in a traditional learning counterpart.

2.2.1 E-learning environment and recommendation

As e-learning systems begin to expand and are ever-expanding, the users need to first process a large amount of information before it meets their needs and provides them with items that are relevant to them. E-learning's fast development has altered traditional learning conduct and presented both the students and teachers with a fresh scenario. Students often find it difficult to navigate through a huge number of exercises and courses, and teachers find it difficult to recommend students with materials. One of the answers to this information overload issue in e-learning is a recommendation framework [26].

A personalized recommendation system in e-learning and online learning environments provides learning suggestions to students. In [27] Lu discusses a framework for a personalised learning recommender system. [26] discusses a fully mature recommendation system architecture model for an e-learning framework.

In [28] we see the application of Hierarchical Clustering (HC) on a TEL dataset built with Coursera data known as DAJEE. Recommendation from this system was done on the basis of three educational entities, Instructors, Courses and Lessons. The main teaching context considered were teaching preference of instructors, the course information and the lesson information.

In [29] the author discusses a hybrid approach for the recommendation on MOOC platforms which uses social network analysis and association rule mining techniques. User contributions on the forums and their social interaction and peer review activities were taken in to account to extract information for social network analysis. Followed by the utilising this information for the collaborative filtering approach. This approach was again tested on the Coursera dataset which showed a well-performing system given the limited information available in the dataset.

2.2.2 Recommendations in Scratch

In [18] the authors discuss recommender system in scratch with respect to a course framework for teaching Foundations of Computer Programming at Universidad Estatal de Milagro (State University of Milagro), Ecuador, with Scratch. consisting of various exercises that a student follows.

Benavides [20] proposes CARAMBA [30] a Scratch extension that recommends exercises, based on taste and program complexity. CARAMBA can recommend personalized exercises for students. This again considers problem statements as exercises that would be recommended. It also conducts a study to evaluate the improvements in student activity as a result of the recommended exercises.

2.3 Evaluation of a recommendation system

Recommendation algorithms typically perform differently on different recommendation tasks and domains. Therefore it is critical from a practical and research perspective to select the proper algorithm that matches the domain and tasks that is of the interest. The standard way to make such a choice is by comparing different algorithms offline using some assessment metric. Survey in [31] discusses different matrices for evaluating recommending algorithms.

In [32] the author discusses various evaluation methods for a recommender system, particularly on the top N recommendations tasks. It proposes that while assessing algorithms in the top-N suggestion task, to pick the test set cautiously otherwise precision measurements are strongly one-sided.

The area of recommendation system research has utilised different types of metrics to assess the quality of the recommendation framework. Mainly these are classified into two categories [17]:

- **Statistical accuracy metrics evaluation** computes the correctness of the recommendation framework by equating the value of the generated recommendation score with the user given rating in the test dataset. One of the most widely used methods for evaluating the statistical accuracy is called the Mean Absolute Error (MAE). MAE is calculated between the predictions and the user-ratings. MAE gives the deviation of the recommendation score from the user given score.

- **Decision support accuracy metrics evaluation** evaluates how effective and useful the predictions are at helping the user find the relevant item from the set of all items. These metrics consider the process of recommendation as a set of binary task, wither the recommendation is good or bad, i.e. the item is either predicted or not considered at all. With this in mind, the prediction can be anything like 1 or 2.5, it doesn't matter as the user only selects items with a rating higher than 4. Most common techniques in this category are the ROC sensitivity and the reversal rate [33].

These techniques are covered in detail under chapter 5 Methods of Evaluation.

2.4 Other techniques in recommendation systems

Additionally, other techniques have been applied to the recommender system area, like Clustering, Horting and Bayesian networks.

Bayesian networks work by creating models over a training dataset associated with a decision tree-like structure. Each edge and node of the tree represents knowledge about the user. The model can be trained and completed in the time as less as hours or a few days. This model tends to be very small and very fast and can be as accurate as the nearest neighbour methods [34]. Bayesian can be used practically in environments in which the information about the user preferences changes steadily in reference to the time which the model building requires. But Bayesian networks have been known to fail in areas where user preferences change very rapidly.

On the other hand, clustering algorithms work by classifying users with similar preference into different groups known as clusters. After the clusters have been finalised, predictions for a user are calculated by taking the average of the opinions of other users in the cluster. Some of the clustering techniques consider users in the form of participation across several clusters and the averaging is done by the weighted sum of the degree of the participation. Clustering is known to produce non-personal results in the recommendation, in fact, in some cases, it may give even worse accuracy than the nearest neighbour algorithms [34]. However, once the clustering is done, there is a large increase in performance because the comparing needs to be done on very small clusters. Clustering techniques can also be used in conjunction with nearest neighbour

algorithms to reduce the size of the dataset that must be analysed.

Horting is a diagram-based methodology in which the nodes of the graph represents the users and the connections between them are edges representing the degree of closeness between the users [35]. Recommendations are given by traversing the graph to the nearest node and averaging the opinions of nearby nodes (i.e. users). Horting and nearest neighbour differ from each other in the sense that in Horting graph may be traversed through users who have not had an opinion about the item and therefore considering the changing relationships that nearest neighbour does not consider [2]. In a study with artificial datasets, Horting has shown superior performance to the nearest neighbour algorithms [2].

Chapter 3

Methodology

In the previous chapters, we investigated what are recommendations system and the different techniques applied in the recommender system and also their application in the e-learning domain. This chapter will describe the process in which an item-based collaborative filtering algorithm was applied to the Scratch data to build a recommendation system. We will look at different steps involved in the collaborative filtering methods, various item-similarity computation methods and also the prediction calculation steps.

3.1 Summary of the Collaborative Filtering process

The objective of a Collaborative Filtering (CF) algorithm is to generate a list of recommended items or to estimate the likeliness of the item for a particular user based on the past activities of the user. In an standard CF scheme there is a list of m number of users say $U = \{u_1, u_2, u_3, u_4, \dots, u_m\}$ and a list of n items $I = \{i_1, i_2, i_3, i_4, \dots, i_n\}$. Every user u_i has a list associated with them consisting of items I_{U_i} , which the user has expressed interest in or has an impression about. In the case of Scratch we look at the projects the user has liked and the projects user has created/published in the past. This impressions or likings can either be collected explicitly or implicitly. Notice that the item list for user $I_{U_i} \subseteq I$ and its likely that the I_{U_i} might be an empty set (e.g. when the user newly enters the system). Also, there exists a user u_a from the list of users for whom the task is been carried out i.e. our target user. The goal for CF

algorithm is to discover an object (in this case projects) likeliness i.e. how likely the user is to view or love this item. This can take two forms.

- **Recommendation** R is a list of items such that $I_r \subset U$ which contains the items the user will like the most. It is to be noted that the R should avoid items that the user is previously associated with i.e., for example, the projects that the user already has on his page.
- **Prediction** is a number $P_{a,j}$ representing the predicted likeliness of the item for the user u_a . The predicted value is same as that of the normalized scale of the user ratings [17]. For the case of this study in Scratch we consider this to be the count for remixes, likes and favourites for the user-item pair.

Figure 3.1 shows a diagrammatic representation of the collaborative filtering process. The CF algorithm shows the whole $m \times n$ matrix A as a user vs item matrix of the user ratings. Each cell $a_{i,j}$ in the matrix holds value for every item the user has rated i.e. either liked, favoured or remixed. Every rating is on the numerical scale and can take the value 0 representing the user has not yet given the rating (i.e. liked or favoured or remixed the project) the project or a value 3 (done all three activity) indicating user liked the project very much.

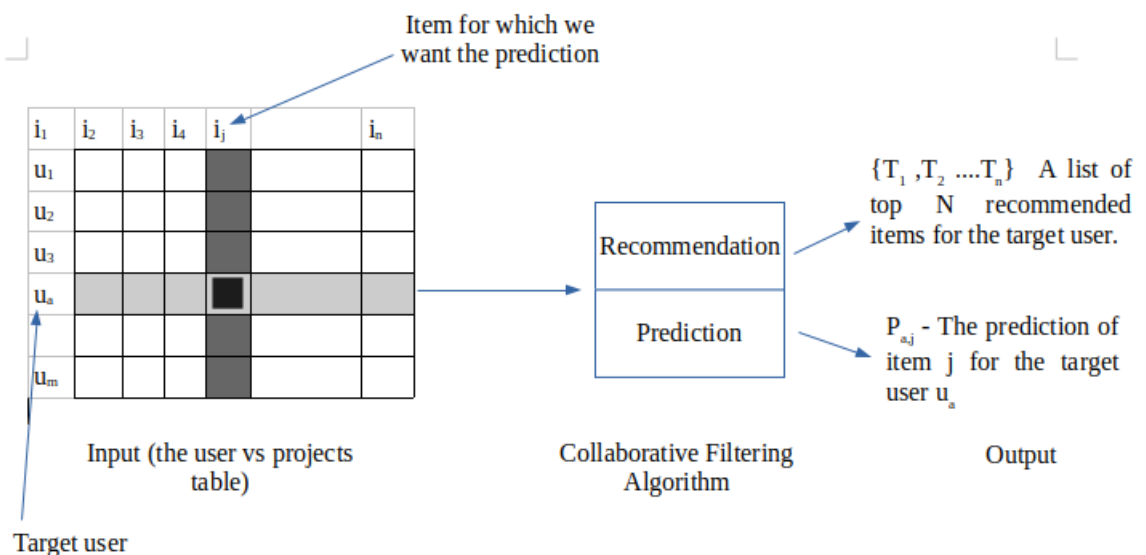


Figure 3.1: The process of a Collaborative Filtering Algorithm

3.2 The item-based collaborative filtering algorithm

Unlike the user-based collaborative filtering algorithm shown above, an item-based collaborative filtering algorithm considers the semantics of the items a user has interacted with (liked, remixed or added to the favoured) to compute the likeness between two things i.e. how closely related items $\{i_1, i_2, i_3, i_4, \dots, i_n\}$ and the item of interest are. Then it selects the k most closely related items $\{i_1, i_2, i_3, i_4, \dots, i_k\}$. Once the task of computing similarities is complete, the predictions are calculated by taking the weighted sum average of the target users ratings on these similar items. This process of similarity calculation and prediction generation is detailed below.

3.2.1 Similarity computation

One of the most critical steps in an item-based collaborative filtering model is the calculation of the similarity matrix between the set of items and then selecting the most similar items. This similarity calculation step between two items i and j , basically performs two steps first it separates the users who have evaluated both the items and then applies a closeness calculation strategy to decide the similarity $s_{i,j}$. Figure 3.2 represents this procedure. Here the matrix rows denote the user and the columns denote the items.

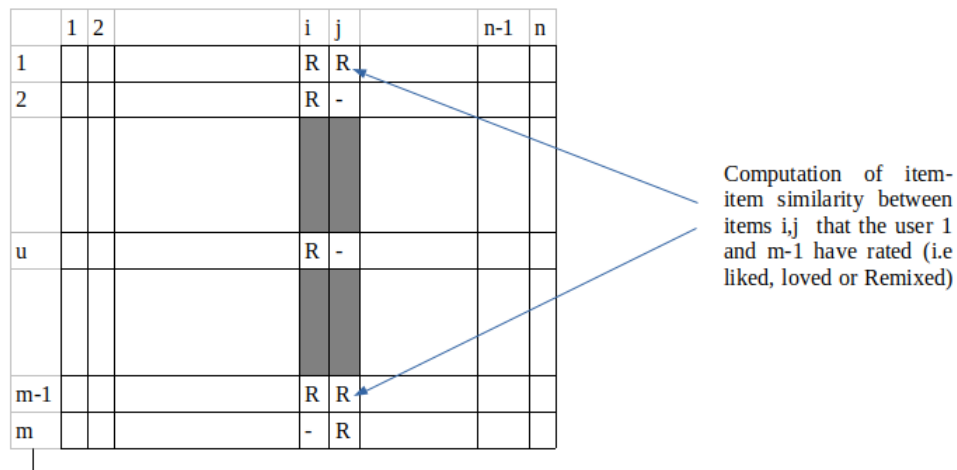


Figure 3.2: Computation of item-item similarity

Various techniques have been proposed to compute the similarity between any two items in statics. Some of the most common and well known once are *Correlation-based similarity*, *Cosine-based closeness* and *Adjusted-cosine based closeness* [19]. We will examine each of this technique and apply them on our dataset to check which of them suits best for the chosen dataset.

3.2.2 Cosine based similarity

Cosine similarity is defined as the measure of similarity between any two non-zero vectors. These vectors reside in an inner product space. Cosine similarity is defined as the cosine of the angle between them. The cosine for a vector with the angle at 0 degrees is 1 and the for any angle between 0 and 90 is less than 1. Therefore it gives us information about the orientation of the vectors and not their magnitudes. Two vectors with orientation in the same direction have a cosine 1 and the cosine for two vectors that are at an angle of 90 degrees with respect to each other is 0. Similarly, two vectors within an exact opposite orientation will have a cosine similarity as -1 irrespective of their magnitude. Cosine similarity mostly is used in positive spaces where we require the output to be neatly bounded between [0,1]. Particularly data mining, information retrieval and text mining all of which have high dimensionality problem extensively use this technique. Each of the terms is assigned a new dimension to calculate the cosine similarity between them. The cosine similarity is useful because even if two vectors are very far apart in terms of Euclidean distance (due to their sizes), there is a chance that they may be similarly oriented closer to each other. The smaller the angle between the two, the higher is their cosine similarity. For this study, we consider two items as the two vectors in the m dimensional user-space. The similarity measure is calculated by computing the cosine of the angle between these two vectors. In Figure 3.2 we see the similarity between items i and j in an m x n matrix for user projects. The similarity is shown by $sim(i, j)$ such that

$$sim(i, j) = \cos(\vec{i}, \vec{j}) = \frac{\vec{i} \cdot \vec{j}}{\|\vec{i}\|_2 * \|\vec{j}\|_2}$$

In the equation above, dot product is represented by ”.”

3.2.3 Correlation-based similarity

In statistics, correlation is a statistical association between two variables, showing the degree to which the two are linearly related to one another. There are many correlation coefficients defined which show the degree of closeness between variables. Pearson correlation, Interclass correlation, Spearman's rank correlation are some of the examples of correlation coefficients used in statistics. Pearson correlation is one of the most widely known and used metrics to show the similarity.

Also, known as the person-r correlation coefficient, it gives us the linear correlation between any two variables X and Y . It can take a value between $+1$ and -1 , where 1 denotes a total positive correlation between the variables, 0 denotes absolutely no correlation between the variables and -1 denotes a total negative correlation [36]. Mathematically it is defined as the covariance of the two vectors divided by the product of their standard deviation. In a correlation-based similarity scenario, the likeness amongst two items i and j is computed by calculating the Pearson-r correlation $corr(i, j)$. For a correlation to be accurate the cases where the users have rated the projects same will need to be separated.

$$sim(i, j) = \frac{\sum_{u \in U} (R_{u,i} - \bar{R}_i) (R_{u,j} - \bar{R}_j)}{\sqrt{\sum_{u \in U} (R_{u,i} - \bar{R}_i)^2} \sqrt{\sum_{u \in U} (R_{u,j} - \bar{R}_j)^2}}$$

3.2.4 Adjusted-cosine based closeness

One basic contrast between the closeness calculation in a user-based collaborative filtering and item-based collaborative filtering is that in the item-based method the similarity is calculated based on columns of the matrix, whereas in user-based method the similarity is calculated along the rows of the matrix i.e., each pair in the co-appraised set relates to a different user (Figure 3.2). Computing similarity utilising the fundamental cosine measure in item-based case has one significant downside - the distinctions in the rating scale between various users are not taken into consideration. The adjusted-cosine based similarity reduces this disadvantage by subtracting the average score from every co-evaluated pair. The formula for this is given as below

$$sim(i, j) = \frac{\sum_{u \in U} (R_{u,i} - \bar{R}_u) (R_{u,j} - \bar{R}_u)}{\sqrt{\sum_{u \in U} (R_{u,i} - \bar{R}_u)^2} \sqrt{\sum_{u \in U} (R_{u,j} - \bar{R}_u)^2}}$$

Where R_u is the normalized score for u th user.

3.3 Prediction Calculation

One of the most significant steps in any CF-based system is to give the output in terms of the prediction. Once we have with us the list of items that are most similar as given by the closeness measures above, the next step is to examine the user ratings and apply a predictive method to get the prediction. Various ways have been proposed to compute the prediction. Regression method and Weighted Sum method are the two such most common technique [19].

3.3.1 Assigning the rating score for user activities

In a conventional personalised recommendation, users feedback is captured through the likes and the rating given to an item. For example in case of Netflix a user rates a movie on a scale of 1 to 5, 1 being the lowest and 5 being the highest. Some times decimal ratings are also allowed. Since there is no rating system in scratch and a user can either like, favourite or remix a project and these three details are publicly available we will use this information to develop a rating system. While viewing a project, a user has a choice of performing various actions on the project page. He/She can either click on Love/Like to show the appreciation or add the project to his favourite or click on remix to copy the project to his own profile and adapt on its code. User can perform one or two or all three of these activities. For each of the user activity (love, favourites and remix) we assign a rating score of 1 if the user has performed the activity or 0 if not. We then take the sum of all three activities to find a user rating on a scale of 3. So for example, if a user has liked and favourited a project but not remixed the total user rating is *love (1) + favourite (1) + remixed (0) = 2*. And If a user has only remixed the project then the total rating for that project is *love (0) + favourite (0) + remixed (1) = 1*. Henceforth, in our weighted sum method we use this rating to find the most relevant projects for the user.

3.3.2 Weighted sum method

In mathematics, a weighted function is a method used when performing an average, a sum or an integral to assign some components more weight i.e. importance than others to increase or decrease their impact on the result. Weighted sums are most commonly used in statistic to decrease or compensate for the bias present in the data. For example, if a quantity F is measured multiple times independently with variance σ^2 , we can find the best approximation of this measure by taking the average of all the measurements with the weight $\omega_i = \frac{1}{\sigma^2}$, giving us a smaller variance than each individual measurement.

This method computes prediction for the project i for user u , by calculating the weighted sum of rating attributes (likes, favourites, remixes etc.) for a similar item. Formally this can be shown as the prediction $P_{u,i}$ as below

$$P_{u,i} = \frac{\sum_{all\ similar\ items,N} (S_{i,N} * R_{u,N})}{\sum_{all\ similar\ items,N} (| S_{i,N} |)}$$

Basically, this method attempts to show how a target user u , is likely to view (favourite it, like it or remix it) based on the similar items that he has seen (liked, loved, or remixed) in past.

3.3.3 Regression model

In statistics, regression modelling is a collection of procedures to estimate the dependency relationships between the variables. Regression analysis aids us in understanding the effect that the dependent variable has when anyone or multiple independent variables are changed. A number of regression techniques have been invented to support various statistical problems. Linear regression, polynomial regression, principal component analysis etc. are some of the popular techniques.

Regression model for calculating the prediction is mostly similar to the weighted sum method discussed above. The only difference is that instead of using ratings of similar items, this method approximates the ratings of items based on the regression models. When similarity is calculated using either cosine or correlation measurement, it may lead to improper conclusions in practice because the two vectors maybe at distance in terms of euclidean distance, but may be highly similar to each other in

terms of orientation. In this case, using the ratings from similar items can result in low-quality predictions. The general idea is the same as that of the weighted sum method, but instead of using raw ratings from the N similar items, this method uses their approximate values which are calculated using a linear regression model.

General formulation of a CF-algorithms and the necessary methods and steps required for performing item-based collaborative filtering have been discussed until now. In the next chapter, all these algorithms will be applied to the actual Scratch dataset.

Chapter 4

Implementation

For implementing the item-based collaborative filtering methods on Scratch, we first look at the data that is required and the available datasets from Scratch.

4.1 Collecting the data

The Scratch Online Community has made the dataset for scratch platform publicly available on [37]. The dataset has the data for the first five years of the data from Scratch (approximately from 2007-2012). The data was collected from the MySQL database from the Scratch website. All datasets are provided in CSV file formats, which can be loaded into any compatible data analysis software. There are a total of 32 datasets which contain information about various aspects of the Scratch Online Community. These 32 datasets are divided into 3 main categories *Core Datasets*, *Text and Code Datasets* and *Project Analytics Datasets*. Table 4.1 shows the description of each and every table that is available in the dataset taken from [37].

4.1.1 Data description

- **Core dataset** consists of metadata and data tables that define major relationships and objects which are captured by the Scratch Online Community application.
- **Text and Code datasets** consist of very large tables consisting of texts sub-

mitted by the users. This can be helpful for NLP and text-based analysis.

- **Project Analytics datasets** contain tables which have quantitative summaries of each project file. A project file is a .sb2 or .sb3 file that contains all of the content of a scratch project.

4.1.2 Ethical implication

The dataset that this research uses is the data from the Scratch Online Community. It has been supplied by the Lifelong Kindergarten Group at the MIT Media Lab under public domain. Although this data is available under public domain, a Scratch Research Data Sharing Agreement (SRDSA) [37] was signed to obtain access to the dataset files. User privacy is fully protected while scraping the data. The rules that define the user privacy such as only information that was publicly visible on the site to anyone visiting the site even if it may be a web crawler software and no action was required by the user or the site administrator were used while scraping the dataset. Some of the user sensitive information like the self-reported gender, age, email, IP address was omitted while publishing the dataset [38].

The users of the Scratch online platform are mostly young children many below 18 years of age, from all around the globe. Therefore, an ethics approval was required by the LLK group from the MIT Committee on the Use of Humans as Experimental Subjects (COUHES). Which was granted to the team and the data was published under the specified protocol by (COUHES) MIT.

All the terms of the access to the data have been adhered to and therefore as the part of this dissertation, no ethics approval was required from the university.

4.1.3 Identification of the relevant dataset

Keeping the objective of the research in mind of building a recommender system that generates recommendation on "projects" for a user, first we need to identify the relevant data for our goal from this dataset.

For recommending projects based on user activity, we require the data related to projects and users on the Scratch platform. Looking at the list of the datasets as shown

Core Datasets	
1. users	Each row in this table represents a user account that was publicly visible on the Scratch website at the time that these data were collected.
2. projects	Each row in this table represents a project that was publicly visible on the Scratch website at the time that these data were collected.
3. galleries	Each row in this table represents a gallery that was publicly visible on the Scratch website at the time that these data were collected.
4. friends	Each row in this table represents the event of auser"friending" another user.
5. downloaders	Each row in this table a represents the event of auserdownloading aproject.
6. favoriters	Each row in this table represents the event of auseradding aprojectto their favorites, i.e., bookmarking it.
7. lovers	Each row in this table represents the event of auserclicking a heart-shaped "love-it" button that appears on everyproject'spage.
8. viewers	Each row in this table represents the event of auserviewing or loading the webpage of aprojectfor the first time.
9. pcomments	Each row in this table represents a comment left on aproject.
10. gcomments	Each row in this table represents a comment left on agallery.
11. projects_galleries	Each row in this table reflects aproject'sinclusion in agalleryat the point of data collection.
12. tags_projects	Each row in this table represents a tag placed on aproject.
13. tags_galleries	Each row in this table represents a tag placed on agallery.
14. frontpage_projects	Each row in this table represents aprojectthat has been displayed on the front page for any of several reasons.
15. featured_projects	Each row in this table represents aprojectthat has been displayed on the front page of the Scratch website in a section called "featured projects" that held three projects at a time.
16. featured_galleries	Each row in this table represents agallerythat has been displayed on the front page of the Scratch website in a section called "featured galleries."
17. studio_galleries	Each row in this table represents agallerythat has been displayed on the front page in a section called "Design Studio."
18. curators	Each row in this table represents auserwho at some point was in charge ofprojectselection for a section of the Scratch website's front page labeled "Curated By."
Text and Code Datasets	
1. projects_text	Each row in this table represents aprojectthat was publicly visible on the Scratch website at the time that these data were collected.
2. galleries_text	Each row in this table represents agallery.This is the text table that contains the free-form and the unstructured text fields in the galleries table.
3. pcomments_text	Each row in this table represents a comment left on a project. This is the text table that contains the free-form and the unstructured text fields in the pcomments table.
4. gcomments_text	Each row in this table represents a comment left on a gallery. This is the text table that contains the free-form and the unstructured text fields in the gcomments table.
5. tags_text	Each row in this table represents a tag used on a project and/or gallery. This is the text table that contains the free-form and the unstructured text field in the tables tags_projects and tags_galleries.
6. project_block_stacks	Each row in this table represents the textual representation of a code block associated a sprite used in the most recent version of a project shared on the Scratch website at the point of data collection.
7. project_block_stacks_disconnected	Each row in this table represents Scratch blocks that are never executed i.e., those blocks that do not have an execution trigger or "hat block" on top.
8. project_strings	Each row in this table represents a text string free-form text strings typed by users as part of their code used in the most recent version of aprojectshared on the Scratch website at the point of data collection.
Project Analytics Datasets	
1. project_blocks	Each row in this table represents a project shared on the Scratch website at the point of data collection. Each column of this row represents the frequency counts of each block type for a particular project.
2. project_drums	Each row in this table represents a drum used in the most recent version of a project shared on the Scratch website at the point of data collection.
3. project_media	Each row in this table represents a media item (e.g., an image or a sound) attached to a sprite in the most recent version of a project shared on the Scratch website at the point of data collection.
4. project_midi_instruments	Each row in this table represents a musical instrument used in the most recent version of a project shared on the Scratch website at the point of data collection.
5. project_save_history	Each row in this table represents a time when a user saved a project to their local storage device (e.g., hard drive) in the most recent version of a project shared publicly on the Scratch website at the point of data collection.
6. project_sprites	Each row in this table represents a sprite used in the most recent version of a project shared on the Scratch website at the point of data collection.

Table 4.1: Description of tables in dataset from Scratch online community [37].

in Table 4.1 we observe that `projects.csv`, `projects_blocks.csv`, `projects_sprite.csv` and `users.csv` have most of the data related to users and projects.

- **Users** - holds the data about the user and which project the user has worked on.
- **Projects** - contains general information about the projects and the user associated with each public project.
- **Projects_block** - holds the information about each block (the logical component of the code) used in a project, these are the extracts of the `sb2` and `sb3` files.
- **Projects_sprite** - contains the information about sprite (scripts, sounds and images) used in each project.

Apart from these tables, the following datasets were also used to obtain information required for user rating calculation.

- **Lovers** - which has information about the projects that the user has clicked Love for.
- **Friends** - contains each user and its friends (the user who follow each other).
- **Favourites** - shows an event where a user has added a project to their Favourites.

4.2 Data Analysis

After identification of the data required, we will apply some preliminary data analysis techniques to check for consistency and quality of the data.

Projects - We see that the more of the projects are created in the latter half of the dataset time frame i.e. after 2010. This shows the popularity of the Scratch in the initial phases.

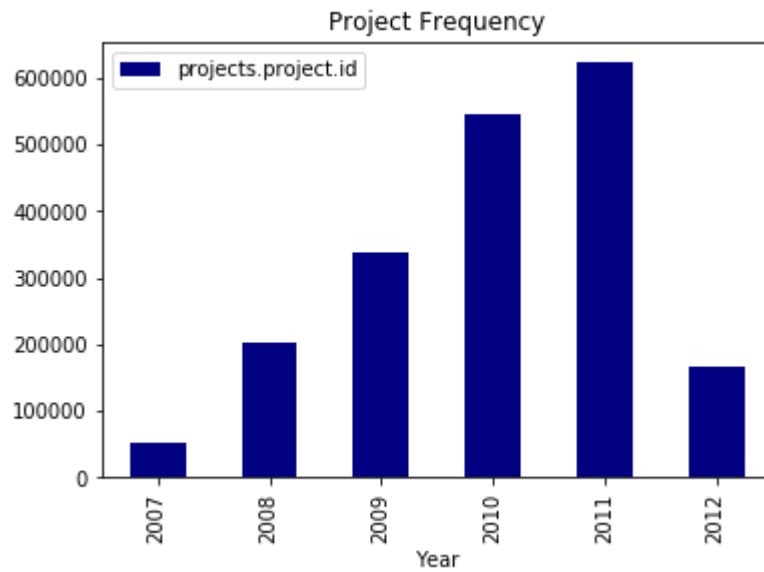


Figure 4.1: The distribution of projects in dataset over the period of 5 years

Project Blocks - There are about 171 of these key block attributes associated with each project in this dataset. Table 2 shows each of this block. Figure 4.2 below shows the frequency of each block used in the projects. We see an even distribution of the most common operators in the projects, while the less common ones are not used frequently. We can assume from this distribution that project dataset is good enough for finding the correlations between projects.

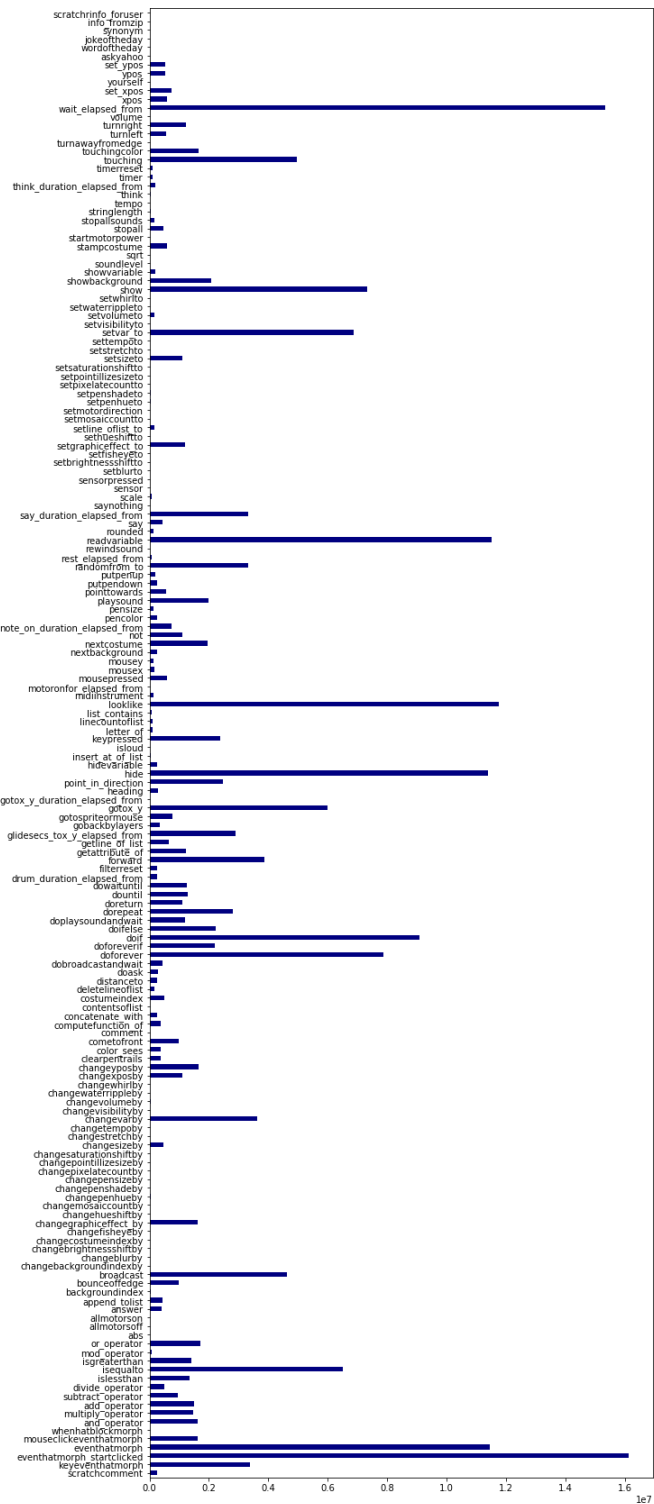


Figure 4.2: The frequency of the use of project blocks

Users -The following figure 4.3 shows the distribution for the accounts in scratch done by creation date over the period of 5 years.

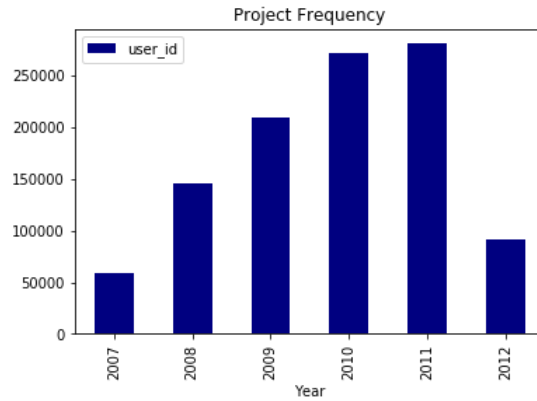


Figure 4.3: Distribution of the user accounts created by creation date

Friends - An event where a user adds another user as a friend on the Scratch community website is recorded in this dataset. Figure 4.4 shows the distribution of this relationship over time.

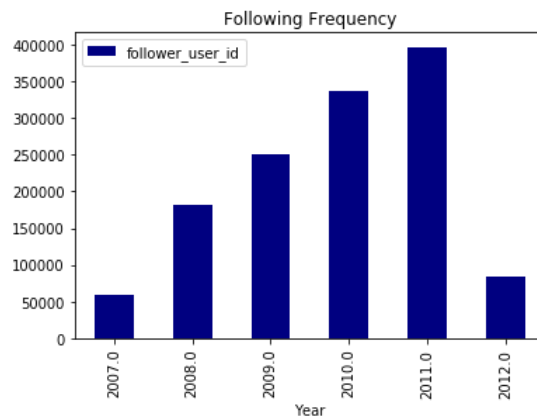


Figure 4.4: Distribution of the friend's data over time

From table 4.1, we observe that that on an average 6.32 projects have been created per user with a SD of around 26. And on an average, a user has around 11 Friends per user with a SD of 55. The variance may seem huge but considering the size of the data which contains around 1.3 million records for friends and 1.9 million records

for projects in the dataset, the variance is expected. The high variance values in both datasets indicate number of missing values from the data, this will be handled in the data cleaning and imputation section 4.3.2.

Observations	Mean	Var	SD	Max	Min	NAs
Projects	6.32	721.9964	26.86999	3047	1	0
Friends	11.31493	3073.802	55.44188	4981	1	0

Table 4.2: Mean, Variance and Standard deviation for Friends and Projects data

4.3 Calculating similarity matrix

After the data analysis, we will apply the three similarity computation matrix discussed in chapter 3 on our projects dataset. All the programming and computation is done in python as the programming language. Various packages and libraries have been used to facilitate the easy application of different algorithms. Python's sci-kit learn (sklearn)[39] library, which houses many machine learning packages has been used for applying the cosine- correlation and adjusted cosine correlation methods. Another library known as Pandas [40] has been used for computing the Pearson correlation matrix.

4.3.1 Preparing the data

Since our `project_sprite` and `project_blocks`, both contain some information about the project components we can merge the two datasets to form single dataset. This has two benefits first it reduces the process of applying computation on two different datasets. Secondly, it gives us 3 more parameters(sounds, images and scripts) from `project_sprite` which will add to the existing 171 parameters from `project_block` dataset to calculate the correlations.

```

1 #Load projects data from csv
2 projects_df = pd.read_csv('data/CSVs/project_blocks.csv', sep=',',
   index_col=0,)
3
4 #Imputing the others data since it has mostly the null values.
5 projects_df = projects_df.drop(columns="other")

```

```

6 #Load projects_sprites data from csv
7 project_sprites_df = pd.read_csv('data/CSVs/project_sprites.csv', sep=
  ', ', index_col=0)
8
9 #Merge both the panda dataframes on project_id
10 main_frame = pd.merge(projects_df, galleries_df, left_index=True,
  right_index=True)

```

4.3.2 Imputing missing data

As we have seen in our previous data analysis section, there are some missing values in our dataset. For our algorithm to work correctly we need to replace this missing data. For replacing the value we need to apply some data imputation techniques which estimates the best possible replacement by looking at neighbouring values.

- **Mean imputation** - In the mean imputation methods, the missing values are replaced by the mean of the observed variable. For example, if the variable of sound (number of sounds used in the project) is missing for a particular project, we take the mean of all the projects who have sound in them and replace the missing value with it.
- **KNN imputation** - In K nearest neighbour imputation, the missing values are replaced by the approximate value based on the nearest points. The assumption being that the given K nearest neighbour point the missing value is an approximation of it, k being a number of closest points to look for and is supplied by the user it is the most commonly used technique.

For our purpose we use SimpleImputer and IterativeImputer from sklearn for imputing the data values.

```

1
2 #Impute the missing values using IterativeImputer from sklearn
3 MAX_ITER = 10
4 imp = SimpleImputer(missing_values=np.nan, strategy='mean')
5 imp = IterativeImputer(max_iter=MAX_ITER, initial_strategy='knn',
  random_state=0)
6 imputed_DF = pd.DataFrame(imp.fit_transform(main_frame))

```



```

7 imputed_DF.columns = required_columns_df.columns
8 imputed_DF.index = required_columns_df.index

```

4.3.3 Correlation-based similarity

For correlation-based similarity computation, Pearson-r correlation method is applied. The `corr(method='pearson')` methods from the Pandas directly applies the selected correlation method on the dataset. The correlation matrix is obtained as shown in Figure 4.5. This matrix shows the item correlation between all the projects from the dataset.

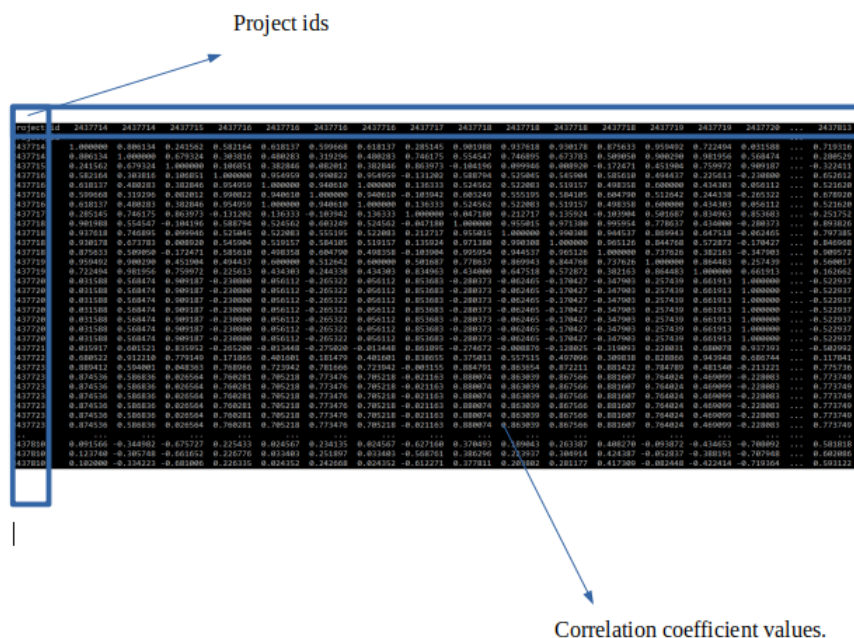


Figure 4.5: Pearson correlation output

Note the numerous negative correlation values in the matrix, this suggests that many of the projects are not similar at all.

4.3.4 Cosine-based similarity

For cosine based similarity computation we apply the `cosine_correlation` from the `sklearn.metrics.pairwise` module on the dataset. Figure 4.6 shows the output matrix

for cosine based similarity, This matrix shows the cosine correlation values for all the projects in the dataset.

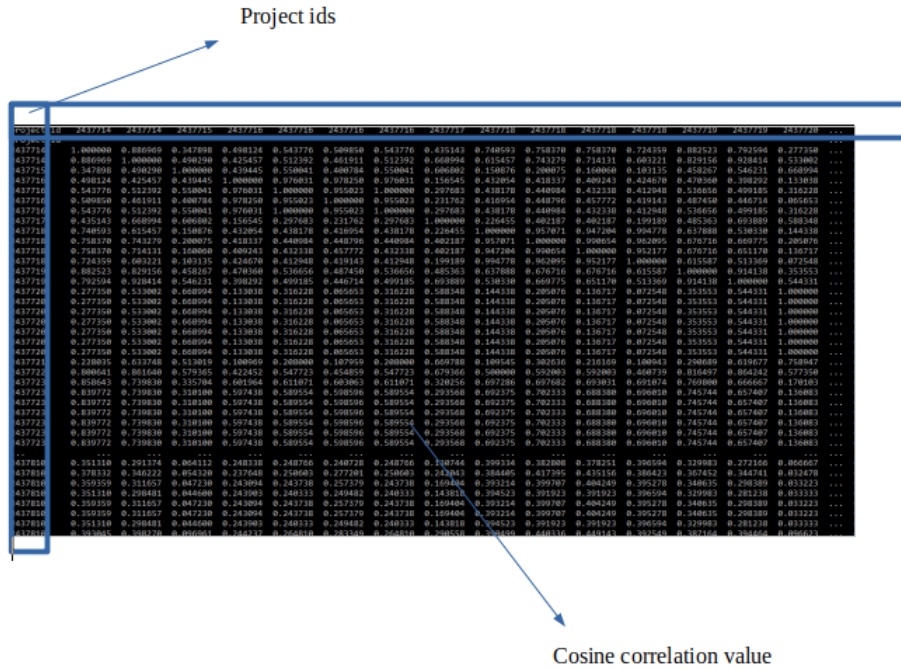


Figure 4.6: Cosine similarity computation output

4.3.5 Adjusted-cosine based similarity

Similar to the previous method after applying the *adjusted_cosine* functions from the sklearn library, we obtain the similarity matrix shown in Figure 4.7. The matrix shows correlation values of different items from dataset.

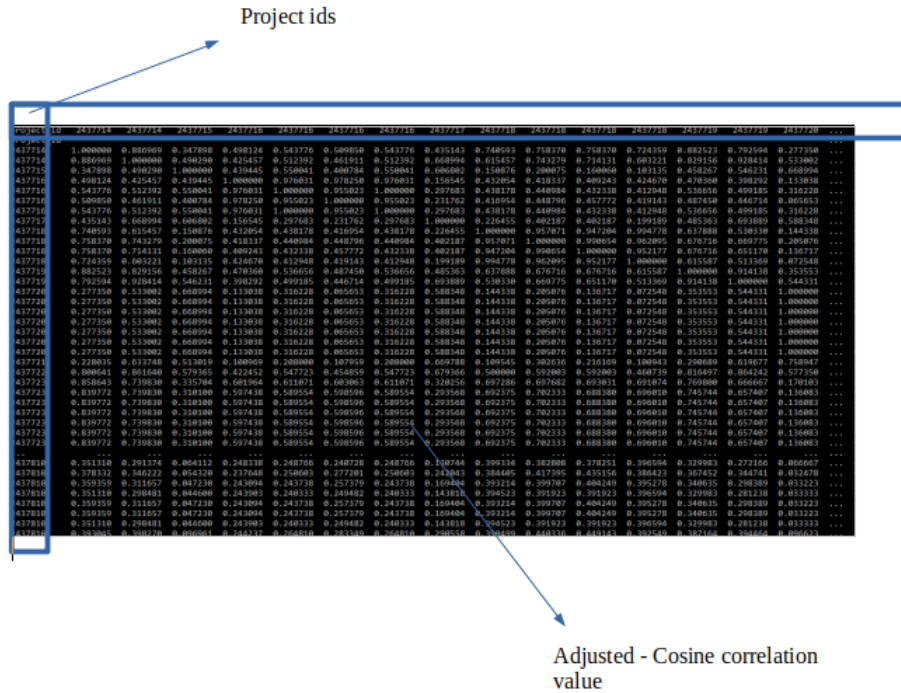


Figure 4.7: Adjusted cosine similarity output

Note here that since adjusted cosine is performed over columns rather than rows, the computation time for this method is lower compared to the other two. This computation time can be reduced by using a machine of higher computation memory or by reducing the size of the dataset

4.3.6 Selecting top-N recommendations

From the results of the similarity obtained above, for a given item i of a particular user we get the correlation score for each of the similar items from our dataset. We select the top - N values from the matrix ($N=5$) to be presented to the user U . This gives us the top 5 similar items to the user item I .

```

1 #Get Top 5 from the similarity matrix
2 top5 = final_df.loc[project_id].nlargest(5)
3 recommendation_pairs = top5.to_dict()

```

4.4 Prediction Calculation

After we have the item-item correlation matrix ready, we proceed to calculate the prediction score for each of the item based on users activity.

4.4.1 Weighted Sum method

Weighted sum method has been used for prediction calculation because of its simplicity to implement. For an item I for user U, this method calculates the prediction score by evaluating the sum of the ratings given by the user on other similar items to item I. Each rating by the user is assigned a weight $s_{i,j}$ based on the similarity output of the item.

```
1 #Calculates the prediction score for a user given recommendation as
  the list of similar recommended projects and target project_id
2 def calculate_prediction_score(user_id,project_id,
  similar_recommendation):
3     _w = [get_user_ratings(user_id)]
4     _x = similar_recommendation
5     score = sum( _x * _w for _x, _w in zip( x, w ) )/ calculate_score(_x
  )
6     return score
```

```
[522 rows x 173 columns]
Index(['scratchcomment', 'keyeventthatmorph', 'eventthatmorph_startclicked',
      'eventthatmorph', 'mouseclickeventthatmorph', 'whenhatblockmorph',
      'and_operator', 'multiply_operator', 'add_operator',
      'subtract_operator',
      ...
      'set_xpos', 'yourself', 'ypos', 'set_ypos', 'askyahoo', 'wordoftheday',
      'jokeoftheday', 'synonym', 'info_fromzip', 'scratchrinfo_foruser'],
      dtype='object', length=170)
project_id
2437714    1.000000
2437719    0.928414
2437771    0.919428
2437735    0.917011
2437735    0.895890
Name: 2437714, dtype: float64
```

Figure 4.8: Top 5 project ids according to prediction

The figure shows the top 5 most similar items computed after applying the weighted sum method. For the supplied user_id: xxxxxxxx and project_id: 2437714 all the other are the recommendation scores for the top 5 similar items.

4.4.2 Comparing the recommended items

On comparing the results that we get from the recommendation method we observe the following. We follow the link of each project ID to see the structure of the project. Every project comprises of similar set coding blocks as of the original project by the user U. For instance every project has one forever loop, one wait timer one costume switch. Also, note the two most similar projects have the sound block explicitly specified and while the next two have an implicit sound build in requiring no sound block. This gives us confidence about the similarity matrix outputs in the N-dimensional user space. It is also observed the second most similar project (id=2437719) is given priority because it was liked and remixed by our test user.

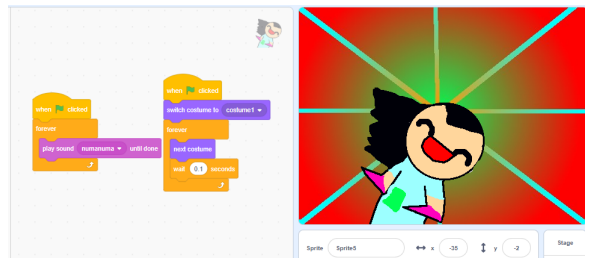
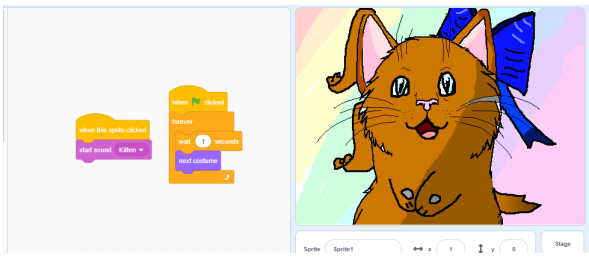
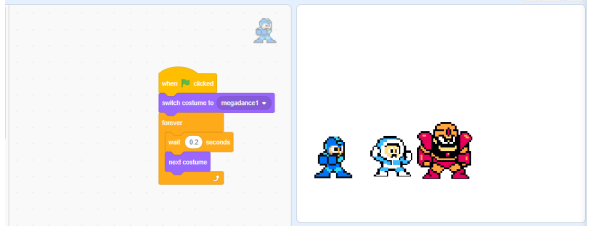
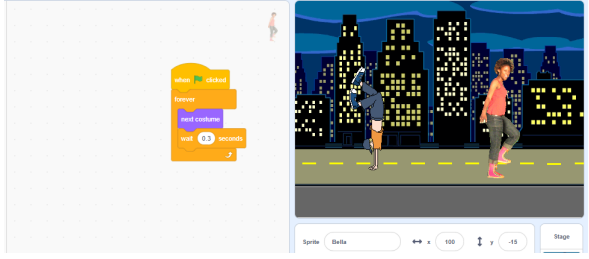
Project URL	Code Snip
<p data-bbox="279 398 874 436">https://scratch.mit.edu/projects/2437714/</p> <p data-bbox="343 660 810 698">Likes: 0, Favourite: 0, Remixes: 0</p>	
<p data-bbox="279 712 874 750">https://scratch.mit.edu/projects/2437719/</p> <p data-bbox="343 974 810 1012">Likes: 2, Favourite: 4, Remixes: 2</p>	
<p data-bbox="279 1025 874 1064">https://scratch.mit.edu/projects/2437771/</p> <p data-bbox="343 1265 810 1303">Likes: 3, Favourite: 3, Remixes: 0</p>	
<p data-bbox="279 1317 874 1355">https://scratch.mit.edu/projects/2437735/</p> <p data-bbox="343 1579 810 1617">Likes: 0, Favourite: 0, Remixes: 0</p>	

Table 4.3: Projects recommended

The next chapter will deal with evaluating and comparing the quality of these recommendations methods and prediction scores more formally. It discusses various different metrics for evaluating the recommendation methods and prediction scores.

Chapter 5

Method of Evaluation

This section discusses various evaluation methods that exist in the recommender system research paradigm to determine the quality and accuracy of the recommendation method discussed in previous chapter. We also compare the recommendation system based on the three similarity computation algorithms discussed previously.

5.1 Experiment

After building the model and applying the methods on our training dataset, we apply the methods on our test dataset for comparing the accuracy. From the dataset of over 1.9 million observations of projects and around 1 million users, the data set was divided into 80% as training data and 20% was used as test data for the model.

5.2 Evaluation methods

Various methods of evaluation have been studied in the paradigm of recommender system research. These methods are mainly used to evaluate the accuracy of a recommender system. These metrics can be broadly categorised into 2 groups as described below.

5.2.1 Statical accuracy metrics

A statistical accuracy measure is the one which evaluates the accuracy of a recommender system by calculating the difference in the actual user given score in the test data and the score calculated by the recommender system.

- **MAE(Mean Absolute Error)** is one of the most popular metrics that is widely used to compare the actual and expected values. Basically, MAE is the measure of the deviation of the predicted recommendation rating from the actual user given ratings. In the scratch dataset for each pair of $\{predicted, user - rating\}$, MAE handles the absolute difference between them equally i.e. it is given by $|redicted - user_rating|$. MAE is computed by summing the absolute difference for all the predicted, user-rating pairs and then averaging it over the total number of pairs. Mathematically it can be written as,

$$MAE = \frac{\sum (predicted\ ratings - user\ ratings)}{Total\ number\ of\ pairs}$$

- **Root Mean Squared Error (RMSE)** is another statistical accuracy metric that is used for measuring model performance. It was first used in the areas of environmental changes and climate research [41]. While MAE assigns the same weight to all the errors, RMSE penalizes large variance in the reading as it assigns errors with large absolute value more weight compared to the smaller values.

$$MAE = \sqrt{\frac{1}{n} \sum_{i=1}^n e_i^2}$$

5.2.2 Decision support accuracy metric

Decision support accuracy metrics are used to measure the usefulness of a recommendation engine at assisting the user in selecting the top-quality items from a list of all available items. They work by assuming the predictions to be a binary process. Either an item is a good one or a bad one, i.e. it is either predicted or excluded from prediction. If on a rating scale of 5 if a user only chooses items with a rating more than 4, it becomes irreverent if the rating is 2.5 or 3. Some of the most widely utilised support

accuracy matrices are ROC sensitivity, weighted error and reversal rate [33].

- **ROC sensitivity - Receiver Operating Characteristics curve** is a graphical plot that is used to visualise and select classifiers based on their performance. ROC graphs are being used since long time in the area of signal detection system for showing variance in a hit and false rates of the classifiers [42]. The medical community has a huge literature on the use of ROC for decision making on diagnostic testing. ROC curves are adopted in machine learning for comparing and evaluating various algorithms. ROC graphs have recently gained popularity in the machine learning community due to the understanding that simple accuracy metrics are a weak measure of the performance of the classifiers [43]. Besides of being useful as a performance mapping technique, they have additional qualities that make them suitable for problems with very skewed data distribution.

We begin with building a confusion matrix for the classifier, counting the number of correctly classified and incorrectly classified values into the matrix. As shown in figure 5.1 the true positive and true negatives are the correctly classified values by the classifier and false positive and false negative are the incorrectly classified values.

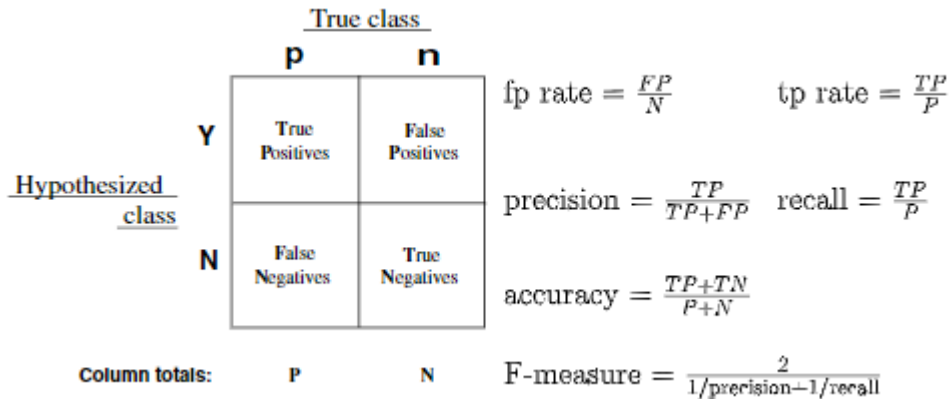


Figure 5.1: Performance metrics calculations from Confusion metrics.

Figure 5.1 shows various standard metrics that can be calculated from the confusion matrix. Formally it is defined as the area under the curve of the ROC curve.

ROC is plotted as sensitivity v/s (1- specificity) of the test. In case of a recommender system Sensitivity is defined as the probability of a random selection of item is accepted by the filtering method (a good prediction) and specificity is defined as the probability of a random selection of item being rejected by the filtering method (a bad prediction). A good filtering method would enable the user to select from 90% of the good recommendation and 10% of the bad ones.

- **Reversal rate** is defined as the frequency at which the recommendation by the system is wrong or incorrect. On a scale of 5, it is commonly given by the percentage of the recommendation where the prediction was varied by 3 or more points.
- **Weighted error** is the metrics which gives extra weight to the errors due to the strong outlook about the recommended item. For instance, if a user considers an item to be his favourite (i.e. a rating of 5 out of 5) the error for this could be considered double or more.

Some of the new and novel approaches have been discussed in [44] which are based on coverage and serendipity of the recommendations. It discusses the metrics beyond the accuracy of the recommendations and takes into consideration the quality and usefulness of the system

For the simplicity and ease of interpretation of MAE, we select it as our preferred choice of evaluation. In the experiment [33] Sarwar notes that MAE and ROC give the same arrangement of different experimental procedures with respect to prediction quality.

5.3 Results

5.3.1 Comparing the similarity algorithms

After running the three similarity computation methods namely cosine, adjusted-cosine and Pearson-r correlation as discussed in previous chapters on our training data and using the weighted sum method to get the predictions score, we calculate the MAE of

for each of the methods. Figure 5.2 shows the results of the three algorithms on the dataset.

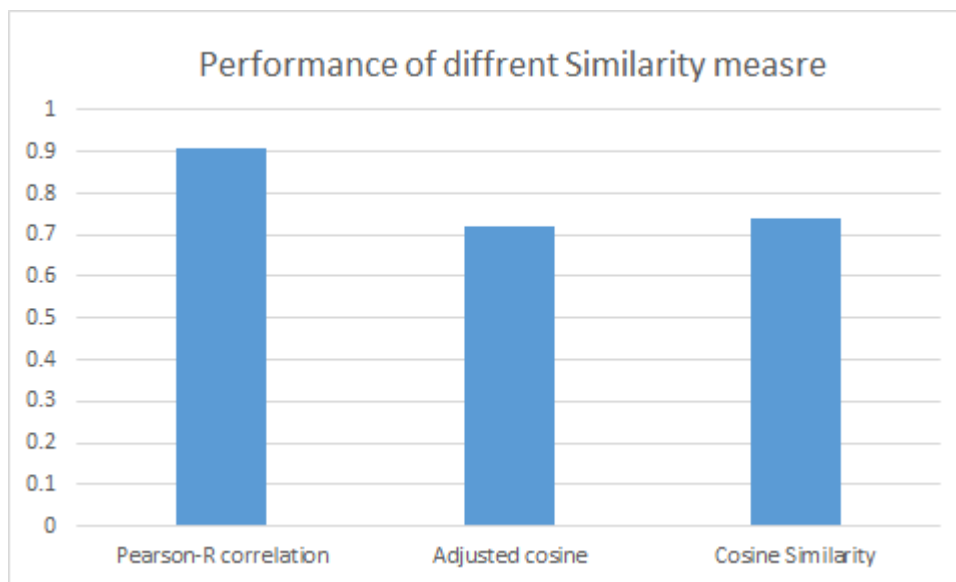


Figure 5.2: Performance comparison of the similarity calculation method

Since a lower value of MAE denotes a better performance, we observe the best performance is given by the adjusted cosine similarity method. However since the computation time required for the adjusted-cosine matrix calculation is high and the difference between Cosine and Adjusted cosine is relatively less we can say Cosine similarity is best suited for the Scratch dataset. Note that since the Person-correlation matrix gave us many negative values its MAE score is higher compared to other two methods.

The client-server model in appendix A which was built to demonstrate the working of recommender system uses cosine similarity implementation.

Chapter 6

Discussion and Future work

Although we have successfully displayed the working of the proposed recommendation methods, there are many more ways in which the recommendation system for a complex platform like Scratch could be developed. In the design of this item-based recommendation, we assume the item-item similarity and user rating are the defining characteristic of the user preference, however that may not be necessarily the case. In this chapter, we discuss the limitations of the current system and the possible improvements that could be made to improve the performance of current recommender system.

6.1 Limitations

6.1.1 Over specialisation problem

One of the major problem faced by collaborative filtering algorithms is the problem of over specialisation. A classic example of this is when one buys a lawn mover on amazon, he only sees recommendations for lawn mover. A good recommender should provide suggestions on a diverse set of items, rather than suggesting the same type of items that the user is already acquired or liked in past. The recommendation system proposed in this study suffers from this problem i.e. if a user has liked a certain type of project in Scratch, the recommended are generated for the project that has a similar characteristic. It prevents the user from discovering new items that might help them. Researchers have tried to solve the problem by neighbourhood based CF and hybrid

recommender systems [45].

6.1.2 Concentration bias

A concentration bias arises when a recommender system proposes the items with higher ratings more than with low ratings. Especially in CF algorithms which use user ratings to recommend the items, they often create the "*rich gets richer*" effect for more well-known products, this concentration bias limits the reach of lesser-known items to the user. Various researchers have tried to eliminate concentration bias problem from classical recommendation algorithms like k-NN filtering [45]. [46] suggests a probabilistic neighbourhood-based method popularly known as k-PN as an alternative which consistently outperforms k-NN in terms of concentration bias.

6.1.3 Cold Start problem

A cold start problem occurs when a new user or item enters the system and there is no information about it. Cold start problem can be broadly divided into three types new item problem, new user problem and new system problem [47]. In the case of a new user, it is very difficult to give a recommendation as there is no information available about the user. In the case of projects in scratch when a new item enters the system there is some information about the project, hence we can at least compute the similarity matrix for the item. However, without a user rating, we cannot calculate the prediction score. Content bases systems succeed in case of the new item as they do not depend on the user for generating the recommendations.

6.1.4 Dataset limitation

Over the years scratch has gone through many iterations of development. The first version (Scratch 0.x to 1.x) of the scratch was implemented in Squeak [48]. Later when Scratch 2.0 was launched in 2013 it shifted to ActionScript. The current version of scratch launched on January 2nd, 2019 i.e. Scratch 3.0 uses JavaScript for its implementation [49]. This dataset was scraped at the initial phase of Scratch from 2007 to 2012, hence it only contains projects from Scratch 1.x.

The development has led to many UI changes to the scratch development workspace

over time. UI may have a significant influence on user activity and skills development [50]. The quality of projects and user data might have been affected by this effect.

Also, many new attributes and features have been added to the newer version of the project files that give us the parameters for item similarity computation. Hence the more recent data collections effort is needed in order to accurately calculate the item similarity between projects.

A more recent dataset would give a better understanding of the user's mindset and may probably give a different result for different similarity algorithms. The dataset also provides very little information on the user rating part. Some implicit information like the clicks the user makes and the changes a user makes to different projects may help in gaining more knowledge about users preference.

6.2 Future work

6.2.1 Qualitative testing

As discussed in chapter 5 we evaluated and validated the prediction accuracy for the proposed recommendation techniques. Research in the recommendation system space mainly focuses on the prediction rating accuracy. And since accuracy only partly comprises the user experience of the recommender system, more qualitative testing is required for evaluating the usefulness of the technique. It needs to be verified against the need of the user who is getting the recommendation. One of the goals of developing this recommendation system for Scratch was to aid the user in their learning journey by suggesting projects that might improve their skills and the techniques they are learning. Some of the methods that could be used are an A/B testing amongst the users, collecting user data about the recommendations or by surveying the users for their opinion about the recommended items.

A/B testing

Also known as the bucket test or split-run test is a method to compare two versions, A and B of the same technique, commonly done on users response to the different versions to determine which one is more effective. Using the A/B testing methods different versions of recommendation can be served to users to see which one is more useful to

the user. For example, if the user chooses to view a more complex project instead of the more similar one to the original, this could give an insight of how to develop a recommendation which will help the user in discovering more complex projects.

User Surveys

A user survey could be conducted to get an understanding of what the user thinks of the recommendations given to him/her. This could help us understand the user needs and performance parameters upon which we could tune the recommendation system. As noted in [49] preference elicitation is a difficult problem since many time a user is unaware of his own personal preference, especially when he is beginning to use the system or is unaware of all the available choices. [49] also suggests asking users a small set of goals in the signup process to understand the user's recommendation parameters for the individual.

Many novel methods have been proposed in [51] which focuses on the user-centric development of the recommendation systems. It proposes a framework that considers how the objective components of the system e.g. algorithms, are perceived subjectively and how these perceptions along with situational and personal characteristics, gives a different user experience leading to different interactions with the system. For example, if the user perceives the recommendation quality of these different algorithms differently, a higher perceived quality of the recommendation leads to higher interaction with the system.

6.2.2 Mining text from the dataset

While exploring the available dataset, we found that the text and code dataset contains much interesting information that might be useful in building a stronger recommender system. The comments posted on a project and galleries are available in `pcomments_text` and `gcomments_text` datasets. `Projects_strings` contain information about all the user-generated strings that have been used in the project. All these text-based information could yield some meaningful information if explored correctly.

A careful sentiment analysis of the comments might help us understand the kind of responses a project is getting. Based on which we could infer project usefulness to various users. In addition, comments on different project by the same user could help us understand users personal taste which could help in predicting user choices. Strings inside projects could help in categorising the project into various groups and sections, which in turn could help in building a cluster-based recommendation. Also, there is information available about the `tag_text` which is the tags assigned to the project by different users. This could also help in categorising the projects. TFIDF algorithms could be used to find the frequency of terms and then classify the projects into different categories.

6.2.3 Social network analysis

Scratch also has a large community website and a community forum where users interact with one another, ask for help and post useful information. Users can follow one another if they find the profile interesting. A social network graph analysis can be done on this information to find users with similar interests and user groups that are closely connected to identify similar interest groups. Followers can be recommended based on this information. Projects can also be recommended amongst the groups themselves based on the idea of being from the same circle.

6.2.4 Recommendation based on heuristics

CF-based and content-based recommender systems are well known in the field. However, they may not be perfect for every recommendation problem. Many different approaches need to be explored for building a good recommendation for scratch. One of the more simple and easy to implement approaches is the heuristics based. In a heuristics-based system, one could implement a different set of rules to recommend projects to users. For example, if the user is new in the system, we might recommend projects that guide the person in getting started and simpler projects. If the user is an experienced one and has built many complex projects, recommendations for projects from other users who have more complex projects in their profile may be recommended.

6.2.5 A better Rating mechanism

In the proposed system we have assigned a 3 point rating system depending upon the user liking, favouriting and remixing the project. Perhaps this is one of the most basic and rudimentary approaches for obtaining user ratings. Because Scratch does not have any direct rating mechanism such as required in any traditional user based recommender system, it is necessary to develop a comprehensive mechanism that truly reflects the user's choices. One approach could be assigning weights to various activities that a user performs on a project such as likes and remixes. This weights could be obtained based on careful observations of user activities. For example, if a user performs a remix of a project only if he/she is really interested in it, more weight could be given to the remix activity. And if the user performs likes more frequently to most of the projects just to show the appreciation the weight of like activity could be reduced. Also, more user activities could be included such as commenting and tagging on a project for more granularity on the rating scale. Other methods could also be explored for more accurate representation of user's taste

Chapter 7

Conclusion

7.1 Summary

This dissertation explores the concept of recommender systems for Scratch. It briefly describes the motivation for the study and focuses on the problem of content discovery and self-directed learning in the Scratch environment. The idea of a recommender system might look like a simple problem from the top but it requires an understanding of the underlying data analysis concepts. Various different recommender systems used in different fields have been discussed. Specifically the ones in the area of e-learning and online teaching, which have gained popularity and are used extensively. Different studies have attempted to develop a recommender system for Scratch in their own teaching context.

Over the period, various research has been done in the area of a recommender system. Various techniques have been proposed for building recommend systems for different applications. This thesis discusses the standard techniques in recommender system space and their uses. Content-based filtering, collaborative filtering and an hybrid approach are most widely used and accepted as standard amongst the community. A Collaborative filtering process is used and applied on the Scratch data to try and build an item-based collaborative filtering recommender system. Basic tasks in a recommender system like generating a recommendation list and calculating the prediction are detailed for the dataset.

For generating a recommendation list in an item-based collaborative technique, the

first step is similarity computation. Three different techniques are used for similarity computation. Cosine based similarity, correlation-based similarity and adjusted cosine based similarity are the three well-known techniques for computing the item similarity. These three techniques were applied on the dataset to get the item similarity matrix for all the project a user has on his profile. Once the item similarity matrix was generated, prediction calculation was done based on the weighted sum method. Prediction output gave us the predicted recommendation rating for the user. Once the prediction was generated. the projects were recommended to the user according to the score.

Different methods of evaluation for the accuracy of recommendation are discussed. For measuring the accuracy of the recommendation we use MAE as the evaluation metrics. While comparing the three similarity calculation methods, it is observed that the cosine similarity is the best performing one on the Scratch dataset.

When we look at the top 5 generated recommendation of Scratch projects similar to the supplied item i.e our seed project, we observe some similarities between the recommended items. All the items had a similar set of script blocks and sprite usages. This shows the correctness of the similarity computation blocks in N-dimensional user space.

Although this thesis demonstrates a working model of a recommendation system for scratch, much more work is needed to validate the usefulness of the proposed methods. Many more useful techniques could be developed by incorporating the user's behaviour over time. This method of recommendation is more prone to general problems faced by recommender systems. Limitations like over specialisation problem, the cold start problem and the concentration bias limit the work of the proposed method. Moreover, the quality of the data available also affects largely on the prediction results. If the data has missing values for the user ratings, predictions cannot be calculated correctly by the algorithms.

Future work for this study would involve qualitatively testing the proposed method for the usefulness of the system. Various testing methodologies could be adopted like a/b testing, user surveys or other novel techniques as discussed in [51]. Also, more data could be extracted from the datasets to gain more information about the user, like a text analysis, or a sentiment analysis could be done on the text data of the projects for understanding the user's preference and taste. A more complex analysis could be a social network graph analysis on the follower-followee friendship to understand the

user interactions and taste. A heuristics-based recommendation could be a fast and effective alternative if developed correctly.

To demonstrate this a client/server model was developed (Appendix A). In the scripts, we use standard Scratch API and Python data analysis libraries and ML libraries for model building.

Bibliography

- [1] M. Resnick and K. Robinson, *Lifelong kindergarten: Cultivating creativity through projects, passion, peers, and play*. MIT press, 2017.
- [2] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond, “The scratch programming language and environment,” *ACM Transactions on Computing Education (TOCE)*, vol. 10, no. 4, p. 16, 2010.
- [3] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. S. Silver, B. Silverman, *et al.*, “Scratch: Programming for all.,” *Commun. Acm*, vol. 52, no. 11, pp. 60–67, 2009.
- [4] J.-M. Sáez-López, M. Román-González, and E. Vázquez-Cano, “Visual programming languages integrated across the curriculum in elementary school: A two year case study using scratch in five schools,” *Computers & Education*, vol. 97, pp. 129–141, 2016.
- [5] S. Cooper, W. Dann, and R. Pausch, “Teaching objects-first in introductory computer science,” in *ACM SIGCSE Bulletin*, vol. 35, pp. 191–195, ACM, 2003.
- [6] M. C. Carlisle, T. A. Wilson, J. W. Humphries, and S. M. Hadfield, “Raptor: introducing programming to non-majors with flowcharts,” *Journal of Computing Sciences in Colleges*, vol. 19, no. 4, pp. 52–60, 2004.
- [7] C. J. Bouras, V. Pouloupoulos, and V. Tsogkas, “Squeak etoys: Interactive and collaborative learning environments,” in *Handbook of research on social interaction technologies and collaboration software: Concepts and trends*, pp. 417–427, IGI Global, 2010.

- [8] D. J. Malan and H. H. Leitner, “Scratch for budding computer scientists,” *ACM Sigcse Bulletin*, vol. 39, no. 1, pp. 223–227, 2007.
- [9] I. F. de Kereki, “Scratch: Applications in computer science 1,” in *2008 38th Annual Frontiers in Education Conference*, pp. T3B–7, IEEE, 2008.
- [10] H.-H. Chen, I. Ororbia, G. Alexander, and C. L. Giles, “Expertseer: A keyphrase based expert recommender for digital libraries,” *arXiv preprint arXiv:1511.02058*, 2015.
- [11] N. R. Council *et al.*, *How people learn: Brain, mind, experience, and school: Expanded edition*. National Academies Press, 2000.
- [12] L. Song and J. R. Hill, “A conceptual model for understanding self-directed learning in online environments,” *Journal of Interactive Online Learning*, vol. 6, no. 1, pp. 27–42, 2007.
- [13] A. Felfernig, K. Isak, K. Szabo, and P. Zachar, “The vita financial services sales support environment,” in *Proceedings of the national conference on artificial intelligence*, vol. 22, p. 1692, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.
- [14] H.-H. Chen, L. Gou, X. Zhang, and C. L. Giles, “Collabseer: a search engine for collaboration discovery,” in *Proceedings of the 11th annual international ACM/IEEE joint conference on Digital libraries*, pp. 231–240, ACM, 2011.
- [15] N. Manouselis, H. Drachsler, R. Vuorikari, H. Hummel, and R. Koper, “Recommender systems in technology enhanced learning,” in *Recommender systems handbook*, Springer, 2011.
- [16] F. Ricci, L. Rokach, and B. Shapira, “Introduction to recommender systems handbook,” in *Recommender systems handbook*, pp. 1–35, Springer, 2011.
- [17] B. M. Sarwar, G. Karypis, J. A. Konstan, J. Riedl, *et al.*, “Item-based collaborative filtering recommendation algorithms.,” *Www*, vol. 1, pp. 285–295, 2001.

- [18] A. Zielinski, “A utility-based semantic recommender for technology-enhanced learning,” in *2015 IEEE 15th International Conference on Advanced Learning Technologies*, pp. 394–396, IEEE, 2015.
- [19] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, “analysis of recommendation algorithms for e-commerce, in proceeding of the 2th acm, e-commerce conference (ec00), october 2000,” 2000.
- [20] C. Jesennia, A. Puris, D. Benavides, *et al.*, “Recommender systems and scratch: An integrated approach for enhancing computer programming learning,” *IEEE Transactions on Learning Technologies*, 2019.
- [21] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, “GroupLens: an open architecture for collaborative filtering of netnews,” in *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pp. 175–186, ACM, 1994.
- [22] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, and J. Riedl, “GroupLens: applying collaborative filtering to usenet news,” *Communications of the ACM*, vol. 40, no. 3, pp. 77–87, 1997.
- [23] U. Shardanand and P. Maes, “Social information filtering: algorithms for automating” word of mouth”,” in *Chi*, vol. 95, pp. 210–217, Citeseer, 1995.
- [24] W. Hill, L. Stead, M. Rosenstein, and G. Furnas, “Recommending and evaluating choices in a virtual community of use,” in *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 194–201, ACM Press/Addison-Wesley Publishing Co., 1995.
- [25] H. Tan, J. Guo, and Y. Li, “E-learning recommendation system,” in *2008 International Conference on Computer Science and Software Engineering*, vol. 5, pp. 430–433, IEEE, 2008.
- [26] A. Pongpech, M. E. Orlowska, and S. W. Sadiq, “Personalized courses recommendation functionality for flex-el,” in *Seventh IEEE International Conference on Advanced Learning Technologies (ICALT 2007)*, pp. 631–633, IEEE, 2007.

- [27] J. Lu, “A personalized e-learning material recommender system,” in *International Conference on Information Technology and Applications*, Macquarie Scientific Publishing, 2004.
- [28] C. Limongelli, M. Lombardi, and A. Marani, “Towards the recommendation of resources in coursera,” in *Intelligent Tutoring Systems: 13th International Conference, ITS 2016. Proceedings*, vol. 9684, p. 461, Springer, 2016.
- [29] A. Kardan, A. Narimani, and F. Ataiefard, “A hybrid approach for thread recommendation in mooc forums,” *International Journal of Social, Behavioral, Educational, Economic, Business and Industrial Engineering*, vol. 11, no. 10, pp. 2195–2201, 2017.
- [30] J. Cardenas, “Scratch and caramba video.”
- [31] A. Gunawardana and G. Shani, “A survey of accuracy evaluation metrics of recommendation tasks,” *Journal of Machine Learning Research*, vol. 10, no. Dec, pp. 2935–2962, 2009.
- [32] P. Cremonesi, Y. Koren, and R. Turrin, “Performance of recommender algorithms on top-n recommendation tasks,” in *Proceedings of the fourth ACM conference on Recommender systems*, pp. 39–46, ACM, 2010.
- [33] B. M. Sarwar, J. A. Konstan, A. Borchers, J. Herlocker, B. Miller, and J. Riedl, “Using filtering agents to improve prediction quality in the grouplens research collaborative filtering system,” in *in the GroupLens Research Collaborative Filtering System???. Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW)*, 1998.
- [34] J. S. Breese, D. Heckerman, and C. Kadie, “Empirical analysis of predictive algorithms for collaborative filtering,” in *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pp. 43–52, Morgan Kaufmann Publishers Inc., 1998.
- [35] C. C. Aggarwal, J. L. Wolf, K.-L. Wu, and P. S. Yu, “Horting hatches an egg: A new graph-theoretic approach to collaborative filtering,” in *Proceedings of the fifth*

- ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 201–212, ACM, 1999.
- [36] C. Zaiontz, “real-statistics.com basic concepts of correlation,” 2019.
- [37] “Scratch research data.”
- [38] “nature.com.”
- [39] “scikit-learn.”
- [40] “Python data analysis library.”
- [41] T. Chai and R. R. Draxler, “Root mean square error (rmse) or mean absolute error (mae)?—arguments against avoiding rmse in the literature,” *Geoscientific model development*, vol. 7, no. 3, pp. 1247–1250, 2014.
- [42] J. P. Egan, *Signal detection theory and ROC-analysis*. Academic press, 1975.
- [43] F. J. Provost, T. Fawcett, *et al.*, “Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions,” in *KDD*, vol. 97, pp. 43–48, 1997.
- [44] M. Ge, C. Delgado-Battenfeld, and D. Jannach, “Beyond accuracy: evaluating recommender systems by coverage and serendipity,” in *Proceedings of the fourth ACM conference on Recommender systems*, pp. 257–260, ACM, 2010.
- [45] S. Jain, A. Grover, P. S. Thakur, and S. K. Choudhary, “Trends, problems and solutions of recommender system,” in *International Conference on Computing, Communication & Automation*, pp. 955–958, IEEE, 2015.
- [46] P. Adamopoulos and A. Tuzhilin, “On over-specialization and concentration bias of recommendations: Probabilistic neighborhood selection in collaborative filtering systems,” *RecSys 2014 - Proceedings of the 8th ACM Conference on Recommender Systems*, pp. 153–160, 10 2014.
- [47] L. Sharma and A. Gera, “A survey of recommendation system: Research challenges,” *International Journal of Engineering Trends and Technology (IJETT)*, vol. 4, no. 5, pp. 1989–1992, 2013.

- [48] “Development of scratch 1.0.”
- [49] “Scratch3.0.”
- [50] D. Cosley, S. K. Lam, I. Albert, J. A. Konstan, and J. Riedl, “Is seeing believing?: how recommender system interfaces affect users’ opinions,” in *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 585–592, ACM, 2003.
- [51] B. P. Knijnenburg, M. C. Willemsen, Z. Gantner, H. Soncu, and C. Newell, “Explaining the user experience of recommender systems,” *User Modeling and User-Adapted Interaction*, vol. 22, no. 4-5, pp. 441–504, 2012.

Appendix A

```
1 #!/usr/bin/python
2 #
3 # Created on Sat Jul 4 20:31:00 2019
4 # Copyright (C) 2019, shaikhr@tcd.ie
5 # @author: shaikhr
6 #
7 # API reference guide : https://github.com/LLK/scratch-rest-api/wiki/Projects
8 # Scratch data guide  : https://communitydata.science/scratch-data/
9
10 #Libs for Serving
11 from http.server import HTTPServer, BaseHTTPRequestHandler
12 from urllib.parse import urlparse, parse_qs
13 from io import BytesIO
14 import urllib.request, json
15
16 #not required for now can be commented
17 from selenium import webdriver
18 from selenium.webdriver.chrome.options import Options
19 import time
20
21 #Libs for data analysis
22 import pandas as pd
23 from sklearn.impute import SimpleImputer
24 from sklearn.experimental import enable_iterative_imputer
25 from sklearn.impute import IterativeImputer
26 from sklearn import preprocessing
27 from sklearn.metrics.pairwise import euclidean_distances
28 from sklearn.metrics.pairwise import cosine_similarity
29 from scipy.spatial.distance import correlation
```

```

30
31
32 #Returns a list of the users that the specified user has followed.
33 def get_user_following(username, LIMIT):
34     if not LIMIT:
35         LIMIT = 10
36     OFFSET = 0
37     with urllib.request.urlopen("https://api.scratch.mit.edu/users/"+
38         username+"/following?limit="+str(LIMIT)+"&offset="+str(OFFSET)) as
39         url:
40         data = json.loads(url.read().decode())
41         return data
42
43 #Returns an array of details regarding the projects that a given user
44     has favourited on the website.
45 def get_user_favriots(username, LIMIT):
46     if not LIMIT:
47         LIMIT = 3
48     OFFSET = 0
49     with urllib.request.urlopen("https://api.scratch.mit.edu/users/"+
50         username+"/favorites?limit="+str(LIMIT)+"&offset="+str(OFFSET)) as
51         url:
52         data = json.loads(url.read().decode())
53         return data
54
55 #Returns an array with information regarding the projects that a given
56     user has shared on the Scratch website.
57 def get_user_projects(username, LIMIT):
58     if not LIMIT:
59         LIMIT = 10
60     OFFSET = 0
61     with urllib.request.urlopen("https://api.scratch.mit.edu/users/"+
62         username+"/projects?limit="+str(LIMIT)+"&offset="+str(OFFSET)) as
63         url:
64         data = json.loads(url.read().decode())
65         return data

```

```

61
62 #Returns an array with information regarding the project with provided
    ID of particular username (dosent work , requires auth)
63 def get_user_project_details(username,project_id):
64     with urllib.request.urlopen("https://api.scratch.mit.edu/users/"+
        username+"/projects/"+project_id) as url:
65         data = json.loads(url.read().decode())
66         return data
67
68
69 #Gets the project name from given project ID (api deprecated , using
    selenium to render JS)
70 def get_project_title(project_id):
71     url = 'https://scratch.mit.edu/projects/'+str(project_id)
72     #rendered page is a JS script, no use of lxml #alternative is using
        selenium
73     chrome_options = Options()
74     chrome_options.add_argument("--headless")
75     driver = webdriver.Chrome('./selenium/chromedriver',chrome_options=
        chrome_options) # Optional argument, if not specified will search
        path.
76     #driver = webdriver.Chrome('./selenium/chromedriver.exe')
77     driver.get(url)
78     time.sleep(5)
79     title = driver.title
80     driver.quit()
81     return title
82
83
84 # ( No working API, therefor using lxml html parsing)
85 def get_author_from_project(project_id):
86     url = 'https://scratch.mit.edu/projects/'+str(project_id)
87     #rendered page is a JS script, no use of lxml #alternative is using
        selenium
88     chrome_options = Options()
89     chrome_options.add_argument("--headless")
90     driver = webdriver.Chrome('./selenium/chromedriver.exe',
        chrome_options=chrome_options) # Optional argument, if not
        specified will search path.

```

```

91 #driver = webdriver.Chrome('./selenium/chromedriver.exe')
92 driver.get(url)
93 time.sleep(5)
94 author = driver.find_element_by_xpath("//div[@class='title']/a").
    text
95 driver.quit()
96 return author
97
98
99 #Takes project json object as input and returns the stats if present.
100 def get_project_stats(project_obj):
101     #Difficult to retrieve the below values from current APIs therefor
    assigning mock values.
102     sprites_website = 5
103     scripts_website = 10
104     blocks = 6
105     block_types = 7
106     images = 2
107     sounds = 2
108     ugstrings = 11
109     stats = {'sprites_website':sprites_website,'scripts_website':
    scripts_website,'blocks':blocks,'block_types':block_types,'images'
    :images,'sounds':sounds,'ugstrings':ugstrings}
110     for key in project_obj:
111         if key == 'id':
112             stats['id'] = project_obj[key]
113         if key == 'stats':
114             stats.update(project_obj[key])
115         return stats
116     return stats
117
118
119 #Checks if the supplied project has a remix associated with it. If yes
    returns the remix object containing the remixes
120 def is_project_remix(project_obj):
121     for key in project_obj:
122         if key == 'remix':
123             #Nested for not too heavy because only 2 remix has only two
    components

```

```

124     for x in project_obj[key]:
125         if project_obj[key][x] != None:
126             return project_obj[key]
127         else:
128             return False
129
130
131 #Gets from the CSV database against the supplies stats of the project
132 def generate_recommndation_from_db(stats_obj):
133     #Load data from the csv
134     projects_df = pd.read_csv('data/CSVs/projects.csv', sep=',',
135                               index_col=0, nrows=100)
136     #index = projects_df.index
137     #columns = projects_df.columns
138     #values = projects_df.values
139
140     #Filter columns
141     required_columns_df = projects_df[['viewers_website', 'lovers_website',
142                                       'downloaders_website', 'sprites_website', 'scripts_website',
143                                       'blocks', 'block_types', 'images', 'sounds', 'ugstrings']]
144
145     project_id = stats_obj.pop('id')
146
147     new_stats_row = pd.Series(stats_obj)
148     new_stats_row.name = project_id
149     required_columns_df = required_columns_df.append(new_stats_row)
150
151     #Impute the missing values using IterativeImputer from sklearn
152     #imp = SimpleImputer(missing_values=np.nan, strategy='mean')
153     imp = IterativeImputer(max_iter=10, initial_strategy='most_frequent',
154                            random_state=0)
155     imputed_DF = pd.DataFrame(imp.fit_transform(required_columns_df))
156     imputed_DF.columns = required_columns_df.columns
157     imputed_DF.index = required_columns_df.index
158
159     #Normalize the data for further calculation
160     x = imputed_DF.values
161     min_max_scaler = preprocessing.MinMaxScaler()
162     x_scaled = min_max_scaler.fit_transform(imputed_DF)

```

```

159 normalised_df = pd.DataFrame(x_scaled)
160 normalised_df.columns = required_columns_df.columns
161 normalised_df.index = required_columns_df.index
162
163 #Calculate the RMSE score
164 normalised_df['RMSE'] = pd.Series((normalised_df.iloc[:,1:]**2).sum
    (1).pow(1/2))
165 print(normalised_df)
166
167 #Generate the similarity matrix based on the the new project stats.
168 sim = cosine_similarity(normalised_df)
169 final_df = pd.DataFrame(sim)
170 final_df.columns = required_columns_df.index
171 final_df.index = required_columns_df.index
172 print(final_df)
173
174 #Get Top 5 from the similarity matrix
175 top5 = final_df.loc[project_id].nlargest(5)
176 recommendation_pairs = top5.to_dict()
177
178 return recommendation_pairs
179
180
181 #Calculates recommendation score by taking the mean of the generated
    recommendation matrix based on provided input stats of the project
    .
182 #Need more optimized method to calculate more accurate score.
    Currently needs work.( Probably apply K nearest neighbor technique
    to get nearest neighbor and calculate further score.
183 def calculate_recommendation_score(stats_obj):
184     MAX_ITER = 10
185     RECOMMENDATION_NUMBER = 5
186     projects_df = pd.read_csv('data/CSVs/projects.csv', sep=',',
        index_col=0, nrows=100)
187
188 #Filter columns
189 required_columns_df = projects_df[['viewers_website', 'lovers_website
    ', 'downloaders_website', 'sprites_website', 'scripts_website', '
    blocks', 'block_types', 'images', 'sounds', 'ugstrings']]

```



```

190
191 #project_id = stats_obj.pop('id')
192 project_id = '10000001'
193
194 new_stats_row = pd.Series(stats_obj)
195 new_stats_row.name = project_id
196 required_columns_df = required_columns_df.append(new_stats_row)
197
198 #Impute the missing values using IterativeImputer from sklearn
199 #imp = SimpleImputer(missing_values=np.nan, strategy='mean')
200 imp = IterativeImputer(max_iter=MAX_ITER, initial_strategy='
    most_frequent', random_state=0)
201 imputed_DF = pd.DataFrame(imp.fit_transform(required_columns_df))
202 imputed_DF.columns = required_columns_df.columns
203 imputed_DF.index = required_columns_df.index
204
205 #Normalize the data for further calculation
206 x = imputed_DF.values
207 min_max_scaler = preprocessing.MinMaxScaler()
208 x_scaled = min_max_scaler.fit_transform(imputed_DF)
209 normalised_df = pd.DataFrame(x_scaled)
210 normalised_df.columns = required_columns_df.columns
211 normalised_df.index = required_columns_df.index
212
213 #Calculate the RMSE score
214 normalised_df['RMSE'] = pd.Series(((normalised_df.iloc[:,1:]**2).sum
    (1).pow(1/2))
215
216 #Generate the similarity matrix based on the the new project stats.
217 sim = cosine_similarity(normalised_df)
218 final_df = pd.DataFrame(sim)
219 final_df.columns = required_columns_df.index
220 final_df.index = required_columns_df.index
221
222 score = final_df.loc[project_id].nlargest(RECOMMENDATION_NUMBER)
223 final_score = 0
224 for x in score:
225     final_score += x
226

```

```

227     return final_score/RECOMMENDATION_NUMBER
228
229
230 #Reformats the stats object in the format required for the
      recommendation method
231 #DICT FORMAT: {'sprites_website': int64(xxx), 'scripts_website': int64
      (xxx), 'blocks': int64(xxx), 'block_types': int64(xxx), 'images':
      int64(xxx), 'sounds': int64(xxx), 'ugstrings': int64(xxx), '
      viewers_website': int64(xxx), 'lovers_website': int64(xxx), '
      downloaders_website': int64(xxx)}
232 def reformat_stats(stats_obj):
233     stats_obj['viewers_website'] = stats_obj.pop('views')
234     stats_obj['lovers_website'] = stats_obj.pop('loves')
235     stats_obj['downloaders_website'] = stats_obj.pop('favorites') #Need
      to fix this
236     stats_obj.pop('comments')
237     stats_obj.pop('remixes')
238     return stats_obj
239
240
241 #Generates recommendations for a given username, and returns a json
      object
242 def get_recommended_projects(username):
243     RECOMMENDATION_REASON_1 = "<i> Recommended because you follow: <a
      href=/users/"
244     RECOMMENDATION_REASON_2 = "<i> Recommended because you have project:
      <a href=/projects/"
245     RECOMMENDATION_REASON_3 = "<i> Recommended because user: <a href=/
      projects/"
246     RECOMMENDATION_REASON_4 = "<i> Recommended because its a remix of <a
      href=/projects/"
247
248     all_followers = get_user_following(username,10)
249     all_recommendation = []
250
251     #Gets recommendation from the csv database against the given user
      data
252     for projects in get_user_projects(username,5):
253         project_stats = get_project_stats(projects)

```

```

254     project_id = project_stats["id"]
255
256     recommendations_from_db = generate_recommendation_from_db(
257         reformat_stats(project_stats))
258     recommended_project_ids = recommendations_from_db.keys()
259
260     for recom in recommended_project_ids:
261         recommendations={}
262         recommendations['id'] = recom
263         recommendations['title'] = recom #get_project_title(recom)----->
264         #replace for getting actual name of the project insted of the ID
265         #but very slow because of selenium call
266         recommendations['stats'] = project_stats
267         recommendations['score'] = recommendations_from_db[recom]
268         recommendations['reason'] = RECOMMENDATION_REASON_2 + str(
269             project_id) + ">" + str(project_id) + "</a>"
270         recommendations['reason_id'] = username
271         all_recommendation.append(recommendations)
272
273     #Get details of the users who the user is following ( projects of
274     #following , favorites of following)
275     for user in all_followers:
276
277         #Projects from followers
278         all_follower_projects = get_user_projects(user["username"],5)
279         for project in all_follower_projects:
280             recommendations={}
281             recommendations['id'] = project["id"]
282             recommendations['title'] = project["title"]
283             recommendations['stats'] = project["stats"]
284             recommendations['score'] = calculate_recommendation_score(
285                 project["stats"])
286             recommendations['reason'] = RECOMMENDATION_REASON_1 + user["
287                 username"] + ">" + user["username"] + "</a>"
288             recommendations['reason_id'] = user["id"]
289             all_recommendation.append(recommendations)
290
291         #Projects that the followers have liked
292         all_followers_favourite = get_user_favriots(user["username"],5)

```

```

286     for favourite in all_followers_favourite:
287         recommendations={}
288         recommendations['id']= favourite["id"]
289         recommendations['title']= favourite["title"]
290         recommendations['stats'] = favourite["stats"]
291         recommendations['score'] = calculate_recommendation_score(
favourite["stats"])
292         recommendations['reason'] = RECOMMENDATION_REASON_3 + user["
username"] + ">" + user["username"] + "</a> has liked it."
293         recommendations['reason_id'] = user["id"]
294         all_recommendation.append(recommendations)
295
296     #Check if project has remix of the project, and add that to the
recommendation list.
297     for project in get_user_projects(user["username"],5):
298         is_remix = is_project_remix(project)
299         if is_remix:
300             recommendations={}
301             recommendations['id']= is_remix["parent"]
302             recommendations['title']= project["title"]
303             recommendations['stats'] = project["stats"]
304             recommendations['score'] = calculate_recommendation_score(
project["stats"])
305             recommendations['reason'] = RECOMMENDATION_REASON_4 + str(
is_remix["root"]) + ">" + str(is_remix["root"]) + "</a>"
306             recommendations['reason_id'] = user["id"]
307             all_recommendation.append(recommendations)
308
309     #Get details of the favorite project of the user ( No working API,
therefor using selenium for parsing).
310     #Too slow needs ,an alternative
311     all_favriots = get_user_favriots(username,5)
312     for favriots in all_favriots:
313         project_id = favriots["id"]
314         #get_author_from_project(project_id)
315
316     #print(all_recommendation)
317     return json.dumps(all_recommendation)
318

```

```

319
320 #Extends python's simple http server to handle GET requests locally
321 class SimpleHTTPRequestHandler(BaseHTTPRequestHandler):
322     def end_headers (self):
323         #Required for running locally on chrome ( allows CORS for all
324         request to the server )
325         self.send_header('Access-Control-Allow-Origin', '*')
326         BaseHTTPRequestHandler.end_headers(self)
327
328 #Gets the username in the GET request url as username and returns a
329 list of recommendations
330
331 def do_GET(self):
332     self.send_response(200)
333     self.end_headers()
334     query_components = parse_qs(urlparse(self.path).query)
335     username = query_components["username"][0]
336     response = BytesIO()
337     response.write(b'The username recived in URL is ')
338     response.write(bytes(username, 'utf-8'))
339     self.wfile.write(bytes(get_recommended_projects(username), 'utf-8'
340 ))
341
342
343 #Handels post request, wrote this to get data via POST, not used
344 currently
345
346 def do_POST(self):
347     content_length = int(self.headers['Content-Length'])
348     body = self.rfile.read(content_length)
349     self.send_response(200)
350     self.end_headers()
351     response = BytesIO()
352     response.write(b'This is POST request. ')
353     response.write(body)
354     self.wfile.write(response.getvalue())
355
356
357 if __name__ == "__main__":
358     try:
359         #NOTE: If you update port here, remember to update in the JS

```

```
extension as well.  
354     PORT = 8000  
355     HOST = 'localhost'  
356     httpd = HTTPServer((HOST, PORT), SimpleHTTPRequestHandler)  
357     print('Started httpserver on port ' + str(PORT))  
358     httpd.serve_forever()  
359  
360 except KeyboardInterrupt:  
361     print( '^C received, shutting down the web server')  
362     httpd.socket.close()
```