Serendipitous Recommendations for the Social Online Collaborative Network GitHub

Guillaume Viger

Department of Electrical & Computer Engineering
McGill University
Montreal, Canada

April 2015

A thesis submitted to McGill University in partial fulfillment of the requirements for the degree of Master of Engineering.

2015/04/03

Abstract

Github.com is a site where open-source projects can be freely hosted and where collaboration is mixed with other more explicit social features such as the capacity to follow other users or be followed. Serendipity is a recent recommendation engine criterion that seeks to measure how surprising and accurate are generated suggestions. The user-project interest links and user-user social links found on github.com provide a unique and realistic context in which to study serendipity since there is a large amount of data and chronological constraints must be respected. The presented thesis compares dissimilarity, unexpectedness and novel social distance based serendipity measures for recommendations made on a one year dataset of github.com's activities. We focus on recommendation approaches that rely on the network structure of the captured social network of users and interest network of user-projects. Item-based collaborative filtering and popularity recommenders are compared with adapted time-based link prediction approaches and a novel Markov chain algorithm. This first side-by-side comparison of serendipity and recall accuracy of graph-based algorithms shows that different serendipity measures favour different algorithms and that GitHub is a dynamic environment where new interests are greatly influenced by recent activities.

Abrégé

Github.com est un site web où des projets à code source ouvert peuvent être gratuitement hébergés et où collaboration et autres fonctionnalités sociales telles que suivre d'autres membres du site se mêlent. La sérendipité est un récent critère de mesure pour les engins de recommendation qui détermine à quel point les recommendations générées sont surprenantes et justes. Les liens d'intérêts entre membres et projets et les liens sociaux entre membres que l'on peut trouver sur github.com fournissent un environnement unique et réaliste où la sérendipité peut être étudiée; les données recueillies sont nombreuses et reflètent une chronologie d'évènements qui doit être respectée. Ce mémoire compare trois mesures de sérendipité basées sur la dissemblance, l'inattendu et, contribution originale, la distance sociale toutes trois appliquées à un ensemble de données correspondant à un an d'activités sur github.com. Nous nous concentrons sur des méthodes de recommendation qui reposent sur la structure du réseau social (membre-membre) et du réseau d'intérêts (membre-projet) obtenus à partir de ces activités. Des méthodes de filtrage collaboratif objets et de recommendations par popularité sont comparées avec des méthodes adaptées de prédiction de liens et un nouvel algorithme basé sur le principe de chaînes de Markov. Cette première comparaison de la sérendipité et du rappel d'algorithmes basés sur les graphes démontre que différents choix de définitions pour le concept de sérendipité favorisent différents algorithmes et que GitHub est un milieu très dynamique où les nouveaux intérêts sont grandement influencés par l'activité récente.

Acknowledgements

I owe many thanks to many people for their support and help in making this thesis.

First I want to thank my family. I am deeply appreciative of the material and not-so material support my parents provided me over all these years of study. My brother and sisters to whom I dedicate this thesis are my role models. Ever since my eldest sister Caroline started her Master's degree, I knew I would do one too. My brother Martin deserves thanks for pushing me toward a field that creates concrete value for others. Finally, I am very grateful to my youngest sister Marie-Élise who has always been there for me and especially so since I started at McGill while she was pursuing her own Engineering Master's degree. I am indebted to all of them.

I am sincerely thankful to my supervisor, Professor Michael (Mike) Rabbat. His approachability, openness to ideas and patience have made it a pleasure to work with him on this thesis. His guidance and comments have made the experience rewarding. I truly appreciated all of the four (!) classes I took with him at McGill.

I would also like to show my appreciation for the friendly and supportive atmosphere provided by my friends and colleagues past and present of the Computer Networks lab. I am grateful for Professor Mark Coates' presence in the lab and feedback on presentations. Thanks to Babak for thesis ideas when I started, Sean for the technical help early on, Benjamin and Arslan for sharing their experience working with a remote supervisor, Jay for the grading help, Milad and Deniz for their deep knowledge and good Iranian restaurant tip, Aida for the occasional chocolate, Shohreh and Naghmeh for the constant good cheer, Santosh for the company on Saturdays and the occasional ice-cream review on Tuesdays, Zhe for all the societal discussions and the computer help and Yunpeng for the mouth-watering chicken, the squash games and the willingness to try out many things.

Last I want to thank Sarah for getting me through some very rough months by her sheer presence. I heartily appreciate all you have done and I am deeply happy to know you.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

API	Application Programming Interface
JSON	Javascript Object Notation
SHA1	Secure Hashing Algorithm Version 1
RMSE	Root Mean Square Error
MAE	Mean Absolute Error
POP	Most Popular Recommender
POPSIM	Most Popular Among Similar Recommender
SIM	Most Similar Recommender
ICN	Interest Common Neighbour Recommender
IAA	Interest Adar-Adamic Recommender
IRA	Interest Resource Allocation Recommender
SCN	Social Common Neighbour Recommender
SAA	Social Adar-Adamic Recommender
SRA	Social Resource Allocation Recommender
MKV	Markov Chain Recommender
T<ALG>M	Time windowed version of <ALG> for last month
T<ALG>W	Time windowed version of <ALG> for last week
T<ALG>D	Time windowed version of <ALG> for last day

Chapter 1

Introduction

1.1 Motivation

The amount of information on the Internet is staggering, and discovering relevant, new and delightful information online is difficult. Internet retail stores, large media sites, online music or movie services, and online collaborative platforms index a significant amount of content, whether it be product dependent or user generated. Browsing through this information is time-consuming. Searching for specific content assumes a known goal and it can filter out potential unexpected discoveries.

In this setting of information overabundance, one of the most prevalent solutions has been to surface the most popular or most recent content. Such top lists expose a human-manageable selection of archived data to the public at large; the music top charts, movie box-office, and latest news are examples of such inherently biased surfacing of information. These charts are impersonal and leave no room for niche interests. Moreover, what is most popular is by very definition what has already been seen or consumed by the greatest number of people.

Recommender systems automatically generate targeted recommendations that stem from usage data. They are an avenue explored in the last fifteen years by some large companies such as Amazon in its 'Your Recommendations' section, Google in its tailored search results, Facebook in its Paper application[1] and Netflix in its custom movie suggestions. In all these cases, the underlying method is generally the same: predictive models

[1] https://www.facebook.com/paper

are generated by individuals' observed or implied interests and then are employed to rate other items—books, movies, news articles among others—where the ones with the highest predicted ratings, i.e., the ones deemed the most relevant, are suggested back to the original individuals. Properties inherent to the used content are also considered and social networks (backed by significant data) are starting to be used.

Generating recommendations that simply *and only* mimic the provided signals of users' interests can be criticized as pushing the pendulum too far the other way. Instead of repeating only what others find most appealing—as is the case in those top lists—, recommendation engines would now repeat only what oneself finds most appealing. In both solutions where one is only recommended the most popular items and where one is only recommended items very similar to one's history of interests, discovery of novel unexpected items is hindered.

Predicting exact future interests has been a long standing goal of recommender systems and more importantly a measure of their success. The different goal of acting as a discovery medium for novel items of potential interest has received less attention.

The discussed solutions to the problem of information overabundance are clearly not focused on discovery but rather on the desire to match a user's interest. Popular items have pleased many individuals and are thus likely to please new individuals. Recommendations too well tailored can trap individuals in their own taste enclosure. A balancing act between relevant and horizon-expanding recommendations must be reached to enable the better exploitation of the ever larger troves of information and provide a positive user experience.

Serendipity is the recommender systems' criteria most aligned with this goal. Serendipity is the sudden discovery of novel items of interest. The Merriam-Webster dictionary defines it as "luck that takes the form of finding valuable or pleasant things that are not looked for". Dissimilarity with past interest and unexpectedness with respect to another prediction approach have been used to assess serendipity, but quantitative comparative studies are lacking in the literature. A clear comparison of the effectiveness of different prediction approaches on these serendipity measures is needed.

A trove of potentially interesting information that would gain significantly by improved discovery mechanisms is github.com. Github.com, also known by its eponymous company name GitHub, is an online collaborative platform where open-source developers can host their projects for free and host private projects at a fee. Members of the site can collaborate with others on these projects as well as take advantage of other social features such as

following one another or showing their interest in repositories publicly by *starring* them —a form of public bookmarking. With close to 7.1 million users and more than 16 million hosted projects as of October 2014,[2] GitHub can provide a satisfying data sample on which to measure how well discovery processes fare and the impact of different serendipity measurements.

Discovery of code repositories on GitHub is a real concern marked by the presence of site features designed to address it and the ecosystem of third-party services attempting to reduce some of the friction in uncovering these valuable code repositories. In effect, the main purpose of the social aspects of GitHub are to enable one to stay abreast of new developments —interviews retrieved from past work and outlined in Chapter 2 confirm this usage. The ability to follow other users is not found in other open-source code hosting sites such as Bitbucket[3] or the late Google Code[4]. Furthermore, GitHub dedicates a section of their website[5] to the very objective of showcasing new projects to potentially interested developers. Third-party programming sites, e.g., Reddit[6], are frequently used to announce new projects and third-party newsletters such as The Changelog Weekly[7] use an editorial approach to surface interesting or up-and-coming projects.

These considerations combined with the presence of social links in the GitHub data and, most crucially, the availability of the GitHub data have guided our choice of dataset.

The GitHub data is naturally modelled via networks and graph-based approaches explicitly rely on the structure of a network to provide recommendations. Because of the novelty in using GitHub as a dataset, graph-based predictive approaches, also known as neighbourhood-based predictive approaches seem a promising avenue. Not only the interest network of user-objects, but the user-user social networks can thus be leveraged directly. These approaches have the advantage of mapping back to clear interpretations as well. This choice narrows our focus and allows further work to build on this base. Understanding the discovery processes with GitHub in mind can start from such an analysis.

[2] https://github.com/about/press
[3] https://bitbucket.org/
[4] https://code.google.com/
[5] https://github.com/explore
[6] https://www.reddit.com/r/programming
[7] http://thechangelog.com/weekly/

1.2 Problem Statement

Recommender systems are automated ways of suggesting relevant items (movies, songs, open-source projects in our case...) to a given user of a larger service provider. We are interested in the 'top-K' formulation of the predictive problem, which can be formally stated as:

Consider a set of M users $\mathcal{U} = \{u_1, \ldots, u_M\}$, a set of N items $\mathcal{I} = \{i_1, \ldots, i_N\}$ and a positive integer K. Given training data $\mathcal{T}$ subdivided into user training data $\mathcal{T}_u(t)$ for each user $u \in \mathcal{U}$ at time t, recommend a set of K relevant items $\mathcal{R}_u^K(t) \subset \mathcal{I}$ to user u.

In this thesis, we extract two novel user-user and user-item networks from hourly github.com activities of more than 46000 users and more than 180000 repositories spanning a one year period between February 2011 and February 2012. We use this collected data as the training data.

Relevancy of the recommendations is assessed by measures of recall—how many of the predicted suggestions are adopted by the targeted users— and serendipity—how unexpected are these predicted suggestions.

To understand the challenges with making serendipitous recommendations with this dataset one has to be cognisant of the context in which our exploratory study embeds itself.

The github.com data under study is implicit; a user interacts with the website and the generated records of these interactions are used to assess preferences without the user ever rating repositories explicitly or providing an explicit record of his or her discovery process. We induce from observed actions principles that can be used to predict actual or potential repositories of interest. We use implicit feedback to generate a list of recommended repositories.

Additionally, there is an inherent tension between providing unexpected recommendations and providing relevant recommendations. *Relevancy* stems from the interaction history and thus is achieved by providing interests similar to the past. *Unexpectedness*, by definition, is achieved by a recommendation algorithm that returns interests different from the past (or from what is *commonly* returned). Serendipity metrics aim to bridge these aspects in their assessments.

From the Github data we can extract social data in addition to preferences. This complementary network of interconnected users potentially provides new insights into the

discovery process. Homophily hypothetically connects users with strong links. Prediction methods must then be adapted to rely on multiple networks to output their suggestions.

Finally, the notion of time adds a further challenge to the lack of explicit ratings and the difficulty in defining serendipity. Time is an important factor in providing realistic recommendations: a suggestion embeds itself at a moment in time and a list of recommended repositories should therefore change over time. Repositories cannot be part of a suggested list if they have not been created yet. The main cost of keeping track of time is in increased computation and implementation complexity. An off-the-shelf approach dealing with time appropriately and that can be used in research directly is for the most part not readily available or needs significant alteration to be adapted to the specificities of a dataset similar to the studied one. As such, all of the explored approaches are implemented and adapted to recreate the realistic setting we are intent on modelling.

1.3 Thesis Contribution and Organization

The areas of investigation of this work are threefold; one of them is to conduct an experimental comparative study of serendipity on a novel dataset, another one is to use and compare graph-based techniques to generate predictions and the last one is to extract from this process some quantified understanding of novel unexpected discovery on GitHub.

The aim of the comparison of simple but solid and understandable predictive approaches is to lay a foundation for understanding and assessing the processes of discovery on GitHub. Because the dataset is new, we favour testing tried approaches with interpretive power in order to cover the basics. We conduct an experimental comparison of popularity based predictions, traditional collaborative filtering approaches, link prediction techniques and a simple novel Markov-chain approach to generate personalized serendipitous recommendations of open-source repositories. We compare their effectiveness across different proposed serendipity metrics: past dissimilarity, unexpectedness of results and a proposed novel measure considering social awareness. The recommendation process is made to mimic as closely as possible a real one by taking into account time and thus inscribing the predictions and their assessment over a timeline.

With serendipity being a hard property to quantify, we analyze the correlation between three different metrics to see if the attempts made at measuring serendipity without user survey reveal similar trends. Two of the considered metrics are (dis-)similarity-based

serendipity and unexpectedness-serendipity. The third is a novel social distance-based serendipity metric where the social network structure due to homophily is used to measure surprise.

To the best of our knowledge our work is the first to compare serendipitous measurements across a gamut of neighbourhood-generated list of predictions. Although GitHub data has been used in other research, our specific dataset is novel. It is tailored to mimic a real setting like few others are and it is used in this context for the first time.

In Chapter 2 of this thesis, we detail the common graph-based recommendation approaches and the notion of serendipity by framing them with respect to the wider literature. We review previous work on serendipity in recommender systems with a focus on recent attempts at defining it for top-K recommendations and identify how neighbourhood-based approaches are an appropriate avenue of exploration. In the following chapter, we explain our data collection process and the nature of the generated dataset. As this is a novel dataset, which we hope might be useful for further research, we provide ample information on the selection and cleaning process as well as the statistics of the data used. Chapter 4 describes all the techniques compared and how they are adapted to our real-time prediction setting. Chapter 5 presents the serendipity metrics used and the experimental results. We conclude the chapter with a comparative analysis of the different serendipity metrics and a discussion of what basic conclusions these entail as to the discovery processes at play on GitHub. We conclude the thesis with a summary of this exploratory research and potential future usage of the dataset as well as untested but potentially valuable recommendation techniques.

Chapter 2

Background and Related Work

In this chapter we begin by giving a summary of the notation used in the rest of this thesis. The various criteria by which recommendation systems have been measured are then listed in order to distinguish and hone in on the criteria of novelty, diversity and serendipity with focus on the latter. Novelty and diversity are common stepping stones to the concept of serendipity which we detail. Then we frame where our work resides in the vast body of literature on recommendation systems. We distinguish neighbourhood-based recommendation approaches from recent latent factors recommendation methods. In particular, we review recent serendipitous recommendation techniques for academic papers, songs and movies that illustrate these two categories with an emphasis on neighbourhood-based recommendation approaches. In the final section of this chapter relevant prior research on GitHub is summarized.

2.1 Notation

The notation used throughout this thesis, unless noted otherwise, is as follows.

Vectors

Vectors are column vectors by default. Vectors of n elements are typeset in lower-case bold with individual elements indexed with subscripts starting at 1:

$$\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}.$$

The transpose of a vector is:

$$\mathbf{x}^T = \begin{pmatrix} x_1 & \dots & x_n \end{pmatrix}.$$

The dot product between two vectors $\mathbf{x}$ and $\mathbf{y}$ is $\mathbf{x}^T\mathbf{y} = \sum_{i=1}^{n} x_i \cdot y_i$. The norm of a vector is denoted by: $\|\mathbf{x}\| = \sqrt{\mathbf{x}^T\mathbf{x}}$ which is the Euclidean length.

Sets

Sets of n elements are denoted by a calligraphic font, and their elements are indexed like vector elements: $\mathcal{S} = \{u_1, \dots, u_n\}$. Their cardinality (number of elements) is $|\mathcal{S}| = n$.

Conventions

Averages and predictions are denoted respectively by a line and a hat over the symbol: $\overline{x}$ for an average and respectively $\hat{x}$ for an estimate.

Long variable names for non-set or vector variables are in sans-serif font e.g.,

$$\mathsf{LongVariableName} = 4. \tag{2.1}$$

2.2 Recommender Systems Criteria

Recommender systems seek to predict from past purchasing, viewing, listening or other usage history the next items an individual would want to purchase, view, listen to or otherwise use. Originally, recommender systems were assessed primarily on their ability to predict users' likes or dislikes (e.g. movie ratings). The usefulness of a recommendation generator depended on predicting previous ratings or interests correctly and precisely, but McNee et al. [1] and Pariser [2] have broadened the discussion. Metrics that only assess how close a predicted rating is to the original rating have detracted attention from other worthy features that contribute to a positive experience.

Pariser [2] makes the point that current algorithms create a filter bubble where our own preferences are parroted back to us with little room for differing points of view. Individuals' taste and point of views are reinforced rather than expanded. Other desirable features exist beyond predicting the past accurately. In fact solely relying on this accuracy metric can be detrimental to the overall quality of the recommendations perceived by the user [1]. Multiple constraints and goals are at play in the type of real-time setting we investigate and must be addressed.

We further categorize the different assessment criteria outlined by reviews of the field of recommendation systems made by Herlocker et al. [3] and Ricci et al. [4]. For many of these assessment criteria, the data is divided into a training set $\mathcal{T}$ and a validating set $\mathcal{V}$. The training set is used to learn the solution model or determine directly the recommendations while the validating set is a set of withheld data that validates the recommendations generated according to the criterion of interest.

Accuracy and Error Metrics. Accuracy and error metrics measure how close the generated predictions are to the withheld usage data. Recall, precision and root mean square error (RMSE) or mean absolute error (MAE) are the basic accuracy metrics by which proposed recommendation techniques are often evaluated.

Recall measures the ratio of predicted correct recommendations over all actually correct (validating) recommendations. Let $\mathcal{U}$ be the set of users, $\mathcal{R}_u^K$ be the K-size set of recommendations for user $u \in \mathcal{U}$ and $\mathcal{V}_u$ be the set of validating withheld items for u. Overall recall is given by Equation 2.2 in the typical top-K recommendations setting.

$$\mathsf{TopKRecall} = \frac{1}{|\mathcal{V}|} \sum_{u \in \mathcal{U}} |\mathcal{R}_u^K \cap \mathcal{V}_u|. \tag{2.2}$$

The average user top-K recall is more representative of the efficacy for each user:

$$\overline{\mathsf{UserTopKRecall}} = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{|\mathcal{R}_u^K \cap \mathcal{V}_u|}{|\mathcal{V}_u|}. \tag{2.3}$$

However, recall does not indicate how well the system avoids recommending disliked items. Precision does by measuring the proportion of predicted correct recommendations over all predicted recommendations:

$$\overline{\mathsf{UserTopKPrecision}} = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{|\mathcal{R}_u^K \cap \mathcal{V}_u|}{K}. \tag{2.4}$$

When ratings are available, RMSE and MAE can be used to measure the average distance over all users between the current user u's rating $r_{u,i}$ for an item i and the predicted rating $\hat{r}_{u,i}$ for that same item:

$$\mathsf{RMSE} = \sqrt{\frac{\sum_{(u,i)\in\mathcal{P}}(r_{u,i} - \hat{r}_{u,i})^2}{|\mathcal{P}|}}, \tag{2.5}$$

$$\mathsf{MAE} = \frac{\sum_{(u,i)\in\mathcal{P}}|r_{u,i} - \hat{r}_{u,i}|}{|\mathcal{P}|}. \tag{2.6}$$

The set $\mathcal{P} = \{(u,i)|r_{u,i} \in \mathcal{T}\}$ is the set of observed (user, item) pairs.

Novelty, Diversity & Serendipity. These criteria measure how well the system reveals previously unknown items to the user (novelty), how different are those (diversity), and how much the interesting data surprisingly delights the user and broadens his or her interests (serendipity). Serendipity itself is nebulous and hard to automatically assess as it seems to be at odds with accuracy metrics. How can a proposed item be original and yet suggested from past recommendations? Conversely, how can a completely unrelated repository be selected to please the user? Serendipity is therefore often seen as a sum of novelty and diversity that is greater than its constituent parts. Direct user surveys have been used in the context of serendipity assessment e.g., Andre et al.'s work on web search serendipity [5]. We explore the literature associated with serendipity in the next section.

While our thesis focuses on the above criteria, what follows are the other criteria that have been proposed in the literature [3,4] to assess recommendations. These criteria should be kept in mind whenever one generates recommendations.

Coverage. Item coverage is the percentage or simply the number of all items that can be suggested by the system. Similarly, user coverage is the percentage or the number of all users for which recommendations can be made. The sparsity of data per user often limits how many users can be recommended against. Higher coverage also incurs higher computational costs.

Learning Rate & Computational Performance. This category of criteria describes how well the algorithms underlying the system learn with additional data, over time and across users. This category focuses on the actual engineering performance of the system. Can it return recommendations in a reasonable amount of time? Can it scale to millions of items and users? Can it be robust to failure? Answers to these questions may limit

the success of other criteria e.g., better accuracy is usually associated with more data, but processing today's large amounts of data is a significant performance challenge time, memory and money-wise.

Confidence & Trust. Confidence and trust are respectively the system's own assessment of the quality or accuracy of its predictions and the user's assessment of the credibility of the system. In other words, this category deals with the general reasonableness and convincing nature of the system. Explaining recommendations can help build trust and educate the user as to what he or she can do to receive better recommendations.

Agent satisfaction. This criteria expresses the users or the system stakeholders' satisfaction through experimental trials, surveys, analyzed behaviours and utility maximization. Sales revenue maximization, or risk-minimization can be seen as examples of goals whose level of attainment will quantify the system's success. This is a catchall category that includes a combination of the other criteria by letting the stakeholders express their satisfaction directly. The two main drawbacks of this method are the constraints of having a limited number of explicit feedback and the necessary interactive nature of the assessments which might prove to be too obtrusive in a practical context.

Our aim is to compare non-personalized popularity approaches and neighbourhood-based approaches in a realistic setting and so we take into account recall, empirically sound coverage, reasonable computational performance and user trust in support of the main serendipity criterion in our processing of the dataset and in our selection of algorithms.

2.3 Framing Serendipity

Novelty, diversity and serendipity are intertwined in the literature. Two overarching ways of characterizing the serendipity of recommendations have been used. On one hand, accuracy, novelty and diversity measures are taken. Serendipity is then said to be favoured when the measurements are all high. On the other hand, two distinct metrics (not compared as of yet) for measuring serendipity directly have been proposed. We summarize these two prevalent views on serendipity —measuring serendipity through other metrics and measuring serendipity directly— here.

2.3.1 Novelty

A recommended item is *novel* if it is not known by the user to whom it is recommended [4]. Recommending an item previously seen is a reminder medium rather than a discovery medium. In our work, we focus on the process of discovery. Recommending the most popular project on GitHub (currently the Bootstrap project https://github.com/twbs/bootstrap) to a developer, i.e. a GitHub user, is less likely to be effective since she will have presumably already been exposed to it. Moreover, if the developer did not show interest in the project already, it is because she is not interested.

Less strict definitions of novelty have also been explored to give more fine-grained results. Taking time into account allows for this. In Sugiyama and Kan's algorithm [6] to recommend scholarly papers, an exponential decay factor is used to emphasize more recent publications over older ones. As such, novelty can be time-dependent where a more recent item is more novel than an older item. These novel items are at the mercy of a dearth of usage data, however. Moreover, focusing on the latest items for the sake of their recency ignores the vast number of past items that might be more appropriate and still novel.

Another aspect to be weary of when considering novelty measures is the degenerate case: recommending all items a user has not seen will maximize pure novelty, but it is impractical for the user. It does not solve the original problem of information overload, it is not accurate and it can lead to mistrust in the system which culminates in non-usage.

2.3.2 Diversity

Diversity represents how dissimilar listed recommendations are to each other. This seemingly quite separate criterion has in fact been used as a proxy to serendipity [6, 7]. In a top-K list, it is often better to have diverse items in order to provide differing thematic options.

Zhang and Hurley [7] pose the problem of maximizing top-K diversity while maintaining similarity with a user's history as a binary optimization problem. The obtained quantization strategy used to solve the relaxation of the binary optimization problem results in what is interpreted to be more serendipitous[1] recommendations. Since users typically have interests that go beyond a single specific area, a *diverse* list is more apt to include these varied interest centres and go beyond into new areas of potential interest.

[1] 'Serendipity' is mentioned all but by name.

Where $\mathcal{R}_u^K$ is a user u's recommended list, $\mathcal{P}_i(u) = \{(i,j)|i,j \in \mathcal{R}_u^K, i \neq j\}$ is the set of all possible different pairs of items in this list, $\mathbf{p}_i$ and $\mathbf{p}_j$ are item profile vectors and $d : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is a dissimilarity function, Zhang and Hurley propose the following measure of average list diversity:[2]

$$\overline{\mathsf{diversity}} = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{|\mathcal{P}_i(u)|} \sum_{(i,j) \in \mathcal{P}_i(u)} d(\mathbf{p}_i, \mathbf{p}_j). \tag{2.7}$$

Item profile vectors, $\mathbf{p}_i$ and $\mathbf{p}_j$, are vectors containing a numerical interpretation of the various features of each respective item i and j.

Dissimilarity functions are usually the converse of similarity functions, the most common one being *cosine similarity*:

$$\mathsf{similarity}_{\cos}(\mathbf{p}_i, \mathbf{p}_j) = \frac{\mathbf{p}_i^T \mathbf{p}_j}{\|\mathbf{p}_i\| \|\mathbf{p}_j\|}. \tag{2.8}$$

The derived dissimilarity can then be:

$$d_{\cos -diff}(\mathbf{p}_i, \mathbf{p}_j) = 1 - \mathsf{similarity}_{\cos}(\mathbf{p}_i, \mathbf{p}_j) \tag{2.9}$$

or

$$d_{\cos -inv}(\mathbf{p}_i, \mathbf{p}_j) = \frac{1}{(\mathsf{similarity}_{\cos}(\mathbf{p}_i, \mathbf{p}_j) + k)} \tag{2.10}$$

where k is a positive scalar used to bound the dissimilarity result as suggested in [6].

Finally, we note that aiming for a diverse recommendation set can also be equated to representing the diversity of the original data. Rarity or uniqueness of certain types of data points (items or users) was used in early definitions of diversity [8]. Recommendations are more diverse if they cover a greater range of rarer items. This aspect is closely tied to the coverage criterion.

2.3.3 Serendipity

Serendipity measures how 'surprising successful recommendations are' according to the Recommender Systems Handbook of Ricci et al. [4]. Herlocker et al. [3] describe a serendipitous recommendation as one that is interesting but would not have been discovered otherwise.

[2] It is referred as 'item novelty' in their paper but it does not have the same meaning as the novelty criterion seen above.

Serendipity should therefore combine a measure of relevance and surprise. Ge et al. [9] propose a metric that mixes the two by taking into account *unexpectedness*. A serendipitous recommender should avoid returning obvious results. To do so a base algorithm referred to as a 'primitive method' is used to produce an *expected* recommendation list $\mathcal{E}_u^K$ for user $u \in \mathcal{U}$. The algorithm whose serendipity is measured produces a recommended list $\mathcal{R}_u^K$ that is compared to $\mathcal{E}_u^K$, to obtain the unexpected set of recommendations:

$$\mathcal{UXP}(u) = \mathcal{R}_u^K \setminus \mathcal{E}_u^K. \tag{2.11}$$

A recommendation's relevance is its usefulness according to Ge et al. [9]. This usually translates to whether or not the prediction is accurate. Lu et al. [10], in their measure of serendipity for Netflix movies recommendations, thus define the top-K serendipity as:

$$\overline{\mathsf{TopKSerendipity}_{\mathsf{unxp}}} = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{|\mathcal{UXP}(u) \cap \mathcal{V}_u|}{K}. \tag{2.12}$$

The only other explicit serendipity measure found in the literature focuses on the definition of serendipity by Herlocker et al [3]. Recommended items are judged to not have been discovered otherwise by a measure of their difference from past items. The more different a suggestion is from previously used, played or purchased objects, the more likely it is to not have been discovered through a casual usage of the providing service. This interpretation explains why the concept of serendipity has been evaluated through diversity: a recommendation list that is diverse is more likely to be different from the past and to contain niche objects.

Zhang et al. [11] and Sugiyama and Kan [6] assess serendipity by considering the distance between a suggested list and the history of items used to generate that list, $\mathcal{T}_u$. The distance function $d(\mathbf{p}_i, \mathbf{p}_j)$ is typically cosine similarity, but variations on graph distances have been used [12, 13]. Zhang et al., in their Auralist music recommendation system, quantify this distance while not having to use a bounded cosine similarity inverse by using Equation 2.13.

$$\overline{\mathsf{unserendipity}} = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{|\mathcal{T}_u|} \sum_{j \in \mathcal{T}_u} \frac{1}{|\mathcal{R}_u^K|} \sum_{i \in \mathcal{R}_u^K} \mathsf{similarity}_{\cos}(\mathbf{p}_i, \mathbf{q}_j) \tag{2.13}$$

A set of items different from what was seen before is more likely to catch a user's new centre of interests. This formulation shows the inherent tension between accuracy and

serendipity: an engine that recommends completely based on past interest would score highly according to Equation 2.13. This is addressed in [11] by measuring $\overline{\mathsf{TopKRecall}}$ as well.

In Nakatsuji et al.'s work [12] on expanding user interests in the domains of music, movies and restaurants, taxonomies are used. Taxonomies are expert-provided semantic classes grouping items together. They are especially useful in providing semantic reasoning behind recommendations. The serendipity provided by item $i \in \mathcal{I}$ is measured by taking into account the taxonomic class C_i to which item i belongs to and a distance function d that takes two taxonomic classes as inputs and returns an integer valued distance:

$$\mathsf{serendipity}(i, u) = \min(\{d(C_i, C_j) \mid j \in \mathcal{I}_u\}). \tag{2.14}$$

The distance between two taxonomic classes is determined by the distance between those classes on the tree connecting classes to each other based on their hierarchy.

In the context of web search, serendipity has been defined by Andre et al. [5] as interesting but irrelevant results from the point of view of the query at hand. This qualitative measure is gathered by user survey or interactive logging which are feedback methods beyond the scope of our approach. We also forgo relying on user queries in our setup.

2.4 Recommendation Approaches

As can be induced from the listing of criteria for recommendation systems given above, these tools to navigate large information spaces have a significant body of literature. Research on recommender systems materialized in the 1990's with the works of Belkin and Croft [14] and Goldberg et al. [15], and has since then progressed with the goal of taking into account the increasing and diverse amount of information available. The papers by Belkin and Croft and Goldberg et al. are credited with the introduction of collaborative filtering as a popular means of suggesting items to users.

Collaborative filtering relies on many individuals making use of a common service to filter and choose which items to show a specific user. The usage patterns, the explicit filtering by users (e.g., marking an email as spam or not) or the given ratings are recorded and used for the benefit of the other individuals interacting with the service. To this day, collaborative filtering is among the most effective means of recommending items to users

given a large amount of data and a few examples of interests per user.

In this context, each user and each item are typically represented by their features. The data makes up a profile used in various techniques. The user profile may be constructed by associating a positive or negative rating to each item used and/or by giving a numerical value to inherent properties of the user: male or female, age group, personal social network, etc. Similarly, item profiles may be constructed by storing the set of users who have reviewed the item, the co-purchased items and/or inherent attributes such as the item's category (book, movie, song...), the item's creator (author, singer...) and the item's genre (e.g., romance, action, rock...). These profiles are usually in the form of numerical vectors of equal length for each user or item.

Two main approaches to generating recommendations based on collaborative filtering can presently be distinguished: latent factor approaches and neighbourhood-based approaches. Although both rely on the user-item network, their means of going about recommending is quite different. We discuss these two approaches next.

2.4.1 Latent Factor Approaches

Latent factor approaches, also known as matrix factorization approaches, are recent extremely successful predictive techniques. Their effectiveness is mostly attributed to their use by the winning team of the Netflix prize competition.[3] From a set of training quadruplets of the form [<user>, <movie>, <rating>, <time>], contestants of the competition had to predict the true rating, $r_{u,i}$, given to a movie i by a user u and beat by a 10% margin Netflix's own Cinematch algorithm. The basic assumption underlying the winning approach of the Bellkor team [16] models the estimated rating, $\hat{r}_{u,i}$, as the dot product of the user and item's latent vector profiles $\mathbf{p}_u$ and $\mathbf{q}_i$ respectively:

$$\hat{r}_{u,i} = \mathbf{q}_i^T \mathbf{p}_u. \tag{2.15}$$

The problem becomes one of finding for all users and items their corresponding latent profile vectors such that the squared rating error—an equivalent of the RMSE—is minimized. Where $\mathcal{T}$ is the set of observed ratings and $\mathcal{P} = \{(u,i)|r_{u,i} \in \mathcal{T}\}$ is the set of observed (user, item) pairs, the squared rating error is minimized for all $\mathbf{p}_u$ denoted with the matrix $\mathbf{P}_u$ and all $\mathbf{q}_i$ denoted with the matrix $\mathbf{Q}_i$ by Equation 2.16.

[3] http://www.netflixprize.com/

$$\min_{\mathbf{Q}_i, \mathbf{P}_u} \sum_{(u,i)\in\mathcal{P}} (r_{u,i} - \mathbf{q}_i^T \mathbf{p}_u)^2 + \lambda(\|\mathbf{q}_i\|^2 + \|\mathbf{p}_u\|^2) \tag{2.16}$$

A regularization coefficient λ is used in this minimization equation in order to prevent overfitting to the training data. The obtained profile vectors can then be used to predict how a given user would rate a never-before-seen movie. Stochastic gradient descent and alternating least squares are efficient algorithms that can be used to solve equation 2.16. For details on these approaches and the many subsequent refinements to this model, we refer the interested reader to recent papers by Koren and co-authors [17,18]. Rating biases, time decay, neighbourhood size and influencing factors are typically added to Equation 2.15. Indeed, because of their effectiveness in predicting Netflix ratings, the latent factors approaches have garnered a significant amount of attention and become the state-of-the-art in the field.

Lu et al. [10] in particular have adopted this latent factor strategy to assess serendipity on the top-K recommendation problem applied to the Netflix movie and Yahoo! music datasets. A loss function $\ell()$ is defined to represent logistic or hinge loss. $\mathsf{Pop}(j)$ denotes the popularity of item j. The authors define $\mathcal{P}_t$, the set of user u, item i, item j preference triplets where u is assumed to prefer i over j because $r_{u,i}$ is above a rating threshold and $r_{u,j}$ is not observed or $r_{u,j} < r_{u,i}$. The authors maximize SAUC, a modified area under the receiver-operating curve specialized for serendipity.

$$\mathsf{SAUC} = \frac{1}{|\mathcal{P}_t|} \sum_{(u,i,j)\in\mathcal{P}_t} \ell(\hat{r}_{u,i} - \hat{r}_{u,j})(\mathsf{Pop}(j))^\alpha \tag{2.17}$$

SAUC is parameterized by α which tunes how much importance is assigned to cases where the user favours a less popular item to a more popular item. A greater α favours less popular items. By their rarity, these items are assumed to be more unexpected and therefore serendipitous.

Strict popularity prediction is used as the primitive method and $\overline{\mathsf{TopKSerendipity}_{\mathsf{unxp}}}$ is computed. Lu et al. [10] obtain superior precision and unexpectedness serendipity scores than by simply using regular latent factors approaches. It is observed that the exponent α in Equation 2.17 gives the best results when set close to 0.5, but not 1. Recommending from the long-tail too much, leads to recommendations that are truly not useful which decreases the serendipity score. Serendipity is a balancing act.

Latent factor approaches rely on factorizing the matrix of user-item ratings. To apply them one needs ratings, but there are no ratings of repositories on GitHub. Hu et al. [17] produce ratings from implicit TV watching feedback through frequency and duration of the collected watchings. Repeat watchings and long TV sessions quantify a confidence score. The matrix factorization approach is then used on these confidence scores to predict confidence in attractiveness of other shows.

Multiple problems remain. RMSE is still the benchmark. The original intent of these techniques is to estimate ratings values rather than recommending a list of interesting items. It has been shown by Cremonesi et al. [18] that collaborative filtering centred on reducing RMSE or MAE does not translate into effective top-K techniques. In fact these ported techniques fare worse than non-personalized approaches in some case. Moreover, interest in repositories cannot easily be quantified by a proxy like TV watching. Contributing to a repository is only one way of showing interest but there exist many others that do not entail repetitive actions. Stepping away from this literature reveals more appropriate means of prediction for this context.

2.4.2 Graph-based Approaches

Graph-based approaches constitute an alternative to matrix factorization. These techniques, although also using the user-item matrix, often represent the interests of users for items through graphs and base their recommendations directly on the graph structure. Neighbourhoods can be formed by users that have rated the same items similarly. Alternatively they may be formed by similarity of content; users interested in similar content form neighbourhoods. Oftentimes, items the neighbourhood rates highly are suggested to a neighbourhood member that has not yet been exposed to them.

The original collaborative filtering papers [14, 15] are examples of such approaches. It is however the use of item-item collaborative filtering by Amazon [19] that has clearly popularized the approach. Algorithm 1 of Linden et al. [19] is used offline to precompute the item popularity score and the item-item similarity matrix; the popularity and similarity of each item and each co-purchased item are determined. Similarity is computed using Equation 2.8 and a profile vector of each item.

Algorithm 1 Amazon Item-to-item Collaborative Filtering Pseudo-code

```
def offline_cf_precomputation(items_catalog):
  for item1 in items_catalog:
    for customer in purchasers_of(item1):
      for item2 in purchases_of(customer):
        popularity[item1] += 1
        popularity[item2] += 1
    for item2 in items_catalog:
      item_item_sim_matrix[item1, item2] = similarity(item1, item2)
```

The scalability of this approach is a great advantage. Usually customers have very few purchases making this algorithm $O(NM)$ where N is the number of users and M is the number of items. Using the graph of user-item connections (purchases, usage...) directly rather than building an intricate model has the further advantage of providing a clear explanation for each recommended item.

Zhang et al. [11] propose a serendipitous recommender that moves away from the popular matrix factorization technique to rely on a setup closer to the one above. Their Auralist artist recommender is an intermediary between latent methods and graph-based approaches. The authors collect which individuals listen to which artist, i.e., a simply binary mapping from individuals to music groups. Each artist is associated with a profile vector mapping to their listener constituencies. These listener constituencies distributions are obtained by Latent Dirichlet Allocation (LDA) [20], another way to obtain latent factors. The artists' profile vectors represent the interest of different listener communities in them. The generated basic score, $\mathsf{Basic}(u, i)$, a user u with history of artist preference T_u gives an unknown artist i is:

$$\mathsf{Basic}(u, i) = \sum_{j \in T_u} \mathsf{similarity}_{\cos}(\mathbf{p}_i, \mathbf{p}_j), \tag{2.18}$$

where $\mathbf{p}_i$ and $\mathbf{p}_j$ are the C-elements LDA profile vectors of artists i and j respectively.

$\mathsf{Basic}(u, i)$ is blended with other scoring approaches. For instance $\mathsf{ListenerDiversity}(i)$ uses entropy to measure how spread out an artist is over multiple listener communities.

$$\mathsf{ListenerDiversity}(i) = -\sum_{c=1}^{|C|} p_{i,c} \log_2(p_{i,c}) \tag{2.19}$$

The more spread out is an artist's Dirichlet distribution (i.e. the greater their entropy) the more different listener communities he or she reaches. By recommending artists that reach numerous affiliated communities to an individual, it might open his or her musical affinity to different genres they did not know existed. In practice this approach is offset by a popularity bias.

Another of the scoring approaches introduced in [11] relies on the clustering coefficient used for de-clustering purposes. The clustering coefficient is a network theory metric indicating how interconnected are the neighbours of a graph node. This is done to favour artists that are the most cluster avoiding in the artist-artist graph where two artists are connected if their similarity score is non-zero. The blending is done with respect to the ranking obtained. The ranking produced by each single technique is blended with the others as follows:

$$\mathsf{Hybrid}(u, i) = \sum_{a \in \mathcal{A}} \lambda_a \mathsf{rank}_{a(u,i)}, \tag{2.20}$$

where $\mathcal{A}$ is the set of algorithms to blend together and the coefficients λ_a , with $\sum_{a \in \mathcal{A}} \lambda_a = 1$, determine the weight given to each algorithm. The resulting rank of algorithm a for user u and item i is $\mathsf{rank}_{a(u,i)}$

Zhang et al.'s [11] approaches show the inherent trade-off between accuracy and serendipity, but not between novelty, diversity and serendipity. All three metrics are increased significantly by their node de-clustering approach. It identifies the usefulness of easily interpretable graph approaches to serendipity measurement. Their user surveys conclude that individuals are ready to accommodate a lower accuracy in exchange for discovery opportunities.

A pure graph-based approach is followed in Sugiyama and Kan's [6] scholarly paper serendipitous recommender. Like Zhang et al. [11] and this work finds, the data considered is most easily modelled by a user-item (researcher-paper in this case) network. In addition to the knowledge of which researcher has written which paper, the time (year) of these scholarly contributions is known. A researcher u is modelled through his or her serendipity profile $\mathbf{p}_u$. An unknown paper i is modelled through its feature vector $\mathbf{p}_i$. The score u

would assign to i, is again $\mathsf{similarity}_{\cos}(\mathbf{p}_i, \mathbf{p}_j)$.

What is different about the Sugiyama and Kan's approach is the way these profile and feature vectors are built. They reflect graph properties and take into account time. The basic user profile is:

$$\mathbf{p}_u = \sum_{i \in \mathcal{T}_u} e^{-\gamma \cdot (\mathsf{year_n} - \mathsf{year_i})} \mathbf{p}_i \tag{2.21}$$

where γ is a tunable forgetting factor and n is the index of the most recent publication of u. The paper profiles $\mathbf{p}_i$ are term frequency vectors summed with the weighted term frequency of cited papers and referencing papers by cosine similarity of term frequencies. This basic researcher profile is complemented in two different ways. One profile construction blends the profiles of dissimilar researchers and the other blends the profiles of other authors in the researcher's co-authorship network weighted by their transitive distance to him or her. By building a user profile based on dissimilar users, the cosine similarity returns results that are more dissimilar to a researcher's interest. Complementarily, by building a user profile based on collaboration, the authors attempt to capture the real-life dynamic of colleagues suggesting avenues unthought of to each other.

A co-authorship network is built for each researcher. It is a tree centred on the user and expanding according to his co-authorship and his co-authors' co-authorships and so on. Many heuristics are experimented to assign a weight $w(ca_{pl})$ to co-author ca's profile $\mathbf{p}_{ca}$ at path length pl of researcher u. The different heuristics are: no weight ($w(ca_{pl}) = 1$), reciprocal path length ($w(ca_{pl}) = 1/(pl+k)$, where k is a constant), dissimilarity ($w(ca_{pl}) = 1/(\mathsf{similarity}_{\cos}(\mathbf{p}_u, \mathbf{p}_{ca_{pl}}) + k)$) and product of reciprocal path length and dissimilarity. In effect Sugiyama and Kan find that using the co-authorship network with the product of reciprocal path length and dissimilarity beats using dissimilar users on a similarity-based serendipity measure. What this research highlights though is the flexibility of heuristics that can be used for serendipity once a network is created and how easy it is to interpret the processes.

Note however that in each case the proposed approaches are validated on a random subset of the collected data that is withheld. None of these works attempt to predict in real-time fashion. Time is used only as a weighting parameter.

The above papers and the ones mentioned in the criteria section cover the essential research on serendipitous predictions. Measurement of serendipity is fragmented. There

is a need for a better understanding of its measurement tools. Also nearly inexistent, is prediction on GitHub data. We describe the research that this new data trove has caused in the next section.

2.5 GitHub Research

The openness of GitHub data has recently attracted several studies [21–25] aimed at characterizing this novel collaboration context. On the one hand, Suchal and Návrat [21] take up the task of predicting repositories users would bookmark. On the other hand, [22–25] analyze collected GitHub data in a software engineering context where the goal is to better understand the interactions and particularities stemming from an environment like GitHub which mixes social and collaborative features. Although not providing adapted recommendation techniques, this body of research produces insights into the motivations of GitHub users and helps us qualify, motivate and explain the results we obtain in our own work. We review this body of work next.

2.5.1 Recommending Repositories

Suchal and Návrat [21] propose a top-K nearest neighbour based recommendation system that leverages the full-text search capabilities of modern database systems to predict repositories GitHub users would star —'bookmark' in the site's parlance. This action is detailed in Chapter 3. Each repository is associated to a string built by concatenating the developers interested in the repository. Similarly each developer is associated to a string built by concatenating its repositories of interest.

The authors generate the top-K list by first selecting the top N users sharing the most repositories in common with the selected user. This neighbourhood is then queried for their repositories and the repositories already seen are removed from the resulting list. The first K returned results are chosen to form $\mathcal{R}_u^K$. This neighbourhood approach relies on the inherent full-text search indexing of the database (MySQL or Sphinx) to rank the results with respect to their match to the queries. A pseudo-code version of their algorithm is shown in Algorithm 2.

Algorithm 2 Full-text Collaborative Filtering Pseudo-code

```
#Inputs: user_id, k, neighborhood size N
#Output: top-k recommended items for user user_id
def recommend_with_fulltext(user_id, k, N):
  items = find_items_connected_to(user_id)
  items_query = create_query(items.ids)
  n_similar_users = query_per_user(items_query, N)
  n_similar_users.remove(user_id) # remove current user
  user_query = create_query(n_similar_users.ids)
  # find items not already seen, based on similar users
  similar_items = query_per_item(user_query, k, items)
  return similar_items
```

The dataset used in Suchal and Návrat [21] paper was directly provided by GitHub as part of a contest held in 2010. The dataset represented 440,237 developer-to-project watching links across 56,519 developers and 120,867 projects, i.e. a 6.44×10^{-5} matrix connection density. For $K = 10$, the best ratio of correct guesses to all guesses was 20.0%. This result beat out all other alternatives. It gives us a benchmark, albeit a precision benchmark, to compare against in terms of the accuracy of our methods.

2.5.2 Qualitative Characterization

To the best of our knowledge Suchal and Návrat's work is the only one to tackle the recommendation problem for the rich data provided by GitHub. The other works focus on characterizing GitHub as a collaborative environment.

Dabbish et al. [22] interviewed 24 developers using the github.com site. Hobbyists, professionals working in a software development organization and professionals working in a (mainly) non-software development organization were selected. This sample was further divided into users with greatly watched repositories (at least 80 watchers) and users without. The goal was to assess the value of user visibility and of a transparent collaboration process. It was found that GitHub members induce a level of commitment, intention behind decisions, relevancy to community and personal relevancy by watching the activities

of other members of the site. These inferences in turn fuel project management, reputation management and personal growth as a developer.

Dabbish et al. [22] importantly provide potential motivations for the actions and reactions of individuals interacting on the site. It was remarked that the decision to contribute to a project often depends on the number of prior contributors or watchers. Because the spent effort will have a bigger impact on a largely watched project than on an unknown one, surveyed users said they were more likely to contribute to popular repositories than unpopular ones. More individuals are interested in the project and thus more individuals will reap the benefits of the contributions.

Prior activity around a project is also a good indicator as to the liveliness of the project. Projects with no recent activity are assumed dead. Therefore individuals are less inclined to submit code to them: their submission is less likely to be reviewed at all, and if it is reviewed and accepted, it is less likely to affect a significant user base.

The popularity of a project is also often seen as a sign of quality and thus influential in its adoption. Because developers can also *watch* each other, the same is said about developers with a large following. Those are seen as noteworthy individuals who are more likely to produce work of quality and are thus more relevant to follow in turn. Another influence on watchings is remarked upon: some developers also follow projects and other developers to be informed about opportunities to contribute and to keep up with potentially problematic changes. This motivation relates to the notion of communities which are supported by GitHub. The social features of GitHub provide a way for developers to keep abreast of the changes and discussions surrounding a project and thus fuel the community around it.

Finally, developers take into account the effect of their own actions on other developers. Contributions, watchings or followings are seen as ways to promote the work of others as well as build their own reputation and encourage them to be more mindful of their work. A process of curation emerges from this dynamic. Some users become curators for others using their notoriety to shed light on specific projects. Brian Doll, a GitHub employee running the marketing section, illustrates it best [26]:

> I just got an email two days ago from an ex-coworker that I had not seen in a couple of years. He said, “It’s been great following you on GitHub. You provide the curation of a lot of the content that I’m interested in because we have similar technical backgrounds. [...] The benefit that he was referring to was that by

> watching me on GitHub, he could see that I had starred certain repositories. For example, thousands and thousands of jQuery plugins vary widely in quality, but if you know somebody who is a prominent jQuery developer and they star a bunch of interesting jQuery plugins, those plugins are probably pretty good.

All in all, the open development on GitHub makes interconnections between users and between users and repositories an important deciding factor in further interconnections according to the surveyed users.

2.5.3 Quantitative Characterization

Thung et al. [23] begin to complement this qualitative description of interconnections on GitHub by a quantitative study of randomly picked users and their repositories. They construct two graphs: a developer graph and a repository graph. The developer graph has a node for each developer and an edge between two nodes if the two corresponding developers have contributed to the same repository. The repository graph has a node for each repository and an edge between a pair of nodes if the corresponding repositories share a contributor in common. The strength of the developer-developer links and the repository-repository links are computed by respectively counting the number of repositories each pair of developers have collaborated on and the number of developers in common between each pair of repositories. Using these measurements, the PageRank algorithm [27] is run to suggest the most influential users and repositories in each graph.

One hundred thousand projects are randomly chosen by interfacing with the github.com site and, from the pool of all developers that have contributed to those, 30,000 developers are randomly sampled. The developer-developer degree distribution exhibits strong variation in its tail, whereas the repository-repository graph exhibits a power-law distribution. The average shortest path of the repository graph's largest connected component is 3.7 and the average shortest path of the developer graph's largest connected component is 2.7. These findings are much lower than what has been observed in other networks implying that communities of developers on GitHub are more tightly knit and benefits from a select number of greatly active users.

The dataset used by Thung et al. [23] is composed of randomly sampled data. Thus it only paints a limited picture without fixed boundaries. Moreover it only considers contributions as the source of connection between developers and between repositories. Many

of the other mechanisms GitHub provides for interaction (such as following other users or starring a repository) are completely ignored.

The most recent paper by Lima et al. [25] is the most current and wide-spanning quantitative analysis of the interaction networks on GitHub. The authors look at an 18-month period of activities from March 11, 2012 to September 11, 2013 and construct the networks of developer-developer followings, developer-developer collaborations and developer-project watchings and contributions from the 183.54 million observed activities. The structural properties of these networks is studied.

The constructed graph of followings contain 671,751 user-nodes and 2,027,564 edges for a density of 4.49×10^{-6}. Following ties are at 90.6% non-reciprocal: a user follows another without the other following them back. This confirms the use of developers as curators or as sources of news and not simply as social acquaintances. We add to this observation that this presents GitHub as more of a goal-driven environment than a social environment. Many of the decisions faced by an active user (e.g., starring or contributing to a repository) are subject to a cost-benefit analysis. Details of the network composition of the other graphs are not provided but the degree distribution of each graph is again shown to follow a power-law which has been observed as typical of networks of people [28].

Lima et al. [25] make two other observations of interest in characterizing the user activities. Firstly, a highly active user is not necessarily followed by a large number of individuals. This reinforces the notion that there are projects receiving sustained attention which are otherwise unnoticed by large swaths of the broader developer community. Secondly, and in line with this observation, collaboration among users on projects is rare. Only 74.22% of their final number of repositories have at least two contributors. In other words, more than a quarter of the observed maintained repositories have only a single developer working on them over the 18-month period. This motivates the need for a better discovery process.

The number of missing interactions per repository is unknown in Lima et al.'s [25] study because the considered activities start on March 11, 2012 and the github.com site was launched in 2008. Moreover, the follow actions prior to the starting date of the collected data are also unknown. One must be careful in the usage of GitHub data.

2.5.4 Limits of the GitHub Data

The above research included all observed events in their analysis with only limited filtering. From our experience and according to Kalliamvakou et al.'s recent work [24], considering this raw data as such is problematic. It is important to filter the collected data appropriately. Specifically, Kalliamvakou et al.'s mentioned work outlines nine perils to which researchers mining GitHub data are exposed. We list these perils in Table 2.1.

Peril	Description
I	A repository is not necessarily a project.
II	Most projects have very few commits.
III	Most projects are inactive.
IV	A large portion of repositories is not for software development.
V	Two thirds of projects (71.6% of repositories) are personal.
VI	Only a fraction of projects use pull requests. And of those that use them, their use is very skewed.
VII	If the commits in a pull request are reworked (in response to comments) GitHub records only the commits that are the result of the peer-review, not the original commits.
VIII	Most pull requests appear as non-merged even if they are actually merged.
IX	Many active projects do not conduct all their software development in GitHub.

Table 2.1 Kalliamvakou et al.'s GitHub data mining perils

Because of the accessibility of the web, various users interact with the GitHub site in various ways. This requires keeping only potentially informative events about repositories for which we know the complete history. We provide a detailed description of some of these perils here in preparation for the explanations of our counter-measures.

Peril I stems from the observation that the artifacts of the work on a software code base in a distributed environment are by definition not centralized. A single repository will not capture the entirety of the history of the changes made leading up to what is presently observed. The original repository and its development copies should be considered together when looking at its development as a whole. A "project" in the description of the perils refers to the ensemble of original repository plus clones. We do not follow this dichotomy in the rest of our work. The last peril, Peril IX, drives at this point also: software engineering is a multi-faceted process for which not all facets are present at the same location. Community

and developer-to-developer discussions are sometimes held on mailing lists or IRC chats. Bugs and issues may also be reported on external trackers (e.g., bugzilla.org used by Mozilla in particular). GitHub captures a large and rich part of the whole process—importantly the code itself—but does not have a monopoly on every aspect.

Perils II and III highlight the limited amount of activity per project found on GitHub. Many projects have few contributions and so it is difficult to extract much information from them. As time goes by, many repositories are left abandoned as well. There is a timeliness to the collaborative and interest processes at play on the site.

Because GitHub allows the hosting of raw files on its servers, many of the observed repositories are not necessarily for software development. Peril IV reflects the notion that the site has moved on to include a slightly wider technologically-savvy audience than simply developers.

The hosting flexibility has made it easy for personal projects to be hosted on the site. 90 out of the 240 respondents to Kalliamvakou et al.'s user survey said they hosted and worked on their personal projects with no intention of collaborating directly with others. 71.6% of repositories analyzed have only one participant. This is even more than what was observed in [25].

Perils VI through VIII deal with pull requests a specific artifact of the collaboration model on GitHub. Pull requests are a way for a developer to solicit inclusion of proposed changes into a repository by its owners. However GitHub's logs are inconsistent when it comes to assessing whether such a request has been accepted and merged back. The example given is the logs of pull requests for the popular 'homebrew' project. On the site 13,164 pull requests have been opened and 12,966 have been closed, but the logs for the corresponding period only show 6,947 pull requests closed. 'Closed' here simply means the request has been addressed —it might have been merged or it might not have. According to [25], GitHub does not log merges made outside the website and so a potentially important number of such pull requests might simply not be logged. We do not dwell on the problems related to these logging artifacts as we circumvent the usage of the pull request logs altogether in our work. The rest of these perils must be kept in mind when dealing with raw GitHub data.

2.6 Summary

This chapter summarized background work on the three fronts of research our work investigates: the notion of serendipity, graph-based recommendation approaches and the nature of GitHub.

Serendipity has been considered an emerging property of the criteria of novelty and diversity and, more recently, it has been analyzed as a criteria of its own. Two specific definitions have been proposed: unexpected but useful discovery and discovery that would not have been made otherwise. However, these diverse definitions have not been compared on the same dataset.

Top-K recommendation methods stemming from collaborative filtering were organized into latent factors approaches and graph-based approaches. We highlighted how other researchers have used graph distances and diverse similarity heuristics to provide recommendations they deemed serendipitous. Modelling predictions through a graph opens up flexible heuristics and interpretative power.

GitHub user surveys have identified the motivations behind following users and repositories. Prior level of activity, popularity of the contributors, popularity of the repositories indicate work of interest, quality, and attractiveness to the wider community. A self-fulfilling prophecy dynamic can occur where individuals work on a project thinking that others will too. Only statistical data characterizations have been done to complement these surveys of agent satisfaction. Usage of these insights for predictive purposes have been largely unconsidered.

In the next chapter, we methodically describe the nature, acquisition processing and statistical properties of our GitHub dataset. We address along the way the mentioned perils of mining GitHub for data.

Chapter 3

The Dataset

One of the important contributions of our work is a novel dataset of GitHub activities. In this chapter we first describe the GitHub site and the type of data of interest to our study. Then the acquisition and the processing of this dataset to obtain timestamped user-user and user-repository networks are presented. A summary of the salient statistical features of the generated networks follows. Because of the novelty of the data, we spend some time justifying the different filtering processes used.

3.1 Github.com

3.1.1 The Website

The site github.com offers free code repository hosting for public projects and paid code repository hosting for private projects. These projects have traditionally been software source code, but recently other types of repositories have appeared: data repositories (e.g. the Code of Laws of the United States of America[1]), various books[2], websites[3] and others. We will use the terms "project" and "repository" interchangeably to denote these online collaborative spaces. Kalliamvakou et al. [24] distinguishes between repositories —associated with a single URL— and projects —groups of repositories that are variations (forks) of each other. As we focus on the process of discovery, discovering a fork or a normal repository, as long as it is of potential interest, is not a distinction worth making. Sometimes

[1] https://github.com/divegeek/uscode
[2] https://github.com/showcases/writing
[3] http://jekyllrb.com/

recommending forks of some repositories is the appropriate action to do.

Public projects are made available to review, use, copy and contribute to by any registered user of the website free of charge. In addition to this collaborative infrastructure, social features are present on github.com. Comments and discussions can be had around pieces of code. Appreciation or interest is made known through the action of *starring* repositories —a form of public bookmarking. Individuals can follow one another much like users of Twitter can in order to stay aware of each other's activities. Following another user on GitHub provides notifications of the followed individual's starrings, repository creations and other relevant actions concerning that individual such as addition to a team. Starring a repository also provides similar notifications, but pertaining to the repository itself: creation of issues, fixes and contributions as well as comments. The site also allows for the creation of organizations which are entities grouping different users together.

Git[4] is the underlying technology behind github.com's infrastructure for collaboration. Git is a distributed version control software that facilitates the exchange and merging of atomic changes known as *commits* between repositories. These commits are *pushed* to a repository on GitHub and can be retrieved by the owner of another repository by *pulling* the commits into his or her own repository. This effectively pushes the changes to the other project hosted on the GitHub platform. Contributions on GitHub can come from email patches, *pull requests* —a suggestion to merge changes from one repository into another— or direct contributions. By analyzing the commits found in the push events, we piece together the contributors of each repository. We explain in some detail how the data was acquired and processed in the next section.

The github.com website leverages Git to facilitate open and accessible online collaboration between individuals. As is exemplified by GitHub's tagline "Social Coding", interpersonal interactions and individual contributions are strongly promoted [26]. Members have a profile page showcasing their own repositories, the repositories they contributed to and their recent activities. A project's web address places the name of the owner first: *github.com/<user name>/<project name>*, and pictures of contributors are associated with any page containing the files to which they contributed. All of these aspects conduce to putting members and their work front and centre.

GitHub's popularity —7.1 million users and more than 16 million hosted projects as

[4] http://git-scm.com/

of October 2014[5]— make it an appealing target for data analysis [22, 23, 25]. GitHub is growing: there were 6.5 million users and 14.3 million repositories in August 2014. Most interestingly for our research, this popularity has made it difficult for users to make serendipitous discoveries as only a fraction of this vast number of repositories is featured prominently on the site.

Two methods, the 'Showcases' and the 'Trending' pages,[6] surface a sliver of the projects for potential discovery. The 'Showcases' page is a set of curated lists[7] of projects grouped by topics e.g., 'Science', 'Emoji', 'Projects that power GitHub'. These lists are manually generated and not exhaustive; they rely on the second method of discovery: trending repositories. The Trending page enumerates the projects that have been the target of recent and frequent user interests. This approach is effectively a time dependent top list. For both discovery methods, the same repositories are exposed to all users and popularity is the main deciding factor. When logged in, the Trending page will also display the recently starred repositories of the individuals one follows. The window of time under consideration can be shortened to the previous day or extended to the previous month with the last week as the in-between option.

All of the above, mixed with GitHub's openness with its data, makes it a ripe target to test the serendipitous nature of automated recommendations. We look at the events occurring on the website to provide our data.

3.1.2 Events

We reconstruct the relationships between users and between users and repositories from the activity feed made available by GitHub. All public events on the site (following another user, starring a repository, contributing to a repository...) are timestamped and logged in Javascript Object Notation (JSON) format [29]. The most recent events are made available through GitHub's application programming interface[8] (API). This API also exposes information on a per-user (user name, followers, repositories...) or per-repository (owner, name, README, main language...) basis.

As of the time of writing of this thesis, there are 19 different visible event types[9] in the

[5] https://github.com/about/press
[6] https://github.com/showcases and https://github.com/trending respectively
[7] https://github.com/blog/1802-showcasing-interesting-projects-in-explore
[8] https://developer.github.com/
[9] https://developer.github.com/v3/activity/events/types/

feed. Table 3.1 contains the subset of six events that are retained for our purposes and the meaning of these events in our context.

The 13 other events were discarded for various reasons. Two of these events are related to other services provided by GitHub (GistEvent and GollumEvent). Three events were better captured by another type of event: the ForkApplyEvent is subsumed by the ForkEvent, and the PullRequestEvent and PullRequestReviewCommentEvent are better captured by PushEvents because the latter confirms acceptance of the contributions. The rest were out of scope for the prediction task at hand (TeamAddEvent, DeleteEvent, MemberEvent,...).

Event Name	Description
CreateEvent	a new repository is created
FollowEvent	a user follows another user
ForkEvent	an independent copy of a repository is made
PublicEvent	a private repository is made public (open-sourced)
PushEvent	at least one Git commit is merged in a repository
WatchEvent	a user *stars* a repository

Table 3.1 Retained GitHub activity events

A CreateEvent is generated in the activity feed whenever a new project is hosted on github.com. This does not include forks or open-sourced projects. Both normal users and organizations can create a repository.

A FollowEvent is logged in the activity feed when a user decides to follow another one. From then on, updates about the followed user will be transmitted to the follower via his or her notification page.

A ForkEvent records the cloning of an existing project for the purpose of contributing back at a later point in time or to initiate an alternative implementation. These events along with PublicEvents are the only other indicators that a new repository has been created.

PushEvents refer to the action of transmitting a series of commits (at least one) to a hosted project. Because Git is a distributed version control system, any permanent update to the project will eventually be pushed to the hosted repository. This makes it possible for us to track all collaborations occurring on the project.

Finally, WatchEvents correspond to a user starring a repository. The notification semantics we explained for FollowEvents and in our original description of the site

applies to starred repositories because this is what starring meant between February 2011 and February 2012 —our period of study. In August 2012, the semantic of starring changed to simply bookmarking with no notification. The name WatchEvent has been kept despite the change and the new notion of watching (to get notifications) does not have a distinct API event.

It is also important to note that only additive events are made available by GitHub. Repository creation, project starring and user following are tracked and made public while repository destruction (or renaming), project unstarring and user unfollowing are not recorded. In the cases where a repository is starred multiple times by the same user (which implies it was unstarred in-between the two distinct events), we have kept the earliest starring time. The same approach was taken for FollowEvents; only the first FollowEvent was kept.

3.2 Acquisition and Processing

We explore the possibility of a personalized recommendation system by the capture of social relationships between users via their 'followership' connections and the capture of users' interests via their contributions, stars and forks of repositories. We describe here how the data was collected and processed.

3.2.1 Acquisition

Because only the most recent activities are available through the GitHub API and because only 5000 requests per hour can be made to the API by a single user, the GitHub Archive site[10] was used to download past compressed archives of hourly activities.

The GitHub Archive stores all events emitted by the GitHub public activity feed since February 12th 2011. This data is made available raw in the form of hourly compressed archives of concatenated JSON strings summarizing the events.

We used archives from February 12th 2011 starting at 00:00 UTC to February 11th 2012 at 23:59 UTC. A one year span was chosen as it is wide enough to provide a representative amount of data, while not too wide as to be computationally prohibitive to process. This selection also addresses Peril III of Table 2.1: "most projects are inactive". All of the studied

[10] http://www.githubarchive.org/

repositories have been active within the year by definition. However, it is also important to note that a serendipitous recommender would help counteract inactivity. Analyzing its performance in a realistic setting where some repositories are less active than others (some repositories have received the attention of a single user over the year for instance) paints a more accurate picture of the reality in any case.

With GitHub being founded in February 2008, the collected data necessarily lacks information from events prior to the earliest retrieved archive. The earliest available archives were therefore chosen so as to cover as many influencing GitHub events as possible and to thus limit the influence of those GitHub events that are unobservable. This has the limitation of collecting repositories that might now be deleted on the GitHub website. Content properties cannot be collected for all repositories and their usage is therefore ignored.

Only repository-related events pertaining to projects for which we have observed a CreateEvent, a ForkEvent or a PublicEvent are kept. Moreover, ForkEvents duplicating a repository for which we have not observed the original CreateEvent or PublicEvent are filtered out. For ForkEvents that target other forks, the creation of the original root repositories of the forking hierarchy must have been seen. Only user-related events for users that interacted with those new projects were considered. This way, all the contributors, starrers and forkers of each considered repository are completely known and the full history of each repository is known as well. In doing so we eliminate cases where unobservable users might have played a significant influential role in the amount of interest devoted to repositories for which we do not have the full history.

Thus, a total of 26,064,630 events were assessed over the one year period and 18,557,838 of them fell in one of the six types outlined in Table 3.1 once malformed observed events were discarded.

3.2.2 Processing

Additional processing to further select valid users and repositories from the raw events was done to produce analyzable networks with enough data points. We process the data in order to predict using networks constructed with sound empirical evidence.

In Table 3.1, a PushEvent is described as potentially containing multiple Git commits. These different commits can be from different authors. To determine the original author of each contribution across the different contribution approaches, we kept track of the

author of each individual commit and the commit's numerical hash. Git associates a 160-bit numerical hash to each commit. This hash is generated using the Secure Hashing Algorithm Version 1 (SHA1).[11] By associating to such a hash the first GitHub user to have contributed it in a PushEvent, we reconstruct the contributions of users to repositories. The SHA1 hash makes it extremely unlikely for an attribution collision to occur among the retained commits in the observed year span. By counting contributions this way we side-step the problems associated with pull requests entirely and thus eliminate Perils VI, VII and VIII of Table 2.1. Only accepted pull requests are taken into account because they will have resulted in a PushEvent containing the pull request's original commits.

Users who have shown interest in at least three projects not owned by themselves through contributing, starring or forking such projects, and who follow or are followed by another user satisfying these interest conditions are kept. This selection process keeps users with a minimum number of external interest examples and ensures that social (following) links are potentially analyzable. The resulting set of repositories these users are interested in is kept except for some of the repositories that are forks.

A common collaboration pattern on GitHub is to fork a project, push changes to the fork, submit a pull request to the original maintainer of the project and have the proposed changes merged back inside the main project (or not). These forks can also merge some of the changes made to the main repository back to them, therefore potentially incorrectly assigning interest on a user's behalf to a fork when the user was only interested in the main upstream repository. Indeed, not all forks should be treated as projects to be recommended since some are simply the byproduct of collaboration on GitHub. To handle such cases, among the repositories that are forks, only those that have been starred are kept. All repositories that are artifacts of the collaborative process are discarded this way and only those forks that have been manually starred by an individual are incorporated in the final set of repositories. These forks were interesting enough to warrant their explicit starring which means these are instances it would make sense to recommend.

3.3 Considered Graphs

The collected and processed data are used to create multiple time-weighted graphs which we use to run our graph-based recommendation algorithms.

[11] http://tools.ietf.org/html/rfc3174

The first of these graphs is the interest bipartite graph, G_I. Nodes in the left partition are users $\mathcal{U}$ and nodes in the right partition are repositories $\mathcal{I}$. There exists an edge (u, r) with $u \in \mathcal{U}$ and $r \in \mathcal{I}$ if user u has shown interest toward repository r in the form of a starring occurrence, a contribution or a forking event. The weight t assigned to edge (u, r) is the earliest time at which the interest artifact was created. If multiple interest events are recorded between u and r, only the earliest interest event's time is kept. We denote the state of the graph immediately prior to time t as $G_I(t)$. Figure 3.1 illustrates an example interest graph.

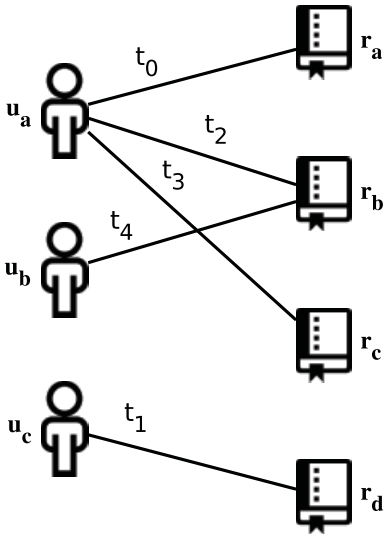


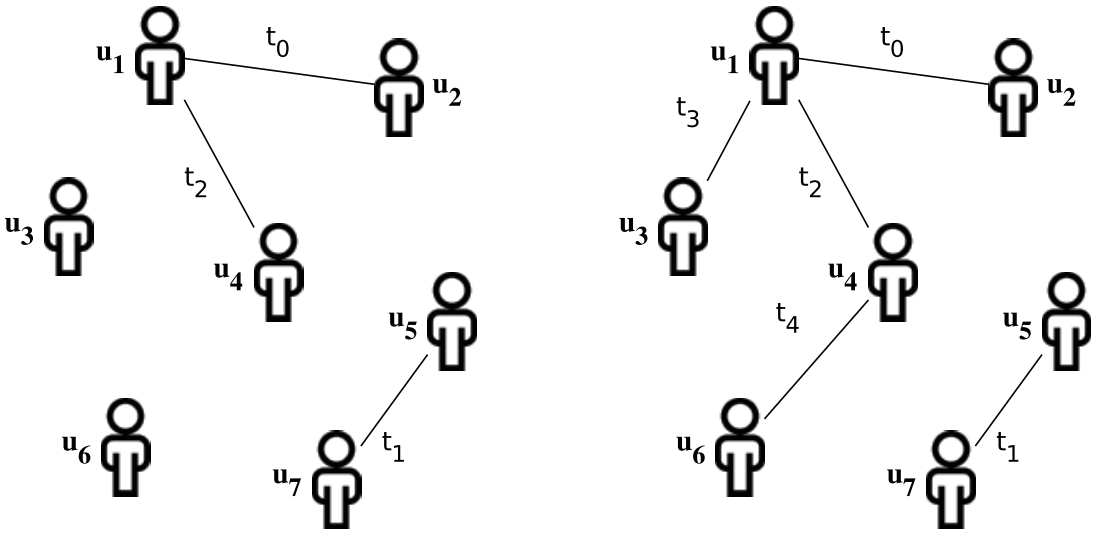
Fig. 3.1 Interest graph G_I: users are on the left and repositories are on the right. User-repository interest links are labelled with the timestamp of the associated earliest interest event

The second graph is the interest-neighbour graph G_N which is not constructed explicitly but rather implicitly. The nodes of G_N correspond to the users and its edges indicate co-interest between two users. There is an edge (u, v) with $u, v \in \mathcal{U}$ if there is a node r in G_I such that the edges (u, r) and (v, r) exist in G_I. The weight t assigned to edge (u, v) is $\max(t_{(u,r)}, t_{(v,r)})$ where $t_{(u,r)}$ is the weight assigned to edge (u, r) and $t_{(v,r)}$ the one assigned to edge (v, r) in G_I. The state of G_N immediately prior to time t is denoted $G_N(t)$. For modelling purposes it is clearer to have it be a separate graph although it can be entirely reproduced from G_I.

The last considered graph is the followership graph or social graph, G_F. We construct it by considering the FollowEvents. Its nodes are users and there is an edge (u, v) with

$u, v \in \mathcal{U}$ if u or v follows v or u respectively. The edges in G_F are undirected even though a follower-followee relationship is not. This was chosen to be so because we consider a FollowEvent to indicate a commonality between two users. If a user follows another one, it must be because they share similar interests irrespective of whether one knows the other or not. The activities of one are attractive to the other. The weight t assigned to edge (u, v) is the timestamp of the FollowEvent or the minimum of the FollowEvents relating users u and v together. Again the state of G_F immediately prior to time t is denoted $G_F(t)$.

Figure 3.2 shows an example user-user graph G_U (in practice G_N or G_F) at two progressive points in time analogous to the observed dynamics.



(a) $G_U(t_3)$ (b) $G_U(t_5)$

Fig. 3.2 G_U at time t_3 and subsequent time t_5

3.4 Statistics

We summarize here the characteristics of the collected data and constructed graphs.

There are 46,423 users and 180,488 repositories satisfying the conditions outlined in section 3.2.2. Figure 3.3 shows the indegree, outdegree and degree distribution of the social (follower-followed) connections. The degree distribution of G_F corresponds to the degree distribution shown in 3.3. There are 156,280 links in G_F for a graph density of

1.45×10^{-4}.

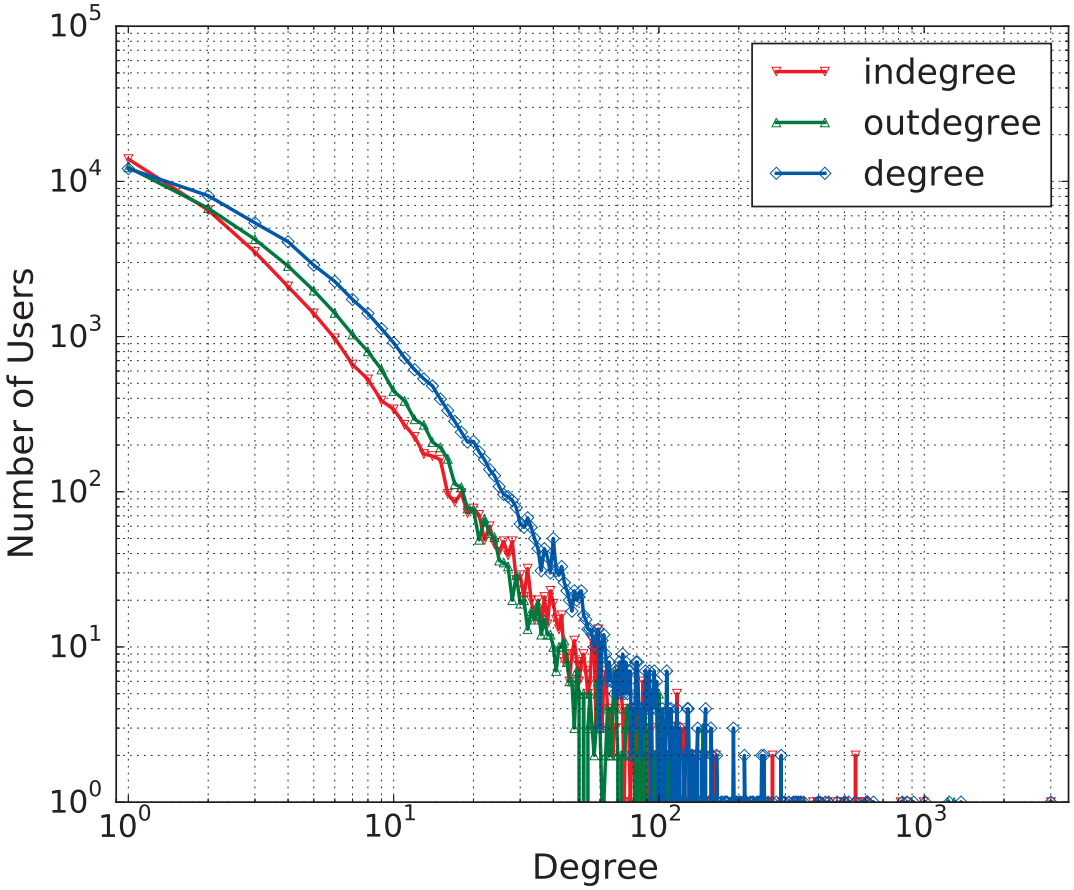


Fig. 3.3 In, out and degree distribution of social connections

The median user degree is 3 (comparable to the average degree of 3.019 observed by Lima et al. [25] for a different and longer period), the tenth percentile is 1 and the 90th percentile is 13. Only 75.6% of the considered users follow at least one other user. In fact 29.3% of users only follow and are not followed, 24.4% are only followed and do not follow and 46.3% follow and are followed. The reciprocity of followership links is of only 8.06%. Reciprocal links are associated with friendship, work relations and mutual admiration.

We conclude from this that the number of explicit followership-social links is limited among active users of GitHub. Lima et al. [25] propose the explanation that the associated notifications are disruptive. They crowd the notifications page with less relevant

information and so following other users has a high cost which elevates the value of the actually followed user. We add that alternative means of following an individual exist such as following them on Twitter, checking their blogs or manually observing what they have recently done on github.com. We hypothesize that the existing social links must carry some influential weight despite their low numbers.

Figure 3.4 shows the distribution of repositories per number of users interested in them (interests). The interest graph, G_I, contains 680,228 edges and its connection density is 8.12×10^{-5} which is comparable to the 6.44×10^{-5} connection density of Suchal and Návrat [21]. The distribution follows a clear powerlaw here. Notably, two-thirds of repositories (66.6%) have only one user interested in them. Kalliamvakou et al. [24] observed that 71.6% of repositories have only one contributor. Moreover, as was observed in the contributors, collaborators and stargazers distributions obtained by Lima et al. [25], there are only very few popular repositories relatively speaking. We observe that only 710 repositories break the 100 interest links threshold. This is 0.3% of all observed repositories. The top 1% of repositories in terms of popularity accounts for 42.4% of all interests. Most of the attention is funnelled on relatively few repositories considering that these repositories cover a variety of programming languages.

Figure 3.5 shows the converse of Figure 3.4, that is, the distribution of users per number of different total repositories they are interested in (per their number of interests). The median interest span of users is 9 which confirms GitHub's ability to involve or simply attract its members to various hosted projects. Indeed the top 1% of involved users only accounts for 10.7% of all observed interests. This decent quantity of interests per user allows us to attempt recommendation methods on a sound training and validating dataset.

Comparing the number of social connections (followers and 'followeds') versus the number of interested repositories for each user does not reveal any apparent correlation. Figure 3.6 illustrates the number of interests versus the number of total social degree over a magnified region. A user with many social connections is not more or less likely to show interest in more or less repositories than a user with few social connections. Most users tend to have a limited number of interests and a limited number of social connections: 75.1% of users have at most 20 interests and at most 10 social interests.

We note that despite the availability of project READMEs (a file describing the project) through the GitHub API, this information was not used in our work. The main reason for this is that the selected repositories obtained from the period under review are not

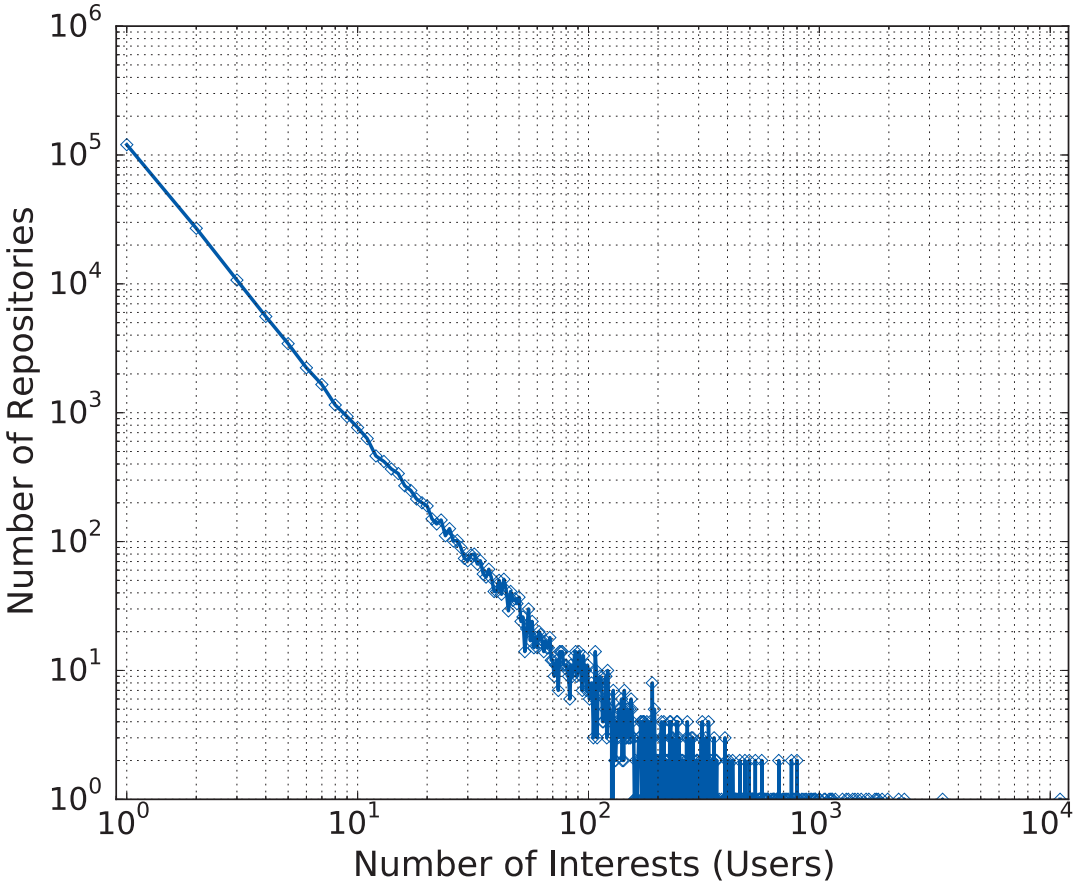


Fig. 3.4 Degree distribution of repositories of G_I

available anymore. In fact 92.2% of the collected repositories have no READMEs that can be retrieved at this point. This is either because the repository has been deleted or renamed since or because the repository had no README to start with. The programming language used can also not be retrieved for 63.4% of the captured repositories. Choosing a more recent period would leave us without the accumulated social links and interests; repositories from this period might additionally still suffer from lack of README and language. We leave the non-trivial work of investigating a temporally wider period for future works.

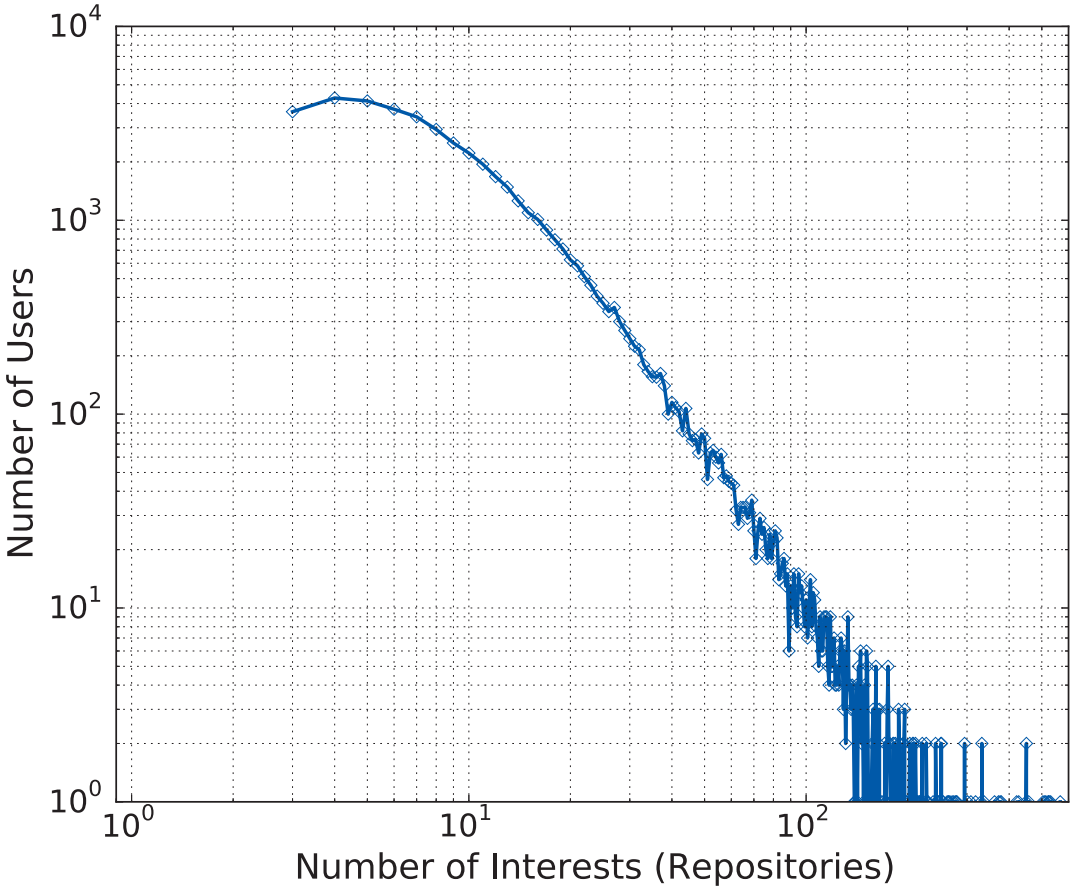


Fig. 3.5 Degree distribution of users of G_I

3.5 Summary

In this chapter we have outlined the workings of the collaborative website github.com. It currently highlights trending repositories and curates topic-focused repositories as its discovery medium.

We collect archived activity events related to project creation, user interaction with those projects and user-user interactions. FollowEvents, WatchEvents, ForkEvents and PushEvents are used to build a social graph, an interest-neighbour graph and a bipartite interest graph, all with fine grained temporal information. GitHub is a rich and dynamic environment where social links and interests are evolving through time.

We have restricted ourselves to users with at least three external interests as our

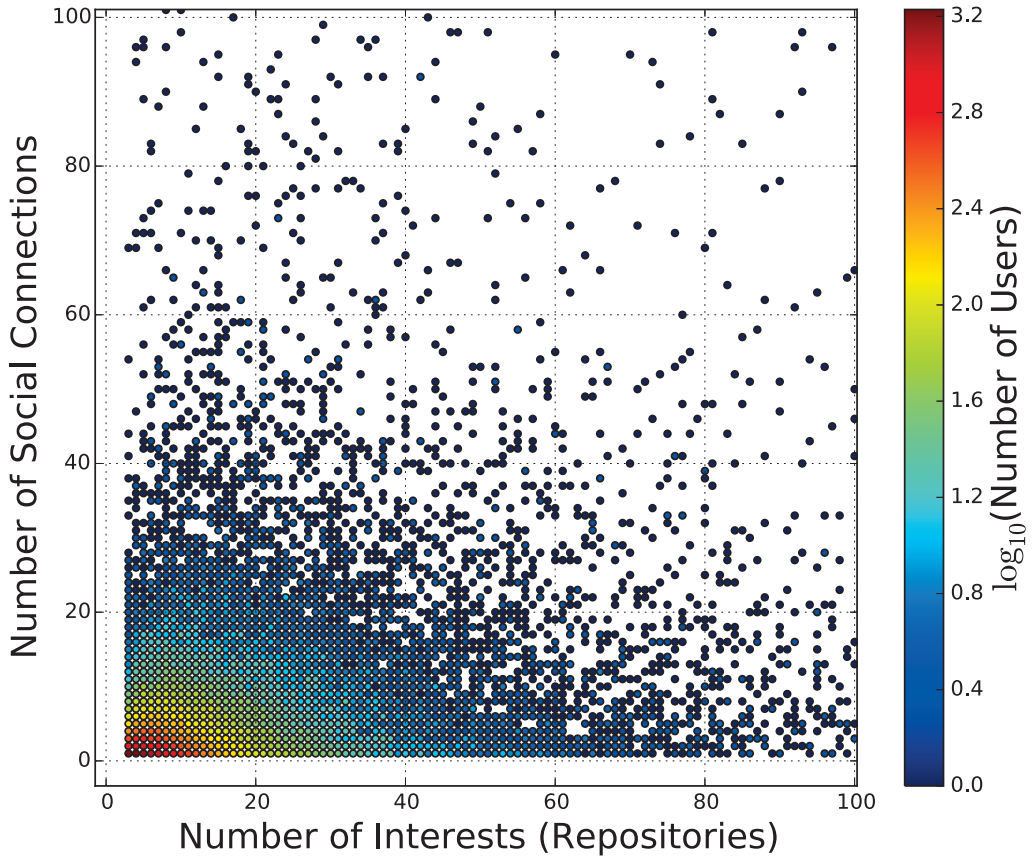


Fig. 3.6 Number of users of G_I per number of interests and social connections (social degree)

goal is to assess recommendation of external repositories with enough empirical evidence. Repository-wise, only those that have been created during the one year span of our collected data have been kept so as to reduce the amount of unobservable influence and thus give richer interpretive power to the graph-based method we will attempt. Collaboration through pull requests is tracked via PushEvents and the potential paucity of information per user is countered by selecting users with a minimum history of activity on the website. We provide in the appendix the location of graphs G_I and G_F for download (G_N is reproducible from G_I).

In the next chapter we present the different graph-based recommendation techniques

whose serendipitous nature is assessed. The techniques presented allow us to propose a more dynamic evaluation of this dataset that goes beyond the static observations made so far.

Chapter 4

Methods

We present here the graph-based recommendation techniques we compare. Specifically, we consider a traditional similarity-based collaborative filterer and a popularity recommender filtered by similarity. Both of these are adapted from [19] to a real-time setting. We then introduce the usage of temporal link prediction techniques for the top-K task. Some of these techniques are based on the interest links of users, and others on their social links. Finally we consider a one-step Markov chain model to explore if paths of interests are formed over time. All of these techniques are benchmarked against recommending time-windowed and un-windowed most-popular repositories.

The goal of these techniques is two-fold. First, and more directly, it is to generate project recommendations for the members of the GitHub site. These recommendations have to reflect some of the actual recorded interests of the considered users and also show signs of serendipity. Second, as a consequence of the results of these recommendations, the other goal is to assess and characterize the ecosystem of user-project and user-user interactions on GitHub. By considering schemes which either use only user-project connections or only user-user connections, and by also considering schemes which combine both types of connections, we highlight which signals are most useful for making recommendations.

The first section provides an overview of the prediction task and the notation used. The subsequent sections group and explain the prediction approaches.

4.1 Overview

Each of the detailed approaches described below assigns a score $s_{u,r}(t)$ by user $u \in \mathcal{U}$ to repository $r \in \mathcal{I}$ at time t where $\mathcal{U}$ and $\mathcal{I}$ are respectively the set of all valid users and valid repositories as described in Chapter 3. The time t is the point in time at which u's top-K recommended repositories, $\mathcal{R}_u^K(t)$, are generated. We summarize the notation used throughout our description of the approaches in Table 4.1.

Symbol	Definition
$\mathcal{N}_{item}(u, G_I(t))$	set of repositories user u is interested in prior to time t
$\mathcal{N}_{user}(r, G_I(t))$	set of users interested in repository r prior to time t
$\mathcal{N}_{user}(u, G_N(t))$	set of users that have shared at least one interest with user u prior to time t (interest-neighbours of u)
$\mathcal{N}_{user}(u, G_F(t))$	set of users that follow u or are followed by u prior to time t (social neighbours of u)

Table 4.1 Symbol definitions

Because all the predictions are generated in pseudo real-time causal manner, a static training set is not used. Instead, a validation threshold time, $t_{threshold,u}$ is chosen for each user from which onwards external interests are predicted. For a user u, $t_{threshold,u}$, is determined by first ordering chronologically the external interest times of user u, then splitting the times into the first 2/3 and the last 1/3, and finally selecting the first time of the last 1/3 of external interest times to be $t_{threshold,u}$. This way we make sure to begin predictions at a time where we have a minimum of two external interests for each user (as per 3.2.2, each user has at least three external interests) and as many internal interests as that user has before $t_{threshold,u}$. As time advances, both internal interests and preceding external interests (validation ones) are taken into account for the next predictions. Only external interests are predicted however. Let $\mathcal{V}_u$ denote the validation set of external interests associated with user u.

GitHub organizations can create repositories. This means that the first signal of interest toward a repository might not be coming from its owner if the owner is an organization. Because the graph techniques used rely on having at least one prior interest to a repository for that repository to be part of the graph and recommended, the instances of first interest toward a repository owned by an organization cannot be predicted.

By relying on a real-time approach rather than a standard training and validating sets

separation, we mitigate further degradation of the number of external interests that could be predicted. A real-time approach allows the algorithms to predict correctly at most 87.6% of validating interests. A standard training/validating split based on time would have only allowed the prediction of 41.7% of interests because many repositories would not have been created during the period covered by the training set.

For a real website constantly serving web pages and under response time limits, when to update the underlying similarities or connection graphs appropriately given their time cost is an open problem. Each hour, each day or each week might be more suitable than trying to compute all intermediate results (similarity scores, connection graphs) every second. A trade-off between an update's cost and its benefit has to be made. For our purposes, we generate K recommendations for each user immediately prior to $t_{vr} \in \mathcal{V}_u^t$ where $\mathcal{V}_u^t$ is the set of timestamps corresponding to each validation repository of u. These recommendations only take into account information available prior to t_{vr}.

Such an online setting is not often considered in the literature due to its computational cost. As discussed in the previous chapter, most validation techniques rely on random training and validating separation rather than chronological separation. Chronological separation has the advantage of representing the reality of how these networks grow over time and thus takes into account the time-dependent nature of discovery. Using future interests to predict past interest has no direct commercial application. To the best of our knowledge our work is the only one to attempt this realistic online prediction setting for the GitHub data.

After generating the scores for each user u, repository r and associated time t, $\mathcal{R}_u^K(t)$ is generated by picking the top K repositories with the highest scores for each user at each validation time. Score ties are broken randomly.

We next group the recommendation approaches by which we attempt to characterize the discovery process.

4.2 Popularity

Popularity approaches are the most common ones in online settings: iTunes top charts and all bestseller lists follow this approach for instance. In particular, popularity is currently used on github.com (Chapter 3, 3.1.1 The Website) with a time-dependent twist.

4.2.1 Most Popular

In order to compare how well our proposed approaches are doing with respect to the current online standard of generating a global most popular list, we generate recommendations based solely on total popularity. The subsequently considered recommendation alternatives should provide better recall and serendipity scores than this benchmark technique.

The "Most Popular" approach (POP) scores repositories as follows:

$$s_{u,r}(t) = |\mathcal{N}_{user}(r, G_I(t))| \tag{4.1}$$

Each repository is assigned a score equal to the number of users interested in it just prior to time t. Sources of interest are contributions, stars and forks as described in Chapter 3. Interests since the beginning of the collected period are taken into account and have equal weight. Although POP is indexed on time, the importance of older interests is not exponentially decayed as others have done [6, 16].

Table 4.2 shows the top 10 most popular repositories after 1 month, 6 months and 12 months. A repository is presented in the format <owner>/<repository name>.

Rank	March 11 2011	August 11 2011	February 11 2012
1	github/pycon2011	harvesthq/chosen	twitter/bootstrap
2	postrank-labs/goliath	defunkt/jquery-pjax	harvesthq/chosen
3	LearnBoost/cluster	altercation/solarized	torvalds/linux
4	defunkt/dotjs	37signals/pow	defunkt/jquery-pjax
5	wesabe/ssu	twitter/twui	zurb/foundation
6	rstudio/rstudio	devstructure/blueprint	fgnass/spin.js
7	cldwalker/one9	jeff-optimizely/Guiders.js	nathanmarz/storm
8	rtomayko/posix-spawn	postrank-labs/goliath	addyosmani/Backbone-Fundamentals
9	nathanmarz/elephantdb	ded/Ender	xdissent/ievms
10	couchbaselabs/iOS-Couchbase	cloudfoundry/vcap	github/hubot

Table 4.2 Top 10 most popular repositories over time

Owners of popular projects like defunkt, nathanmarz or github reappear over time for different projects.

4.2.2 Trending

The trending approach (TPOP) forgets older interests and mimics closely GitHub's own Trending Page which lists repositories that are receiving the most attention in the last day, week or month. Scores are computed by only considering a past interval of time rather than

all information observed so far. The time window captures momentary spikes of attention, but only repeatedly recommends repositories that are continuously popular over time.

Equation 4.2 lists the scoring procedure of TPOP with τ the window of time and $G_I(t-\tau, t)$ the bipartite graph of interests from $t-\tau$ up to and excluding t:

$$s_{u,r}(t) = |\mathcal{N}_{item}(r, G_I(t-\tau, t))|. \tag{4.2}$$

We assign τ the value of a month, a week or a day and denote the respective approaches TPOPM, TPOPW, TPOPD.

Table 4.3 lists the top 10 most popular repositories in the one month, one week and one day interval of time before February 11 2012.

Rank	1 month prior	1 week prior	1 day prior
1	twitter/bootstrap	tommoor/tinycon	nytd/ice
2	tbranyen/backbone-boilerplate	twitter/bootstrap	twitter/bootstrap
3	tommoor/tinycon	blasten/turn.js	Zevas/jQuery-Scroll-Path
4	shichuan/JavaScript-Patterns	visionmedia/uikit	tommoor/tinycon
5	ubuwaits/beautiful-web-type	jairajs89/Touchy.js	jairajs89/Touchy.js
6	Lokaltog/vim-powerline	simplebits/Pears	needim/noty
7	enyojs/enyo	d5/node.native	elabs/serenade.js
8	amoffat/pbs	square/SocketRocket	darcyclarke/Front-end-...-Questions
9	23/resumable.js	chjj/tty.js	steipete/PSYouTubeExtractor
10	visionmedia/uikit	mperham/sidekiq	milohuang/Reverie

Table 4.3 Top 10 most popular repositories prior to February 11 2012

This time-windowing approach has also been tested for all of the techniques below.

4.3 Similarity

Item similarity is the staple of recommendation systems. It is used on amazon.com since at least 2003 [19] to produce high-quality recommendations. Our implementations instill some potentially serendipity-enhancing components.

4.3.1 Most Similar

The similarity approach (SIM) follows the traditional item-based collaborative filtering approach used at amazon.com [19] whose main offline computation is showcased in Algorithm

1. Each repository r is associated to a vector profile $\mathbf{v}^r(t)$ at time t of size equal to the number of valid users such that:

$$v_u^r(t) = \begin{cases} 1 & \text{if } (u,r) \in G_I(t) \\ 0 & \text{otherwise.} \end{cases} \tag{4.3}$$

Users are associated an index in the profile vector. To compute the similarity between two repositories, SIM uses the cosine similarity Equation 2.8. The score for each repository is as follows:

$$s_{u,r}(t) = \max_{p \in \mathcal{N}_{item}(u, G_I(t))} \mathsf{similarity}_{\cos}(\mathbf{v}^p(t), \mathbf{v}^r(t)). \tag{4.4}$$

As it was remarked in the original amazon.com paper [19], similarity is only non-zero for two repositories that share at least one interested user in common. We define $\mathcal{NR}_{item}(u,t)$, the set of repositories which users that have shared at least one interest with user u in the past have also shown interest in prior to time t:

$$\mathcal{NR}_{item}(u,t) = \left(\bigcup_{v \in \mathcal{N}_{item}(u, G_N(t))} \mathcal{N}_{user}(v, G_I(t)) \right) \setminus \mathcal{N}_{item}(u, G_I(t)). \tag{4.5}$$

Computing similarity for these brings the complexity to $O(MN)$.

The SIM approach favours a repository that has many interested site members in common with one of the repository the user has already been interested in. Allowing only one repository in the user history to be very similar to a compared one favours serendipity we hypothesize. A past niche interest can promote a related niche repository to have a higher score than repositories that are similar to the average of a user's history but not quite as close to a specific repository.

4.3.2 Most Popular among Similar

This most popular among similar approach (POPSIM) is our implementation and adaptation of another suggested technique by Linden et al. [19]. The generated recommendations are the result of first filtering the repositories based on their similarity with repositories the user is connected to in the interest graph G_I and of then recommending the overall most popular ones among these.

Since the similarity used is based on interest, we select, for the first step, only repositories that share at least one edge with an interest-neighbour of the considered user; we have defined this set of neighbouring repositories, $\mathcal{NR}_{item}(u,t)$, above in Equation 4.5.

Only the popularity of such kept repositories is computed and used as the score:

$$s_{u,r}(t) = \begin{cases} |\mathcal{N}_{user}(r, G_I(t))| & \text{if } r \in \mathcal{NR}_{item}(u,t) \\ 0 & \text{otherwise.} \end{cases} \tag{4.6}$$

This simple heuristic filters out repositories that might be popular, but that are likely to be unrelated to the sphere of interest of the user. It then hones in on those related repositories that are overall most popular.

4.4 Link Prediction

Link prediction approaches pose the prediction problem as one of estimating which edge will be created in the bipartite interest graph at time t given the structure of the graph prior to t. We consider approaches based solely on the interest network G_I and then approaches taking into account the social network G_F as well.

4.4.1 Interest-based Link Prediction

Interest-based link prediction is a category of approaches stemming for the field of network science which is particularly suited to our knowledge of the evolving structure of G_I. We do not have information about content and thus look to methods with proven track records that only need knowledge of the graph structure. The three recommendation heuristics considered are: Common Neighbours [30], Adar-Adamic Index [31], and Resource Allocation Index [32]. We have modified these approaches to embed themselves in a real-time setting and to operate on a bipartite interest network. They were originally considered on co-authorship networks where every node represents the same entity. Changes were made to take into account the heterogeneity of nodes in our interest network.

4.4.1.1 Common Neighbours

The interest-based Common Neighbours approach (ICN) scores repositories $\mathcal{NR}_{item}(u,t)$ (Equation 4.5), for a given user u at time t, by assigning them their local popularity count

instead of their overall popularity count as in POPSIM. The neighbours of u interested in repository r at time t are defined as such:

$$\mathcal{IN}(u, r, t) = \mathcal{N}_{user}(r, G_I(t)) \bigcap \mathcal{N}_{user}(u, G_N(t)). \tag{4.7}$$

The score is the number of neighbouring users interested in the repository, i.e., the cardinality of $\mathcal{IN}(u, r, t)$:

$$s_{u,r}(t) = \begin{cases} |\mathcal{IN}(u, r, t)| & \text{if } r \in \mathcal{NR}_{item}(u, t) \\ 0 & \text{otherwise.} \end{cases} \tag{4.8}$$

Only the influence of a user's neighbourhood is taken into account. Neighbourhood is defined here as it was before by the commonality of interests of users which means that only users interested in the same repository as the considered individual influence him or her.

4.4.1.2 Adar-Adamic Index

Another variation on the use of local structure to predict link formation is the Adar-Adamic weighting scheme (IAA) originally presented by Adar and Adamic [31] in the context of web page features. This approach works similarly to the Common Neighbours approach but assigns a weight to each neighbour's contribution to the score. This weight is inversely proportional to the base 10 logarithm of the number of interests of the user; a user with a high degree in G_I —a large interest span— is assumed to be less discerning than a user with a small degree —a small interest span. It is assumed that a node of less degree filters its connections more and thus its recommendations are of higher quality. A user with low degree in G_I can also be supposed to mostly focus on their own projects. These works with perhaps less wide attention are promoted with this technique.

We further adapt the reformulation of this index by Lü and Zhou [32]. The assigned score is:

$$s_{u,r}(t) = \begin{cases} \sum_{v \in \mathcal{IN}(u,r,t)} \frac{1}{\log(1+|\mathcal{N}_{item}(v, G_I(t))|)} & \text{if } r \in \mathcal{NR}_{item}(u, t) \\ 0 & \text{otherwise.} \end{cases} \tag{4.9}$$

By definition $|\mathcal{N}_{user}(v, G_I(t)| \geq 1$ so 1 is added to the logarithm to prevent division by

zero. In IAA, the order of magnitude of a user's number of interests determines the user's interest span.

Liben-Nowell and Kleinberg [33] compared Common Neighbours and Adar-Adamic Index to 10 other link prediction techniques, and despite their simplicity these two heuristics have consistently performed well in the studied prediction task on five co-authorship networks. In particular the Adar-Adamic Index achieved the best predictive performance on two of the five considered networks and Common Neighbour has achieved it on one of them.

4.4.1.3 Resource Allocation Index

Resource Allocation (IRA) was presented by Zhou et al. [32]. Instead of considering the order of magnitude of a user's number of interests as is done in the Adar-Adamic weighting scheme, this weighting scheme considers the exact number of interests of a user. Again, more weight is given to a website member's 'advice' the more that member is parsimonious with his interests. The same serendipity interpretation as with the Adar-Adamic Index can be made here.

$$s_{u,r}(t) = \begin{cases} \sum_{v \in \mathcal{IN}(u,r,t)} \frac{1}{|\mathcal{N}_{user}(v, G_I(t)|} & \text{if } r \in \mathcal{NR}_{item}(u,t) \\ 0 & \text{otherwise.} \end{cases} \tag{4.10}$$

Zhou et al. [32] have compared these Common Neighbours and Adar-Adamic Index approaches to the Resource Allocation Index and found that on six disparate networks (instances of which include a protein interactions network, a power grid network and an Internet router topology network) Resource Allocation Index consistently performed the best with Common Neighbour tying or coming in second and Adar-Adamic coming in the top three. This high-performance tied with simplicity of implementation is why we chose all three approaches.

4.4.2 Social-based Link Prediction

Our proposed social-based link prediction techniques take into account the social neighbours of the considered individual. We adapt the above interest-based techniques to formulate the novel Common Social Neighbours, Social Adar-Adamic Index and Social Resource Allocation Index techniques.

The commonality of interests of different developers is not the only way to build a neighbourhood. In fact, the followership graph, G_F, provides a potentially richer definition of neighbourhood through time. Social networks generate additional signals of great import; advice from close friends and colleagues holds more persuasive power than from unknown individuals. We hypothesize these links to be of stronger significance as they are explicitly formed by the users; whereas, the previous interest-neighbour links are implicitly formed.

4.4.2.1 Common Social Neighbours

We adapt Common Neighbours to obtain Common Social Neighbours (SCN). It scores the set of repositories $\mathcal{SNR}_{item}(u,t)$ of the social neighbours of u at time t.

$$\mathcal{SNR}_{item}(u,t) = \left(\bigcup_{v \in \mathcal{N}_{user}(u, G_F(t))} \mathcal{N}_{user}(v, G_I(t)) \right) \setminus \mathcal{N}_{item}(u, G_I(t)). \tag{4.11}$$

The neighbours of u interested in repository r at time t are defined with respect to $G_F(t)$ as:

$$\mathcal{FN}(u,r,t) = \mathcal{N}_{user}(r, G_I(t)) \bigcap \mathcal{N}_{user}(u, G_F(t)). \tag{4.12}$$

The score is assigned as follows:

$$s_{u,r}(t) = \begin{cases} |\mathcal{FN}(u,r,t)| & \text{if } r \in \mathcal{SNR}_{item}(u,t) \\ 0 & \text{otherwise.} \end{cases} \tag{4.13}$$

Here the local social connections filter what is recommended and the score assigned to each recommendation is the number of social neighbours interested in the repository.

4.4.2.2 Social Adar-Adamic Index

The Social Adar-Adamic Index (SAA) is a variant of the Adar-Adamic weighting scheme where the followership graph is used to determine which user advice is taken from. It otherwise operates like the regular Adar-Adamic scheme.

The score is assigned as follows:

$$s_{u,r}(t) = \begin{cases} \sum_{v \in \mathcal{FN}(u,r,t)} \frac{1}{\log(1+|\mathcal{N}_{user}(v, G_I(t))|)} & \text{if } r \in \mathcal{SNR}_{item}(u,t) \\ 0 & \text{otherwise.} \end{cases} \tag{4.14}$$

4.4.2.3 Social Resource Allocation Index

Social Resource Allocation Index (SRA) is our adaptation of the Resource Allocation scheme with G_F used to determine the initial neighbourhood of the considered user.

The score is assigned as defined:

$$s_{u,r}(t) = \begin{cases} \sum_{v \in \mathcal{FN}(u,r,t)} \frac{1}{|\mathcal{N}_{user}(v, G_I(t)|} & \text{if } r \in \mathcal{SNR}_{item}(u,t) \\ 0 & \text{otherwise.} \end{cases} \tag{4.15}$$

The idea behind these social methods is also to take advantage of the notifications on GitHub. Social neighbours' actions are notified to a user on his or her GitHub homepage.

4.5 Markov Chain

The last technique we consider (MKV) uses the idea of a Markov chain [27] to predict which next repository is of interest only considering the previous repository of interest.

Given the history of interests leading to the event that user u shows interest toward repository r_n at arbitrary time t_n, denoted $u \to r_n(t_n)$, the probability of showing interest in r_n is simplified via the Markovian assumption:

$$P(u \to r_n(t_n) | u \to r_{n-1}(t_{n-1}), ..., u \to r_0(t_0)) = P(u \to r_n(t_n) | u \to r_{n-1}(t_{n-1})). \tag{4.16}$$

The probability $P(u \to r_n(t_n) | u \to r_{n-1}(t_{n-1}))$ is given by counting the instances of users showing interest in r_{n-1} and then immediately showing interest in r_n.

When a repository is new, it may not be common to observe transitions from it to another one. This is why we admit the more realistic proposition that a transition from r_{n-1} to r_n not only contributes to the likelihood of transitioning from r_{n-1} to r_n for other users, but also from r_n back to r_{n-1} to a lesser extent. Indeed we want to capture the correlation between the two while still favouring the transition from r_{n-1} to r_n.

Let $N_{r \to p}(t)$ be the number of observed transitions from repository r to repository p

prior to time t. The scoring equation must be reformulated as follows where p is the last repository user u has shown interest in and $s_{u,r}(t)|p$ denotes the score assigned to r given p:

$$s_{u,r}(t)|p = N_{p \to r}(t) + \alpha N_{r \to p}(t) \tag{4.17}$$

with α a parameter belonging to $[0, 1.0]$ assigning more or less weight to the reverse transition.

We justify this approach as being graph-based as it leverages explicitly the graph dynamics. We tie these transitions to the notion of interest paths. We want to test whether there are sequences of repositories that are more likely to be visited over time. In other words, do users tend to follow similar paths of repository interests strictly based on their current interests? The reverse transitions might also open up surprising avenues of interests.

4.6 Summary

We have presented above the five groups of recommendation techniques assessed in our research. Traditional popularity and similarity scorers represent the most commonly applied commercial solutions to the problem. They form the benchmark of our study, although we have chosen a similarity scorer that allows for niche recommendations. A novel application of Common Neighbours, Adar-Adamic Index and Resource Allocation Index link predictions to a real-time setting is proposed to take advantage of local structure and users with few interests. A first version is based on interest-neighbours and a second on social neighbours. Finally a Markov chain of transitions is also considered to see if there are interest paths over time and if interest progression has any potential. A time-window is also used to test the effectiveness of a crude forgetting tactic in each case.

The main goal of this selection of algorithms is to test their effectiveness at predicting future interests on GitHub and to compare the extent to which they provide serendipitous recommendations. The novelty of the studied data requires techniques that score high on recall first to shed some light on the processes at play.

In the next section we show the obtained results of using the above algorithms. The serendipity measures studied are also presented and a new one is proposed. Discussion of the assessment results follows.

Chapter 5

Results

In this chapter we assess the proposed methods on recall and on three serendipity measures. The first two serendipity measures —dissimilarity with past history and unexpectedness with respect to a base method— can be found in the literature, while the third measure is a proposed novel assessment tool that considers the social distance of a recommendation.

We present the measure of recall adapted to real-time recommendations first. Then we define the serendipity measures. Following these definitions are the experimental comparative results of the various techniques on these measures. Finally the implications of such results and their wider meaning for the GitHub milieu are discussed.

5.1 Measuring Recall

Serendipity requires some measure of accuracy. To measure accuracy we use a formulation of recall adapted to the evolution of the networks on GitHub. In particular, only TopKRecall and $\overline{\text{UserTopKRecall}}$ are considered in terms of accuracy metrics as precision is naturally averse to serendipity. Suggesting a wide list of repositories containing many unsubstantiated recommendations is not problematic as long as this list does contain correct predictions as well. The goal is to extend an individual's interest horizon and so producing many recommendations that do not concur with future interests is expected. Furthermore, disliked repositories are unknown on GitHub which makes it inappropriate to assign a great penalty to missed predictions. These predictions are not bad per se, but simply unproven.

Error metrics require ratings, but individuals do not rate repositories on GitHub. Hu et al. have proposed a method to turn implicit feedback into confidence and preference

ratings [17] in the context of TV watching. How many times and for how long a show is watched gives a real valued confidence rating in their context. Because two of the three interest indicating events (forking and starring) are not (technically) repeated events on GitHub, this avenue is unfit. Moreover, because the focus is on repository discovery, a single interest signal is enough. Recall is therefore chosen to measure accuracy.

We adapt the formulation of recall for top-K lists originally proposed by Cremonesi et al. [18]. Let $\delta(e, \mathcal{S})$ denote the indicator that the element e is in the set $\mathcal{S}$, formally defined as,

$$\delta(e, \mathcal{S}) = \begin{cases} 1 & \text{if } e \in \mathcal{S} \\ 0 & \text{otherwise.} \end{cases} \tag{5.1}$$

Let $\mathcal{V}_u^p$ be the set of validating repositories and their associated time for user u. $\mathcal{V}^t$ is the set of validating times only. $|\mathcal{V}^t| = \sum_{u \in \mathcal{U}} |\mathcal{V}_u^p|$. The TopKRecall and the average user recall are then:

$$\mathsf{TopKRecall} = \frac{1}{|\mathcal{V}^t|} \sum_{u \in \mathcal{U}} \sum_{(r,t) \in \mathcal{V}_u^p} \delta(r, \mathcal{R}_u^K(t)), \tag{5.2}$$

and

$$\overline{\mathsf{TopKRecall}} = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{|\mathcal{V}_u^p|} \sum_{(r,t) \in \mathcal{V}_u^p} \delta(r, \mathcal{R}_u^K(t)). \tag{5.3}$$

TopKRecall indicates the total number of correct predictions made, while $\overline{\mathsf{TopKRecall}}$ indicates the average over all users of correctly predicted percentages per user. The latter represents how on average a user would perceive the performance of the recommendation algorithm.

5.2 Measuring Serendipity

We measure the serendipity of the produced recommendations on three metrics: similarity-based serendipity as defined by Zhang et al. [11], unexpectedness-based serendipity proposed by Ge et al. [9], and our own serendipity metric based on social-distance.

5.2.1 Similarity-based Serendipity

The similarity-based serendipity metric is taken from Zhang et al.'s work on the Auralist serendipitous music recommender [11]. Equation 2.13 of Chapter 2 outlines their definition. We adapt it to our real-time setting. Let $\mathbf{v}_q(t)$ and $\mathbf{v}_r(t)$ be the profile vectors built according to Equation 4.3 at time t of repository q and r respectively. We define the unserendipity of a user u at time t for a set of K recommendations to be:

$$\overline{\mathsf{unserendipity}}(u,t,K) = \frac{1}{|\mathcal{N}_{item}(u,G_I(t))|} \sum_{q \in \mathcal{N}_{item}(u,G_I(t))} \frac{1}{K} \sum_{r \in \mathcal{R}_u^K(t)} \mathsf{similarity}_{\cos}(\mathbf{v}_q(t), \mathbf{v}_r(t)) \tag{5.4}$$

Averaging the $\overline{\mathsf{unserendipity}}(u,t,K)$ metric over all users $\mathcal{U}$ and over all times of the kept validating interests $\mathcal{V}_u^t$ for each user u, gives:

$$\overline{\mathsf{TopKSerendipity}_{\mathsf{sim}}} = 1 - \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{|\mathcal{V}_u^t|} \sum_{t \in \mathcal{V}_u^t} \overline{\mathsf{unserendipity}}(u,t,K) \tag{5.5}$$

The $\overline{\mathsf{TopKSerendipity}_{\mathsf{sim}}}$ metric reveals how different on average, in terms of cosine similarity, recommended repositories are from the ones a user has favoured in the past. The more distinct recommended repositories are from a user's history, the more likely it is that the recommendations are quite surprising. Recall must be computed separately and evaluated jointly to confirm the relevancy aspect of the implied serendipity.

5.2.2 Unexpectedness-based Serendipity

Ge et al. [9] measure serendipity by comparing the recommendations of the considered approach with those of a primitive method that provides obvious recommendations. The number of produced recommendations that are not predicted by the primitive method and that are relevant forms the basis of this assessment of serendipity.

Let $\mathcal{E}_u^K(t)$ be the primitive method's recommendations and $\mathcal{UXP}(u,t,K)$ be the time dependent unexpected set of repositories:

$$\mathcal{UXP}(u,t,K) = \mathcal{R}_u^K(t) \setminus \mathcal{E}_u^K(t). \tag{5.6}$$

In real-time, serendipity is then assessed as follows:

$$\overline{\mathsf{TopKSerendipity}_{\mathsf{unxp}}} = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{|\mathcal{V}_u^p|} \sum_{(r,t) \in \mathcal{V}_u^p} \delta(r, \mathcal{UXP}(u,t,K)). \tag{5.7}$$

For assessment purposes we chose to use POP as the primitive method generating the obvious set of recommendations $\mathcal{E}_u^K(t)$. POP is a legitimate choice as it recommends the most well-known repositories.

5.2.3 Social distance-based Serendipity

The concept of surprise found in the definition of serendipity can generally be thought as a measure of how far recommendations are from the user's sphere of awareness. For instance $\overline{\mathsf{TopKSerendipity}_{\mathsf{sim}}}$ models this sphere of awareness via similarity: the more similar to a user's history a repository is, the more likely it is that the user is aware of the project before it is recommended to him or her. The farther a recommended project is from this sphere of awareness, the more surprising it is. In $\overline{\mathsf{TopKSerendipity}_{\mathsf{unxp}}}$ the sphere of awareness is explicitly defined via $\mathcal{E}_u^K(t)$. Projects out of this set are unexpected.

In this vein, we propose to use social distance as a measure of a user's sphere of awareness. We follow the remarks of Dabbish et al. [22] regarding the usage of the follow feature: this feature is used to keep track of interesting individuals and by proxy of noteworthy new projects. Individuals use the ability to follow one another as a means to keep abreast of changes and new repositories of interests. Value to oneself and to the greater community is assessed through GitHub's transparent collaborative process. We measure how far on average a recommended repository is from the considered member's social connections.

To make this definition clear, we define $d_{social}(u, r, t)$ the social distance from a repository r to a user u at time t as the minimum graph distance on $G_F(t)$ from u to any member of $\mathcal{N}_{user}(r, G_I(t))$. If there does not exist a path from u to any member of $\mathcal{N}_{user}(r, G_I(t))$, $d_{social}(u, r, t)$ is set to the eccentricity of u in its connected component, i.e. the length of the longest path from u to any other user node in u's connected component. If u is not connected to any other user, the number of users minus one is assigned.

Averaging the $d_{social}(u, r, t)$ metric over all users $\mathcal{U}$ and over all validating interest times $\mathcal{V}_u^t$ for each user u, gives:

$$\overline{\mathsf{TopKSerendipity}_{\mathsf{sdist}}} = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{|\mathcal{V}_u^t|} \sum_{t \in \mathcal{V}_u^t} \frac{1}{K} \sum_{r \in \mathcal{R}_u^K(t)} d_{social}(u, r, t). \tag{5.8}$$

Naturally, the social based approaches SCN, SAA and SRA are expected to score 1 on this approach. Like $\overline{\mathsf{TopKSerendipity}_{\mathsf{sim}}}$, $\overline{\mathsf{TopKSerendipity}_{\mathsf{sdist}}}$ needs to be analyzed along recall as it would be easy to recommend the socially furthest away, but irrelevant, repositories.

5.3 Result Comparisons & Discussion

5.3.1 Recall

Table 5.1 shows TopKRecall and $\overline{\mathsf{UserTopKRecall}}$ for all the algorithms for $K = 10, 20, 40$. Despite score ties being broken randomly, it was observed that the standard deviation across 10 runs for each algorithm was negligible and so a single run is used instead. Different values of α for MKV were tried and $\alpha = 0.6$ gave the best recall results.

Algorithm	K = 10	K = 20	K = 40
POP	3.30% 4.56%	4.75% 6.15%	7.08% 8.44%
TPOPM	6.45 6.93	9.93 9.79	14.44 13.59
SIM	4.47 7.03	6.11 9.31	8.26 12.02
POPSIM	4.00 6.16	5.76 8.41	8.45 11.44
ICN	6.20 9.84	8.79 12.94	12.02 16.51
IAA	6.48 10.44	9.14 13.67	12.52 17.42
IRA	6.62 **10.68**	9.36 **13.98**	13.07 **18.03**
SCN	4.46 6.19	6.25 8.25	8.22 10.32
SAA	4.43 6.54	6.11 8.59	8.06 10.62
SRA	4.11 6.29	5.79 8.39	7.70 10.46
MKV0.6	**13.26** 10.60	**16.27** 12.62	**19.29** 14.61

The first inner column of each K column is TopKRecall and the second is $\overline{\mathsf{UserTopKRecall}}$. The highest score for each K is in bold.

Table 5.1 Recalls of algorithms for $K = 10, 20, 40$

The rise of the recall score across K's is expected as a higher K means a larger recommendation list that includes the previous recommendations of the smaller lists.

All of the proposed methods score higher on TopKRecall and $\overline{\mathsf{UserTopKRecall}}$ than the most basic non-personalized benchmark of POP at least. POPSIM is only marginally better and still less accurate than SIM except for TopKRecall($K = 40$); the pure similarity approach does better without the popularity factor taken into account. Repositories with

a similar composition of interested members as a given member's repository present in his or her history make better recommendations than overall popular projects.

The two best approaches are IRA and MKV. They distinctly divide the best results with IRA scoring higher on $\overline{\mathsf{UserTopKRecall}}$ while MKV scores (much) higher on $\mathsf{TopKRecall}$.

We see two factors contributing to the relatively high performance of MKV. First MKV takes timeliness into account by only considering the last repository of interest as its recommendation jumping point. This repository is more likely to be current and thus more related with other timely repositories. Considering timely projects, projects that receive at least one show of interest in the short period of time before the recommendation, is a beneficial approach as the near doubling of accuracy TPOPM has over POP seems to attest to. We come back to this point in the next section. Second, it seems that there are indeed some paths of interest over time. Correlation between repository transitions identifies interest migration patterns rather than strict interests.

For instance, if one typically uses the list of most popular repositories as a means to discover new repositories, MKV will be able to predict these repositories since every other user who does (and importantly *did*) the same contributes to making such next repositories more probable. This self-fulfilling process would even capture the effect of other top lists found outside of github.com. If a user tends to typically show interest in projects featured on prominent technology news sites such as Hacker News[1], Reddit[2], Slashdot[3] or others, then MKV can capture that pattern as well. Note that these top lists are usually similar to each other. Thus MKV will not only recommend a project featured on these sites because it will have seen many other users do the same type of attention transition, but the recommended repositories are chosen as to be related to a user's last interest and so more likely to be relevant for that user.

We note that personalization of recommendations through social peers (SCN, SAA, SRA) fares better than an impersonal top list (POP), but a personalization based on actual interests fares much better. Also taking time into account as TPOPM does surpasses any usage of the social network. Compared to the other approaches, social-based recommendations perform the weakest overall, but it is important to remember that the median social degree is only 3 and so most users have a smaller outreach to potentially recom-

[1] https://news.ycombinator.com/
[2] https://www.reddit.com/r/programming
[3] http://slashdot.org/

mendable repositories. For such a limited source of influence, the results are not orders of magnitude different from the others.

It can also be argued, that other approaches take into account social influence implicitly. We compare SCN, the best of the social-based link prediction techniques, to SIM, IRA and MKV to see how much these other techniques subsume SCN in Table 5.2. IRA and MKV do overlap significantly with SCN.

Algorithm	K = 10	K = 20	K = 40
SIM	21.9%	24.3%	27.6%
IRA	37.8	40.0	42.9
MKV	32.5	34.0	35.8

Table 5.2 Commonality of SCN's valid predictions with those of interest-based prediction techniques

SCN's recommendations overlap more with IRA's than with SIM's. Its overlap with IRA is probably due to the fundamental reliance on common neighbours of both IRA and SCN. However SCN is complementary to SIM. SIM relies on the overall composition of the interested members in a repository whereas SCN considers the number of friends interested in the repository rather than its complete composition. These two tactics lead to different results.

The near doubling of TopKRecall when considering the month-long previous time window for POP is surprising. TPOPM scores among the top results of the techniques other than MKV or IRA. Timeliness seems to provide an advantage that we confirm next.

5.3.2 Windowed Recall

The advantage of forgetting older interests and only relying on recent ones motivates the analysis of recall for windowed versions of the best algorithms of each category. Table 5.3 shows the obtained recalls for relevance windows of one month, one week and one day. The results for month, week and day windowed versions of SCN are two orders of magnitude lower than the rest and as such they are not displayed here.

The highest recalls are now those of TIRAD and TPOPSIMD. Therefore, on average, recommending repositories from peers is more effective when the weight given to each peer's repository is inversely proportional to the interest span of that peer or when that peer's repository is quite popular. TPOPSIMD leverages the high results of TPOPD to signifi-

Algorithm	K = 10	K = 20	K = 40
TPOPM	6.45 6.93	9.93 9.79	14.44 13.59
TPOPW	12.52 11.24	17.44 15.00	23.14 19.51
TPOPD	19.60 15.95	24.84 19.84	30.35 24.07
TSIMM	3.53 5.53	4.70 6.96	6.25 8.74
TSIMW	2.96 3.68	4.19 4.93	6.17 6.78
TSIMD	3.81 3.19	6.61 4.80	11.1 6.90
TPOPSIMM	7.34 8.69	11.39 12.33	16.39 16.94
TPOPSIMW	13.63 12.77	19.32 17.27	25.68 22.54
TPOPSIMD	20.51 16.82	26.33 **21.12**	**32.50 25.95**
TIRAM	9.98 12.47	14.60 16.81	20.72 22.20
TIRAW	15.99 15.29	22.07 20.15	28.64 25.18
TIRAD	**22.10 17.13**	**27.42** 20.82	**32.50** 24.16
TMKV0.6M	11.93 8.90	14.57 10.56	17.16 12.20
TMKV0.6W	9.81 6.30	11.75 7.41	13.35 8.25
TMKV0.6D	5.56 3.02	6.09 3.24	6.42 3.37

The first inner column of each K column is TopKRecall and the second is $\overline{\text{UserTopKRecall}}$. The highest score for each K is in bold. Only IRA was chosen from its group as it had the strongest previous recall scores.

Table 5.3 Recalls of best windowed algorithms

cantly boost its selection process. Another way to see this is that its refinement of choosing among interest-neighbour gains it only a few percentage points above TPOPD. Filtering solely by commonality of interest gives a slight recall advantage. Choosing selective peers which act as apt guides in the plethora of on-going activities provides great accuracy on its own. Combined with timeliness IRA is quite effective.

All of the other algorithms show a degradation of performance when a smaller time window is used. The limited outreach of SCN is worsened making its recommendations devolve into randomness. We confirm this in the next section. Strict similarity and Markov transitions also see their pool of influence much reduced when considering a small time window. The absence of related activities (transitions or interests) in the preceding time window provides less information to benefit from than the full history.

5.3.3 Distribution of Recall Scores

The standard deviation for $\overline{\mathsf{UserTopKRecall}}$ is not provided because the obtained distribution clearly does not follow a normal Gaussian. The effectiveness of the techniques remains restricted to a minority of users. We provide the distribution of recall scores in Figure 5.1 for MKV, TIRAD, TPOPSIMD and $K = 20$. A similar distribution is observed for other Ks. Each recall percentage interval includes its lowest value and excludes the highest value. The distribution is over percentages. Therefore a few hits or misses translate into large swings of percentages in some cases, while numerous hits or misses are needed to provide the same swings in other cases.

What is observed is that a large portion of users are still eluding the best techniques. The distribution is multi-modal with bumps past the 0-10 range in the 20-40, 50-60 and 90-100% intervals. Most of the gain due to using a time window is found in providing users with no previously correct recommendations some number of correct recommendations.

A challenge with real-time predictions like ours is the variation of 'training' data per user and over time. An approximation of this data with respect to interests per user over time is the total number of interest instances for that user. Figure 5.2 shows the distributions of $\overline{\mathsf{UserTop20Recall}}$ with respect to the number of total interests per users.

We see that the median of $\overline{\mathsf{UserTopKRecall}}$ (other values of K exhibit the same pattern) rises with the number of interests. It is indeed the users with the lowest number of interests that are most poorly recommended against. As more interests can be taken into account,

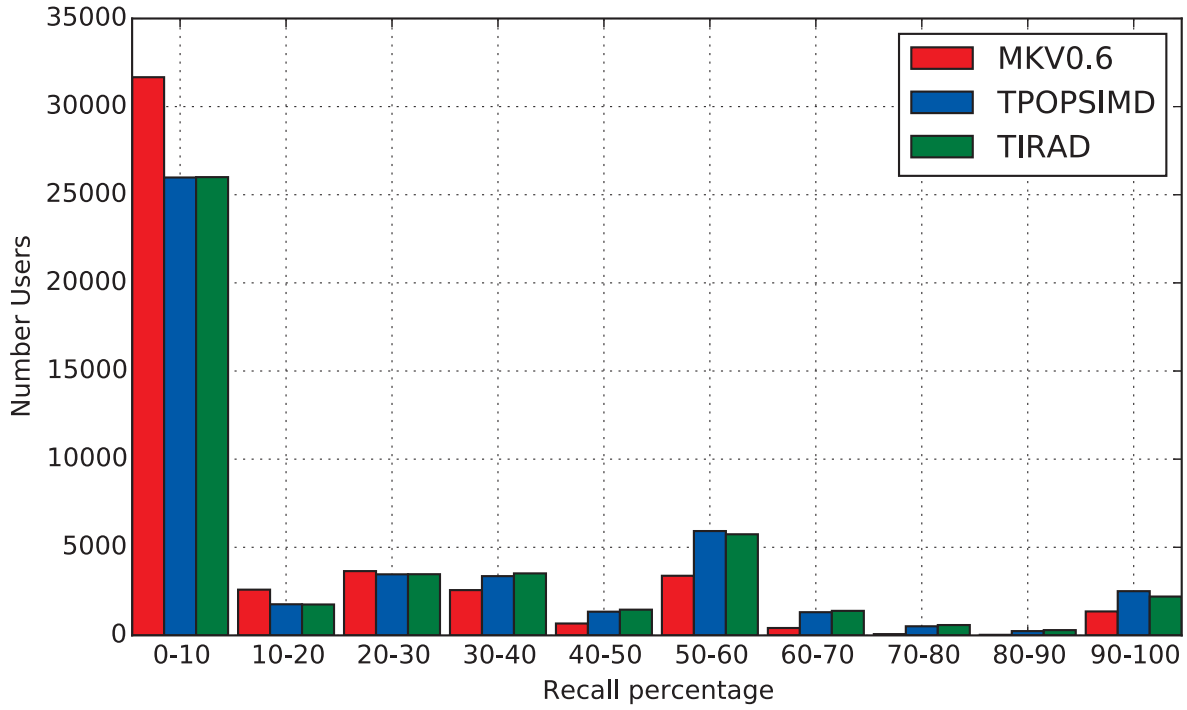


Fig. 5.1 Recall distribution of users

the more accurate are the proposed techniques (a similar trend is observed for TPOPSIMD).

5.3.4 Serendipity

We present here the obtained serendipity measures for the similarity, unexpectedness and social distance-based approaches. Table 5.4 summarizes the results obtained for the best recall algorithms of each category with the similarity-based metric. Table 5.5 summarizes the results obtained for the unexpectedness-based metric and Table 5.6 summarizes our findings for social distance-based approaches.

5.3.4.1 Similarity-based Serendipity

In practice similarity-based serendipity measures how different from the past all recommendations are. As can be seen from Table 5.4, TPOPD is slightly more serendipitous than the alternatives. TPOPD does not cater to users' histories and thus recommendations are dissimilar from their history. TPOPD also achieves high recall scores and so according to

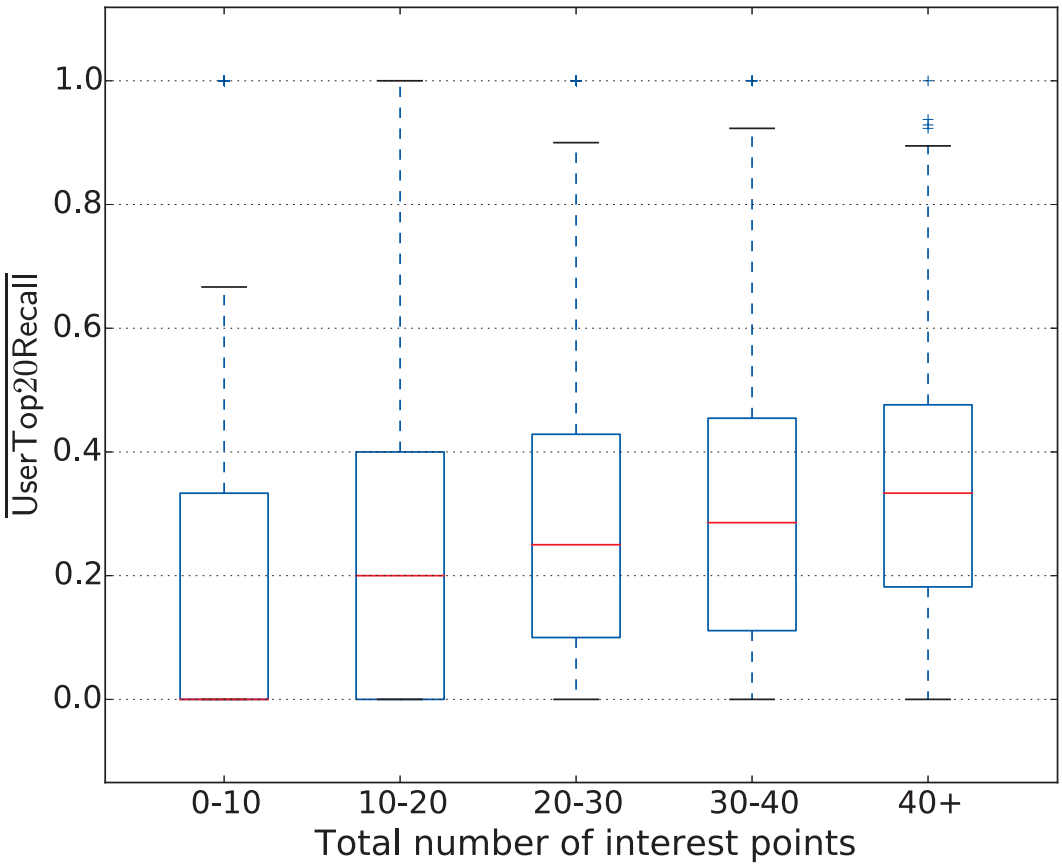


Fig. 5.2 Distributions of TIRAD $\overline{\text{UserTop20Recall}}$ per number of total interests

this serendipity metric it serves its purpose of presenting serendipitous recommendations; the recommendations are accurate and less likely to be shared by the same users as before.

The differences between results are not tremendous however. In effect the results shown are averages across users but the standard deviations are relatively large as was the case for recall. The distribution is similar.

5.3.4.2 Unexpectedness-based Serendipity

Next we consider the performance of the algorithms in terms of unexpectedness serendipity. In Table 5.5, the top result is TIRAD. In part this is unsurprising as TIRAD scores

Algorithm	K = 10	K = 20	K = 40
TPOPD	**0.9861**	**0.9876**	**0.9885**
SIM	0.9097	0.9231	0.9368
TPOPSIMD	0.9797	0.9825	0.9841
TIRAD	0.9805	0.9833	0.9860
SCN	0.9697	0.9756	0.9817
MKV0.6	0.9659	0.9741	0.9806

The best score for each K is in bold.

Table 5.4 Similarity-based serendipity of algorithms

among the highest $\overline{\mathsf{UserTopKRecall}}$ and TPOPSIMD is presumably more likely to recommend repositories POP would recommend. It however confirms that TIRAD averaged more correct predictions different from POP than the alternatives; it could have been that another method with a lower recall deviated from POP more.

Algorithm	K = 10	K = 20	K = 40
TPOPD	0.1222	0.1484	0.1713
SIM	0.0617	0.0788	0.0979
TPOPSIMD	0.1261	0.1540	0.1786
TIRAD	**0.1470**	**0.1710**	**0.1876**
SCN	0.0501	0.0661	0.0816
MKV0.6	0.0715	0.0811	0.0873

The best score for each K is in bold.

Table 5.5 Unexpectedness-based serendipity of algorithms

The difference between TPOPSIMD and TIRAD is not big either. Already the differing conclusions between Table 5.4 and Table 5.5 show the complexity of assessing serendipity. Table 5.6 adds to this.

5.3.4.3 Social distance-based Serendipity

In this measurement of serendipity, outlined in Table 5.6, SIM is the most serendipitous. Its suggestions are on average the most distant from a user's social circle. As we have seen in Table 5.2 there is little overlap between social and the purely interest-based approaches.

Note that the recommendations that fall outside the social components would skew the results much higher and hide the true most common size of the social components. Because

of the high number of total recommendations (from 89 to 95%) that fall in users' social components, the results shown exclude recommendations outside social components.

The observed average social distances are very low. This indicates that the recommendations across all techniques tend to be socially close to their associated user.

The fact that SCN does not show $\overline{\mathsf{TopKSerendipity}_{\mathsf{sdist}}} = 1$ is a testament to its lack of outreach. After recommending all repositories socially connected, it reverts to recommending random repositories until its quota of K repositories is met.

Algorithm	K = 10	K = 20	K = 40
TPOPD	2.339 (91.6%)	2.424 (91.6%)	2.489 (91.6%)
SIM	**2.946** (89.5)	**2.975** (89.4)	**3.000** (89.4)
TPOPSIMD	2.161 (92.3)	2.240 (92.2)	2.296 (92.2)
TIRAD	2.444 (92.0)	2.507 (92.0)	2.580 (92.0)
SCN	1.035 (95.1)	1.203 (94.4)	1.457 (93.7)
MKV0.6	2.472 (91.1)	2.542 (90.9)	2.610 (90.7)

The best score for each K is in bold. The percentage in parentheses is the ratio of the number of recommendations that are in a social component over the total number of recommendations.

Table 5.6 Social distance-based serendipity of algorithms

MKV surprises here, by performing rather well: it ranks second on this metric. MKV thus shows serendipitous potential as well.

5.4 Overall

We conclude these results with a few remarks. Like recall, serendipity grows with K in each case. Lower ranked recommendations are typically not similar to past interests and the less personalized a recommendation is the more likely it is to not be part of a user's social component. $\overline{\mathsf{TopKSerendipity}_{\mathsf{unxp}}}$ is positively correlated with recall and so it naturally grows with K.

As far as social influence is concerned on GitHub, it in fact does not seem to be a major factor in the discovery of new repositories, at least with respect to techniques that specifically target it. Followership links created before March 2011 are missing. The paucity of social links may explain the lower performance. However creating links between individuals based on their participation in the same commit discussion seems tenuous. In addition,

MKV and TIRAD do significantly overlap with SCN. Social influence plays some role, but less so than other factors.

To have a better idea of the serendipity scores, recall must be taken into account. Figures 5.3, 5.4 and 5.5 position these algorithms with respect to their $\overline{\text{UserTopKRecall}}$ and a measure of serendipity.

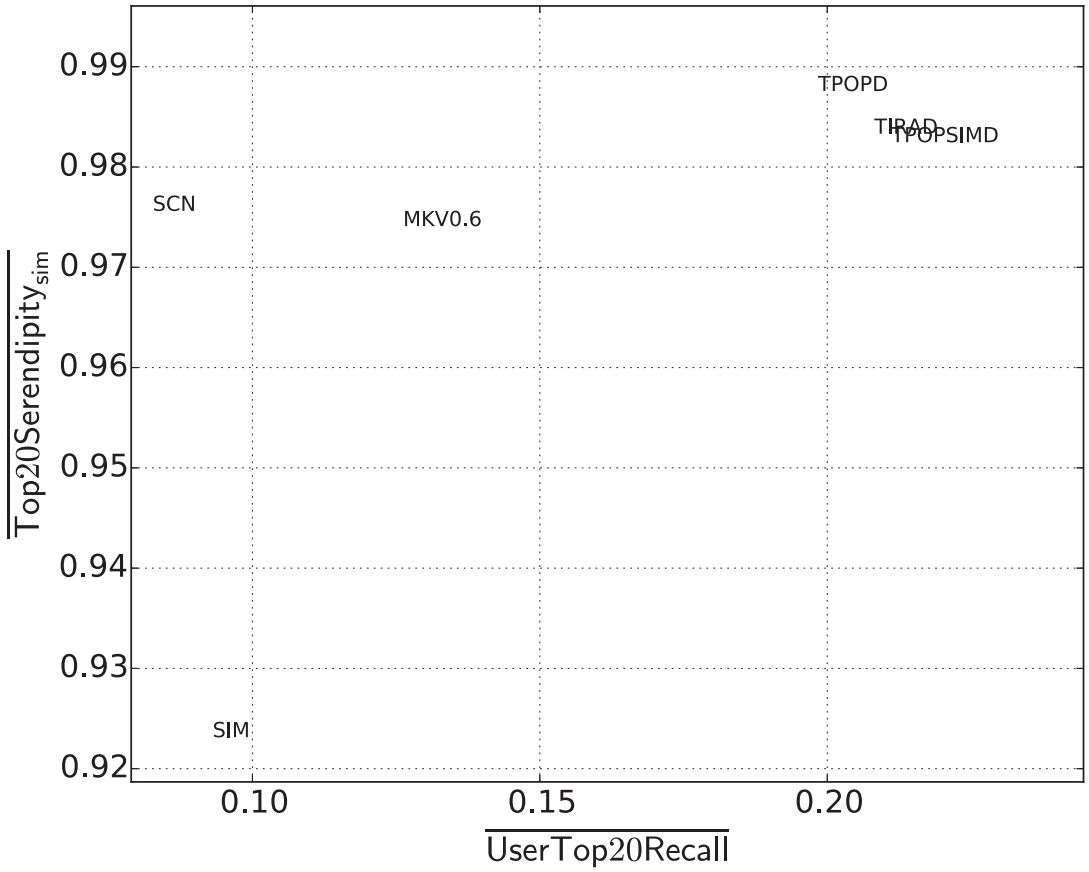


Fig. 5.3 Algorithms with respect to recall and $\overline{\text{TopKSerendipity}_{\text{sim}}}$

The day-timeliness indicates the fast-paced nature of attention in the GitHub ecosystem. Taking it out of the equation, we can see that MKV is a surprisingly serendipitous technique. It combines relatively high recall with high serendipity scores in nearly each case. Also it is worth noting that MKV will recommend less popular repositories which allows it to potentially recommend up-and-coming repositories rather than established pop-

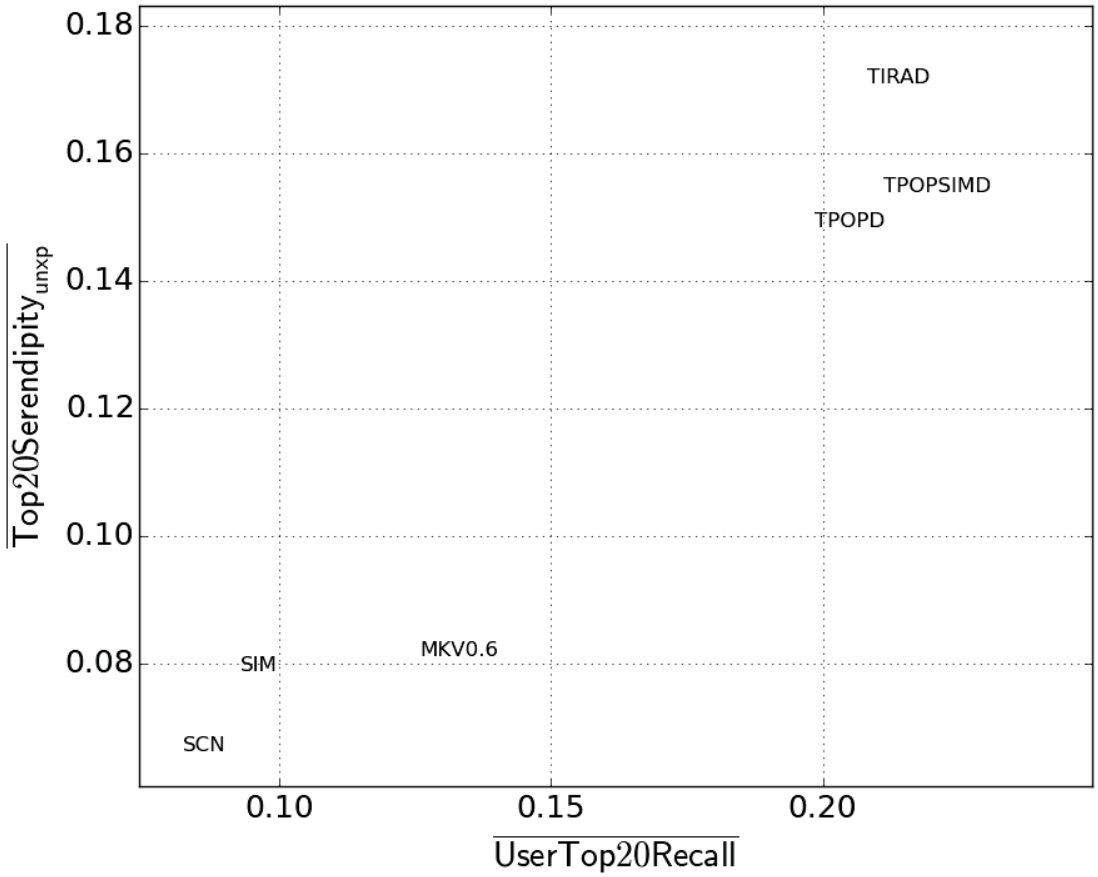


Fig. 5.4 Algorithms with respect to recall and $\overline{\text{TopKSerendipity}_{\text{unxp}}}$

ular repositories since MKV will start recommending a repository as soon as it has seen someone show interest in it. We leave this avenue of research for future work.

5.5 Summary

In this chapter we have presented $\overline{\text{TopKSerendipity}_{\text{sim}}}$, $\overline{\text{TopKSerendipity}_{\text{unxp}}}$ and the novel $\overline{\text{TopKSerendipity}_{\text{sdist}}}$, three means of assessing serendipity. Our experimental recall results and serendipity results were then produced and compared. Although the top approaches, TPOPD, TPOPSIMD and TIRAD, stay in the same serendipitous region across different measurements, SCN performs markedly better for $\overline{\text{TopKSerendipity}_{\text{sim}}}$, reciprocally SIM per-

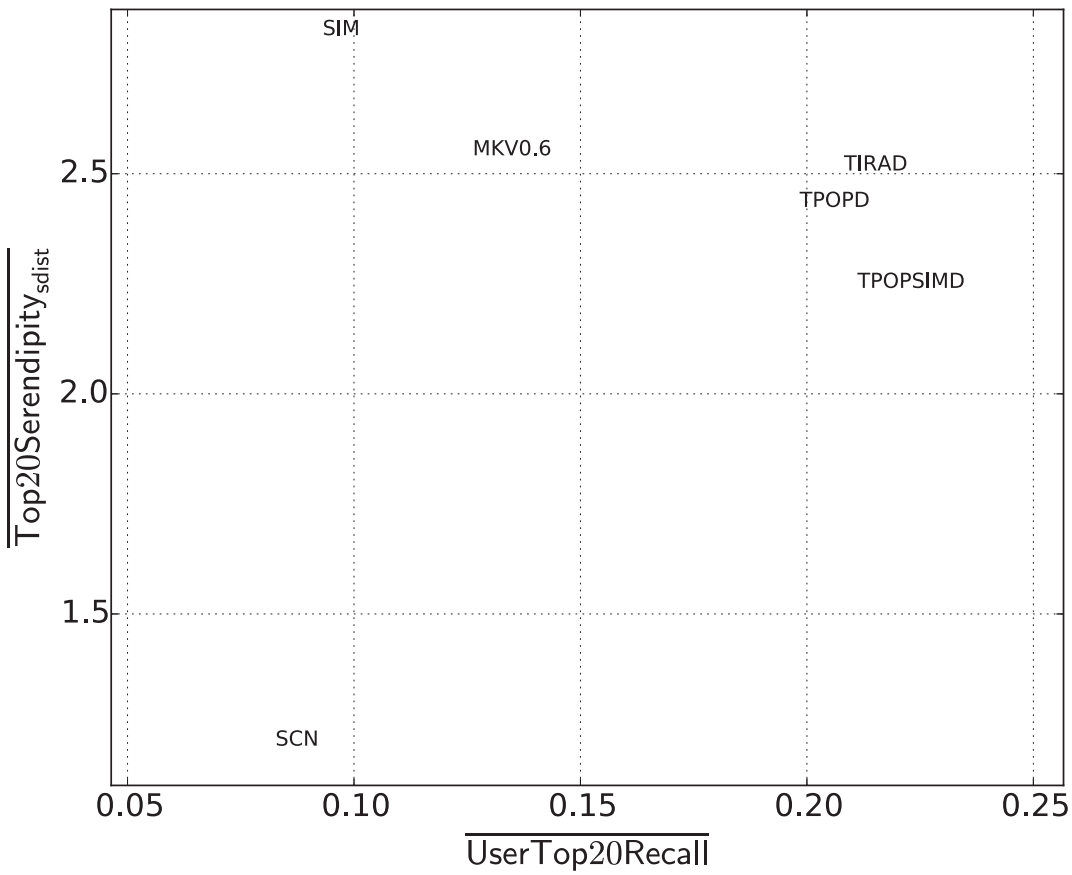


Fig. 5.5 Algorithms with respect to recall and $\overline{\text{TopKSerendipity}_{\text{sdist}}}$

forms much better for $\overline{\text{TopKSerendipity}_{\text{sdist}}}$ and MKV consistently positions itself in a second position of sort across all measurements. The exact numerical results show that different definitions of serendipity favour different techniques over others. We can conclude that the measures of serendipity used in the literature do not agree on the relative serendipity of algorithms. The addition of the $\overline{\text{TopKSerendipity}_{\text{sdist}}}$ also shows how discovery is perhaps not as reliant on social connections as user perception [22] might lead us to believe. One must then be careful when choosing serendipity metrics and interpreting the obtained results.

There are no clear winner either among the top approaches although TIRAD, TPOPSIMD and TPOPD perform well accuracy-wise and in two recall metrics. This is because the differences in results between the techniques are not tremendous and the standard de-

viation of the results cannot be trusted to properly frame the precision. A large portion of users have low 'training' data and are therefore hard to predict against which skews the results.

Timeliness on GitHub is an important factor —specifically past day activity. In fact, GitHub's trending page was implemented in August 2013[4] which might imply they have discovered this very same fact about their data. A form of timeliness is also present in MKV which performs surprisingly high in all serendipity metrics. This approach is worth investigating further.

[4]https://github.com/blog/1585-explore-what-is-trending-on-github

Chapter 6

Conclusion

6.1 Concluding Remarks

We have put forward in this thesis a first preliminary comparison of serendipity measurements on graph based techniques for a novel dataset of GitHub activities. Only newly created repositories were selected to track their growth and only their associated users were kept. The similarity, link prediction and Markov chain approaches have all been generated in pseudo real-time to verify their effectiveness in the same context as GitHub would be running these techniques in. These graph-based approaches were compared with an all-time popularity recommender and a time-interval based popularity recommender.

The comparison of recall across these techniques revealed the effectiveness of considering only past day activity in the recommendation process. This is a surprising result. It implies less past data is more informative and that GitHub is a very dynamic setting where new interests are made everyday. Interestingly, GitHub must have noticed this pattern as well since it implemented a Trending page based on last month, week and day of activity in August 2013.

From this first side-by-side comparison of serendipity measures, we can tentatively say that the intuition of different serendipity concepts leading to different serendipity results is true to some extent. The dissimilarity-based serendipity slightly favours unpersonalized last day popularity. The unexpectedness-based serendipity slightly favours a last day version of resource allocation and our introduced social distance-based serendipity favours the (all time) similarity approach which is solely interest-based. No technique is strictly more serendipitous than all the others across all three proposed serendipity measurements.

Another surprising conclusion drawn from this study is the apparent less effectual approach of relying mostly on social links. In effect their lower performance is in part due to their lower numbers. The observed users on GitHub are better predicted for by relying on their interests rather than their social links because the interests are more numerous and influential social links may be lacking. During the observed period, the influence of interests is stronger than the influence of social connections.

On the other side, the introduced Markov chain approach for this prediction task fared relatively well on both recall and serendipity measures. Its ability to take timeliness into account and reproduce popular interest transition patterns are its strengths. We hypothesize that its inability to reach as high performance as last day interest-based recommendation approaches is because of the lack of transition patterns from or to repositories created in such a short time frame.

6.2 Future Work

This work has looked at simple techniques under a realistic setting. A great number of users have not been recommended any valid repositories however. As such a first avenue of future work is to try more advanced approaches in this real time setting where validation is done in chronological order. Blending and latent factors are avenues of interest in this regard. The realized comparison of recommendations in common between certain techniques lets us believe that trying mixed techniques that put more weight on certain algorithms versus others for certain users might be beneficial. The time model used was simple as well. An exponential time decay could be explored to enhance results by gradually phasing out the importance of older interests rather than removing their influence abruptly.

Looking beyond network-based predictive approaches to methods based on natural language processing, activity or language of the repositories is another avenue of work possible. Such an approach was tried in a limited fashion, but was discarded because of the lack of available features. Only 7.8% of captured repositories had READMEs in the period considered and only 36.6% had a retrievable language. Extending the period or taking a more recent one with these findings in mind is a worthwhile next step.

Considering an even larger data set might help some techniques such as the Markov chain. The Markov chain approach relies on the last repository of interest, but in our dataset this repository had to be created during the year considered. In effect, the last

repository of interest in our analysis might not be the true last repository of interest if that repository was created before the observed period of time. By taking into account all interests, even the interests for repositories not created during the observed period of time, the generated transitions would better reflect reality. This avenue was not investigated in this thesis due to computational memory constraints. Processing all this data requires dedicated technology that goes beyond a simple desktop computer.

Another avenue of work the Markov chain algorithm opens for further investigation is its ability to potentially predict up-and-coming repositories. Because it does not rely on overall or even day-of popularity, it might recommend repositories that *will* be popular. Understanding if this is indeed the case is an open question.

All in all serendipity is hard to assess automatically on a large scale without user feedback or content-based data. The three serendipity metrics that are considered explore different definitions of surprise. Understanding whether there is a favoured one among members of a given community could be done by asking a large sample of them which serendipitous ranking is more 'serendipitous' to them and which better reflect their own discovery process. A fine grained user survey asking how repositories of interest were discovered might also highlight the avenues to focus on with this dataset. Indeed the original discovery process is unknown and more than likely to vary among users. GitHub is well-placed to lead such a survey. It could also leverage its undisclosed data such as the logs of which page were visited by which users. A Markov chain based on these visits might be an interesting avenue.

Appendix A

Appendix

A.1 Source Code and Processed Data

The source code used for this research is made available at github.com/fenekku/Masters. The two networks used in this research, the followership network and the interest network, are also made available there in Matrix Market format.

References

[1] S. M. McNee, J. Riedl, and J. Konstan, "Accurate is not always good: How accuracy metrics have hurt recommender systems," in *Extended Abstracts of the 2006 ACM Conference on Human Factors in Computing Systems (CHI 2006)*, 2006. 8, 9

[2] E. Pariser, *The Filter Bubble: How the New Personalized Web Is Changing What We Read and How We Think*. Penguin Group US, 2011. 8, 9

[3] J. L. Herlocker, J. A. Konstan, L. G. Terveen, John, and T. Riedl, "Evaluating collaborative filtering recommender systems," *ACM Transactions on Information Systems*, vol. 22, pp. 5–53, 2004. 9, 10, 13, 14

[4] F. Ricci, L. Rokach, and B. Shapira, *Introduction to recommender systems handbook*. Springer, 2011. 9, 10, 12, 13

[5] P. André, J. Teevan, and S. T. Dumais, "From x-rays to silly putty via uranus: Serendipity and its role in web search," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, (New York, NY, USA), pp. 2033–2036, ACM, 2009. 10, 15

[6] K. Sugiyama and M.-Y. Kan, "Serendipitous recommendation for scholarly papers considering relations among researchers," in *Proceedings of the 11th Annual International ACM/IEEE Joint Conference on Digital Libraries*, JCDL '11, (New York, NY, USA), pp. 307–310, ACM, 2011. 12, 13, 14, 20, 48

[7] M. Zhang and N. Hurley, "Avoiding monotony: improving the diversity of recommendation lists," in *Proceedings of the 2008 ACM conference on Recommender systems*, pp. 123–130, ACM, 2008. 12

[8] G. Patil and C. Taillie, "Diversity as a concept and its measurement," *Journal of the American Statistical Association*, vol. 77, no. 379, pp. 548–561, 1982. 13

[9] 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. 14, 58, 59

[10] Q. Lu, T. Chen, W. Zhang, D. Yang, and Y. Yu, "Serendipitous personalized ranking for top-n recommendation," in *Proceedings of the The 2012 IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technology-Volume 01*, pp. 258–265, IEEE Computer Society, 2012. 14, 17

[11] Y. C. Zhang, D. Ó. Séaghdha, D. Quercia, and T. Jambor, "Auralist: introducing serendipity into music recommendation," in *Proceedings of the fifth ACM international conference on Web search and data mining*, pp. 13–22, ACM, 2012. 14, 15, 19, 20, 58, 59

[12] M. Nakatsuji, Y. Fujiwara, A. Tanaka, T. Uchiyama, K. Fujimura, and T. Ishida, "Classical music for rock fans?: Novel recommendations for expanding user interests," in *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, CIKM '10, (New York, NY, USA), pp. 949–958, ACM, 2010. 14, 15

[13] U. Bhandari, K. Sugiyama, A. Datta, and R. Jindal, "Serendipitous recommendation for mobile apps using item-item similarity graph," in *Information Retrieval Technology*, pp. 440–451, Springer, 2013. 14

[14] N. J. Belkin and W. B. Croft, "Information filtering and information retrieval: Two sides of the same coin?," *Commun. ACM*, vol. 35, pp. 29–38, Dec. 1992. 15, 18

[15] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry, "Using collaborative filtering to weave an information tapestry," *Commun. ACM*, vol. 35, pp. 61–70, Dec. 1992. 15, 18

[16] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009. 16, 48

[17] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets," in *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pp. 263–272, IEEE, 2008. 17, 18, 58

[18] 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. 17, 18, 58

[19] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: Item-to-item collaborative filtering," *Internet Computing, IEEE*, vol. 7, no. 1, pp. 76–80, 2003. 18, 45, 49, 50

[20] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *the Journal of machine Learning research*, vol. 3, pp. 993–1022, 2003. 19

[21] J. Suchal and P. Návrat, "Full text search engine as scalable k-nearest neighbor recommendation system," in *Artificial Intelligence in Theory and Practice III*, pp. 165–173, Springer, 2010. 22, 23, 40

[22] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in github: transparency and collaboration in an open software repository," in *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pp. 1277–1286, ACM, 2012. 22, 23, 24, 32, 60, 72

[23] F. Thung, T. F. Bissyandé, D. Lo, and L. Jiang, "Network structure of social coding in github," in *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*, pp. 323–326, IEEE, 2013. 22, 25, 32

[24] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining github," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, pp. 92–101, ACM, 2014. 22, 27, 30, 40

[25] A. Lima, L. Rossi, and M. Musolesi, "Coding together at scale: Github as a collaborative social network," *arXiv preprint arXiv:1407.2535*, 2014. 22, 26, 28, 32, 39, 40

[26] A. Begel, J. Bosch, and M.-A. Storey, "Social networking meets software development: Perspectives from github, msdn, stack exchange, and topcoder," *Software, IEEE*, vol. 30, no. 1, pp. 52–66, 2013. 24, 31

[27] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web.," 1999. 25, 55

[28] D. Easley and J. Kleinberg, *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge University Press, 2010. 26

[29] "Ecma-404 (rfc 4627) the json data interchange format," ECMA Standard, ECMA International, October 2013. 32

[30] M. E. J. Newman, "Clustering and preferential attachment in growing networks," *Phys. Rev. E*, vol. 64, p. 025102, Jul 2001. 51

[31] L. A. Adamic and E. Adar, "Friends and neighbors on the web," *Social networks*, vol. 25, no. 3, pp. 211–230, 2003. 51, 52

[32] T. Zhou, L. Lü, and Y.-C. Zhang, "Predicting missing links via local information," *The European Physical Journal B-Condensed Matter and Complex Systems*, vol. 71, no. 4, pp. 623–630, 2009. 51, 52, 53

[33] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *Journal of the American society for information science and technology*, vol. 58, no. 7, pp. 1019–1031, 2007. 53