TRUST AWARE RECOMMENDER SYSTEM FOR SOCIAL CODING PLATFORMS (GitHub CASE STUDY)

A Thesis Presented to the

Department of Computer Science

African University of Science and Technology

In Partial Fulfillment of the Requirements for the
Degree of Master of Science

By

Nkoro Joseph Ahamefula

Abuja, Nigeria November, 2017

.

CERTIFICATION

This is to certify that the thesis titled "*Trust Aware Recommendation System for Social Coding Platforms (GitHub Case Study)*" submitted to the school of postgraduate studies, African University of Science and Technology (AUST), Abuja, Nigeria, for the award of the Master's degree is a record of original research carried out by

Nkoro Joseph Ahamefula in the Department of Computer Science.

TRUST AWARE RECOMMENDER SYSTEM FOR SOCIAL CODING PLATFORMS (GitHub CASE STUDY)

By

Nkoro Joseph Ahamefula

A THESIS APPROVED BY THE COMPUTER SCIENCE DEPARTMENT

RECOMMENDED: ----------

Supervisor, Dr. Victor Odumuyiwa

Head, Department of Computer Science

APPROVED: ----------

Chief Academic Officer

Date

ABSTRACT

Social networking systems have found their way into all sectors of life. With the advent of social coding platform like GitHub, networks of developers can be inferred based on the projects they participated in. When a new project is created by a developer on such social coding platforms, these platforms lack the capacity to recommend potential collaborators. Recommender systems are software techniques and tools that give item suggestions to users who might be interested in such an item. Having identified this problem, we developed ProjectTrust, a trust-aware recommender model which evaluates trust between projects and developers. A natural language processing approach was identified to be a good tool for text feature extraction in GitHub readme files. As the verification of the proposed framework, experiments using real social data from GitHub are presented and results show the effectiveness of the proposed approach.

Keywords: Trust-aware, Recommender Systems, Natural Language Processing, Term Frequency, Inverse Document Frequency.

ACKNOWLEDGEMENTS

I would like to express my gratitude to God almighty for making this work come to a fulfillment.

This work wouldn't have been possible without the financial support of African Development Bank and African University of Science and Technology through the Masters of Science scholarship which they granted to me and other well deserving colleagues. I am specially indebted to my supervisor Dr. Victor Odumuyiwa, who throughout this period has been supportive and has actively provided me with the academic time to pursue this goal and fulfill it. I will not forget to mention the fatherly support of Prof. Amos David, Head of Department computer science and engineering stream of African University of Science and Technology, for his continuous effort made in making sure that I and my colleagues never relented in completing our tasks. Other faculties of the computer science stream like Professor Lehel Csato, Professor Ben Abdallah, Professor Mohamed Hamada and Dr Ekpe Okorafor, Dr. Onime Clement, Dr Moses Akanbi, Dr victor Odumuyiwa, my supervisor and many others, your efforts and inspiration are highly appreciated. The constructive criticism of my fellow colleagues (Abraham Ezema, and David Clement), and other persons whom I have had the pleasure to work with during this period of my Masters degree study are also appreciated.

Nobody can be more important to me in the pursuit of this project than members of my family, the Nkoro's. Special thanks goes to my beloved parents Elder and Mrs Nkoro for their unending prayers and support and I cannot but be grateful to you both. My siblings Kelvin, Davidson, Peace, Clara, the iheanacho's and Ogechi, you all have made me strong during this period. For friends who were more like a family to me, Akuma, Kingsley, Engr uche, Dr juliet and all members of Church of Christ Kado, I am indeed grateful to you all for always advising and praying for me.

DEDICATION

To my late Grandmother who taught me a lot about life.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE
INTRODUCTION OF CONCEPT

1.1 INTRODUCTION

With the growth of web technology, there has been an explosive growth in the size of content available on the Internet, social network interaction has exploded as well and has become a regular part of people's life. Other social lives activities like buying and selling now have a place to fit into social networks. Researchers and scholars need information and resources from on-line document repositories and digital libraries for proper conducting of research work and they also require collaborative researches; casual chatting and communication via mails are also parts of the exploits made from advances in web technology. The Web has turned to the best medium for many database applications, like e-commerce and digital libraries. Many of these applications have even extended their functionalities by making use of APIs (Application Programming Interfaces).

The cyber world has increasingly become social in the last 10 to 15 years, but the productivity implications remain insufficiently exploited. We can track a person’s moment-by-moment status updates on Facebook, photo update on Instagram, and updates on Twitter, blogs and wikis.

1.2 SOCIAL CODING AND VERSION CONTROL SYSTEMS

Social networks are the biggest explosions of the 21st century. We possess the technology to remain connected perpetually with our network, and they could be considered as resources for reaching out to people from all around the world in an instant. The web has enabled communities to emerge and collaborate on challenges like building the most sizably voluminous compendium of all human knowledge, Wikipedia, providing the resources and tools for communities to achieve a common goal.

The quick advancement of social coding devices is leading to a transformation in software product development. Social communications have turned into an essential factor in the assessment of

the software product improvement process. Version control systems (VCS) are the basic piece of a social coding stage. These days, different VCS instruments, like CVS, SVN, Git and so on, are much of the time utilized by software advancement groups. Software developers can build their own code versions, and submit changes into the decentralized VCS frameworks. Distinctive versions of a software are managed by the VCS framework, and potential clashes of software products are avoided. Early VCS frameworks are utilized just by fairly small software development groups, and are for the most part installed inside small network systems, like organizational LANs. The quantity of projects maintained under those early VCS frameworks are moderately few. As Git can make software development coordination effortless using its distributed coding collaboration feature, little wonder why it is picking up its popularity. “The forty year history of version control tools shows a steady movement toward more concurrency. In first generation tools, concurrent development was handled solely with locks. Only one person could be working on a file at a time. The second generation tools are a fair bit more permissive about simultaneous modifications, with one notable restriction. Users must merge the current revisions into their work before they are allowed to commit. The third generation tools allow merge and commit to be separated” (ericsink.com, 2017).

Table 1.1: Generations of Version Control Systems

Generation	Networking	Operations	Concurrency	Examples
First	None	One file at a time	Locks	RCS, SCCS
Second	Centralized	Multi-file	Merge before commit	CVS, SourceSafe Subversion Team Foundation Server
Third	Distributed	Changesets	Commit before merge	Bazaar Git Mercurial

With the current advances in distributed computing innovation, distributed social coding gets a major lift. Prevalent social coding stages would now be able to have a great many software projects. These days, an ever-increasing number of individuals acknowledge the possibility of

"social coding". Contributions to software product advancement process are made in a distributed, collaborative effort by a virtual community. Software developers all over the world can participate in a similar software project, editing distinctive parts of the code and producing different branches in the project source tree. There are presently no expressed limits on a software team. “A software project may be developed by an ever changing set of software engineers, and a software engineer may contribute to a set of different software projects hosted in a remote server” (Hu et al., 2016). Social coding has to a large extent changed the style of software development activities. The social network of software engineers persistently interacts with the network of software projects. There have been a few social coding platforms that encourage software developers far and wide to contribute to a wide range of software projects together. Distributed development tools, like. Git, serve as the base standard of social coding platforms. In light of Git, the GitHub platform has pulled in numerous software developers to work together on huge number of open source projects. In GitHub, projects have transformed into repositories. Repositories house more information inside. As the number of GitHub users keep increasing, GitHub repositories keep growing, but user to user trust relationship is not being explored. This trust relationship can provide insight in recommender systems.

1.3 TRUST

It is an established fact that one of the significant parts of human relationship is trust. And this has made trust a multidisciplinary topic treated in both psychology, sociology and computer science alike. As a result of this, arriving at a general meaning for trust that touches each of these areas has always been a tedious task. There are numerous definition of trust. One of the definitions is that “trust is a measure of confidence that an entity will behave in an expected manner, despite the lack of ability to monitor or control the environment in which it operates” (Sherchan, 2013).

1.3.1 TRUST IN PSYCHOLOGY

From the area of psychology, one of the accepted definitions of trust is that found in Rousseau et al. in 1998. “Trust is considered to be a psychological state of the individual, where the trustor

risks being vulnerable to the trustee based on positive expectations of the trustee's intentions or behavior" (Sherchan, 2013). From this definition, trust is argued to have emotional, behavioural and cognitive implications.

1.3.2 TRUST IN SOCIOLOGY

Sociologists incline to emphasize the relational characteristics or social qualities of trust, and relational characteristics or social qualities, in most part are characterized by the level of aggregation (e.g., individual level, community level, population level, organizational level and societal level). One of the most reviewed statements in sociology famously proposes that trust for a trustee will be a matter of the trustee's perceived ability, integrity and act of giving and of the trustor's propensity to trust.

Sociology defines trust as a bet about the future possible actions of the trustee. For this bet to be considered trust, it must have some effects upon the action of the trustor (i.e the person who makes the bet). Trust in sociology as with psychology considers trust from two viewpoints: the societal and individual trust. At individual point, the vulnerability of the person who makes the bet (i.e. trustor) is the major factor that influences trust. At societal level, it is obvious that trust is a property of every social group. This is expressed as psychological state of every member of the group towards one another and this makes each member of the social group to act in a way that they expect other members of the group to be trustworthy and also that they should be trusted by others. This means that at societal level, social trust can be grouped into an institutional or system aspect of trust.

1.3.3 TRUST IN COMPUTER SCIENCE

Just like in the other areas, the definition of trust varies from researchers to researchers although trust is one of the widely used term in computer science and network security. Before some definitions, it is worth pointing out that in computer science, trust is broadly classified into two broad categories i.e. user trust and system trust. The concept of user trust is derived from the work done by Marsh in 1994 from psychology and sociology where he gave standard definition

as “subjective expectation an entity has about another's future behavior” (Marsh, 1994). Another definition could be found in Wikipedia which says that: “Trust is a particular level of the subjective probability with which an agent assesses another agent or group of agents will perform a particular action, both before he can monitor such action and in a context in which it affects his own action.” System trust on the other hand is an expectation that a system or device will behave faithfully in the manner of fulfilling its expected purpose. As a note, this thesis considers trust from the perspective of user trust in computer science. In the light of user trust, it can be implied that trust is inherently personalized. Taking online platforms like Amazon, inferring trust is based on feedback from users based on past interactions. In this sense also, trust is relational. The relationship between two members is strengthened as the two members continually interact with each other. If the outcome of this interactive experience turns out to be positive, trust evolves and increases, and decreases otherwise.

There are two fundamental sorts of trust in online systems: they are the direct trust and recommendation trust. Direct trust depends on direct experience of a member with the other party.

1.4 CHARACTERISTICS OF TRUST

1. Context Specific: Trust is context-specific in its scope. Trust context refers to the area in which the trust relationship exists. Examples are social networks, law enforcement and many others. To further explain the concept of context specific, ‘Killian’ trusts ‘Avil’ as his pilot but he doesn’t trust Avil to be his driver to drive him around. Therefore Avil is only trusted in the context of being a pilot.
2. Dynamic: Trust changes with time. It increases or decreases as new experiences are being gained during the period of the relationship or interaction. In some cases, trust can even decay to a level of distrust. Old experiences become obsolete and in most instances are irrelevant with time thus newer experiences are very necessary. In computer science, this is the most considered since the web of trust (WOT) keeps track of connections at every moment. Much research work has been done to model the dynamicity of trust.

3. Propagative: This is the most studied property of trust. The propagative property of trust is like the word of mouth propagation of information by humans. As a result of this propagative nature of trust, trust information can be passed down in the social network trust chain (trust chain is the network formed as trust is passed from one person to the other). Example, if Gori trusts Joseph, who in turn trusts Amanda whom Gori doesn't know, Gori can derive some extent of trust on Amanda based on how trusted joseph has found Amanda to be. Various trust models [Schillo et al. 2000; Mui et al. 2002; Sabater 2002; Yu et al. 2004] have used this property. Similarly, literature based on the FOAF (Friend-Of-A-Friend) topology are all based on the propagative nature of trust (Sherchan, 2013). This characteristics of trust should not be interpreted as transitivity.
4. Non-transitive: If Joseph trusts Gori, and Goris trust Amanda, this doesn't imply that Joseph trust Amanda. In trust, "transitivity implies propagation but the reverse is not true". In general terms, we can say that trust is not transitive. It is very unfortunate to say that this property of trust is always confused with the propagative property of trust.
5. Composable: When various network chains recommend diverse trust values to a member, the trustor needs to compose the trust information. This occurs probably because trust and distrust propagation along social chains permits a member in the network to compute trust to other members not directly connected to it. Tidal trust had to take this into consideration. Typically, models that make use of the composition feature also employ the propagative feature to compute trust values from several trust chains in order to make a trust decision. For instance, Joseph is recommended to Gori by several chains in her network. In this case, Gori needs to compose the trust data received from different chains to decide whether she can trust Joseph.
6. Subjective: This property of trust brings to view that trust computation is also personalized. The choice of the trustor determines to a large extent the value of trust being computed. For example, Gori gave a positive review about a book and Hillary always believes that Gori's reviews are always good, Hillary can go for the book based on the subjectivity of trust. On the other hand, Emeka doesn't find Gori's reviews to always

be good, then he might be sceptical in choosing the same book based on Gori's review although he might go for the book based on personal conviction.

7. Asymmetric: Asymmetry in trust can be seen as an advanced personalized trust characteristics. Bob may trust Gori more as compared to the trust Gori has for Bob. When one member in a social network is perceived to act in an untrustworthy manner, the other member of the network might be forced to reduce its own trust value of the trustor. Asymmetry might be caused by differences in members' beliefs, expectations and perceptions.
8. Event-sensitive: Because trust is built up over a very long period, care is always taken in order for it not to shatter under a high-impact event. "This aspect of trust has received even less attention in computer science" (Nepal et al 2010).

1.5 RECOMMENDER SYSTEMS

Recommender systems are subsystems that provide suggestions related to item(s) to user(s) who are making use of a particular system in order to ease the decision making process. These item(s) are generally used to refer to what the system recommends to the user(s). As a result of the overwhelming items, users are most likely to get confused on what to select, which item to add to their cart, which website to visit, which item to add to their wish-list and many others. Recommender systems are usually designed to be item specific and as such all the components of the systems like the user interface, the recommendation techniques, the algorithms and the user interaction are always customized to provide useful personalized suggestions of specific items to a user or a user group. Moreover, in some instances, the suggestions could be non personalized as can be seen in newspapers and top 10 selections of books etc. This type of non-personalized recommendation systems are not always of research interest since they are in general terms just like the normal web services rendered to all users using a platform. A case in point is a developer recommender system that recommends developers for a newly created project. Popular websites like Facebook use recommendation system to recommend friends of a friend; Amazon and eBay and most e-commerce websites use their recommender system to suggest items to be bought. Personalized recommendations are provided as a list ranked in order

of item preferences. This ranking is performed by predicting the suitable services, products or items a user might need based on the users preferences and constraints. The collection of the user(s) preferences could be done explicitly (e.g, user product rating, user friend rating, reviews etc.) or inferred based on user-system interaction (e.g., page navigation, clicks and time spent reviewing an item) as a sign of indirect preference for such an item ("Recommender system handbook," 2014). With the rise in e-commerce websites and new e-business services (item comparisons, annotated searches, service rendering platforms) the need to render recommendations from filtering the whole contents on a particular website is pressing. Recommender systems have shown positive solutions in dealing with information overload by pointing users towards newly created item lists that can be of interest to their current task/need.

1.6 PROBLEM STATEMENT

With the advent of social networking which has found its use in all sectors of life and human development, social coding platforms not excluded, part of the challenge is recommending possible projects of interest to users. Studying the social network structure that is developed as a result of developers collaboration while working on projects can give us an insight on generating a trust network which can assist in developing a recommender system algorithm that can recommend possible developers on any project being created on such platforms.

1.7 OBJECTIVE OF THE RESEARCH WORK

Although much research work has been done both in the academic and industrial area on social recommender systems, little attention has been paid to applying the concept of recommender systems in social coding platforms. The primary objective of this work is to review extensively existing trust-aware recommender algorithms and making proper documentation of them. This will lead to applying these algorithms on GitHub datasets to see if trust networks could give better recommendations than conventional recommender system techniques. The final objective will be to devise a hybrid or novel algorithm that can outperform the existing algorithms.

1.8 RESEARCH METHODOLOGY

The methods adopted to carry out this research work centres on exploration of information retrieval techniques in identification of similar projects. To achieve this, a natural language processing (NLP) technique called term frequency - inverse document frequency is applied in feature extraction from readme files and a similarity metric is adopted in computing similarity. The datasets are converted to graphs using a graph library and a new graph called a trust graph is generated. Trusted user experience levels and skill set levels are evaluated, which aids in the recommendation list. Word cloud and Pearson product-moment correlation is used for evaluation and experimentation.

1.9 SCOPE OF WORK

The scope of this work includes:

1. Applying an existing algorithm on a GitHub dataset;
2. Developing a trust inference algorithm; and
3. Applying the new algorithm to the GitHub dataset.

1.10 ORGANISATION OF WORK

Chapter Two presents an extensive literature review on trust inference algorithms and GitHub repository recommendation. Chapter Three presents the research methodology and implementation of work. Chapter Four highlights the experimentation performed and results obtained. Finally Chapter Five presents recommendations for future work and summary of work done in this thesis.

CHAPTER TWO
LITERATURE REVIEW

2.1 RECOMMENDER SYSTEM

As defined earlier, recommendation systems are software techniques and tools that give item suggestions to users who might be interested in such an item. This suggestions are geared towards helping the users in their various decision making processes while using a platform or system. Suggestions could be in the form of what to buy, books to read, what news to read, what places to visit, hotels to book and many others. Because of information overload balancing by recommender systems, the use of recommender systems has proven to be a useful means to personalize items in electronic commerce. Consequently, numerous techniques for generating recommendation have been proposed in the last few years in the academic field and many of them have also been deployed successfully in the commercial environment (Carrasco, 2012).

Recommendation systems for social networks are faced with three main challenges: first, they are faced with learning to rank the information from a user's friend. Secondly, other than the user direct friends, they have to discover other information sources. Finally, they have to decode information provided by the structure of the network and user interaction (e.g. trust and friend relationship) (Carrasco, 2012). When system users find out that the system has a form of recommendation, they tend to rely on the system a lot especially if the recommendations are always correct or close to being correct. Travel intermediaries (like wakanow.com) introduced travel recommender system in Nigeria, to increase their turnover, make more hotel rooms sales or even increase the number of tourist destination. But users basic motivation for going to the system (i.e. wakanow.com) is to find the cheapest flight or travel service, find the cheapest of the hotels or to find the best tourist places to visit during holidays. This means that there are various functions of a recommender system both to the users of a recommender system and to the owners of the systems.

2.1.1 FUNCTIONS OF RECOMMENDER SYSTEM

1. **Increased conversion rate:** From the service providers point of view, the elementary goal of a recommender system is to increase the number of conversion made by users from using the service (i.e. number of users that accept the recommendation and consume the item other than users that visits and browse through the information without consuming the item). This is the most important function of recommender systems from a commercial point of view. To achieve this goal, the recommender system tries to recommend items that are likely to be pleasing to user's needs. Also, the recommendations are made without pay and as such the user gets to notice the recommendation after making one or several consumptions. Non-commercial recommender systems achieve the same goal in most instances (Francesco, Lior, Bracha, & Paul B., 2014).
2. **Select more diverse items**: The use of a recommender system enables users to select diverse items that would have been extremely difficult to select without the help of RS. This is so because the service provider at each point in time wants to advertise all the items in their catalogue to each user. Hence, recommenders help the users to make choices even when the items are not of popular demand but suites the taste of the user at that moment. A quick example is the Amazon recommender that provides recommendations based on content filtering and demography of the user. As such, users are always provided with items that suite their age, sex, contents of previous choices and many others.
3. **Better grasp of user needs:** Another important role of a recommender system is getting a better description of user preferences, which could either be collected explicitly or system predicted. The collected information can then be used for intelligent decision making in order to achieve other organisational goals and also provide better service to the user.
4. Other importance of RS can be narrowed down to two broad areas i.e. value for the customer and value for the provider (Jannach, Dortmund, & Friedrich, 2013).

Table 2.1: Other functions of recommender system

Other functions of recommender systems.	
VALUES FOR CUSTOMER	VALUES FOR PROVIDER
Finding things of interest	Personalized service provision to the customer
Narrowing down of set of choices	Increased click through rates (clicks without conversion)
Explore and discover new items. And provision of entertainment	Better chance for promotions and persuasion.

2.1.2 ELEMENTARY STRUCTURE OF RECOMMENDER SYSTEM

The elementary component of a recommender system consists of a recommendation component which could be a software component or an algorithm with tries to make predictions based on the available input.

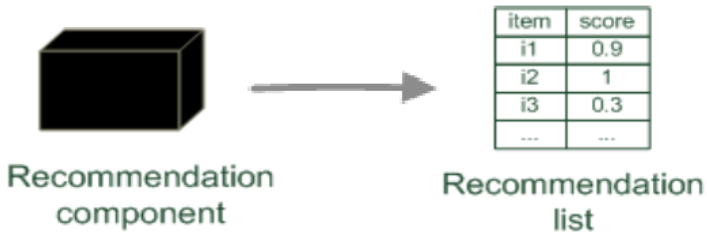

Figure 2.1: Elementary Recommender System

In this elementary case, the system does not consider any other parameter in making its decision. As stated earlier, this phase of recommender system is not considered in most research as it is not personalized.

2.1.3 PERSONALIZED RECOMMENDATION

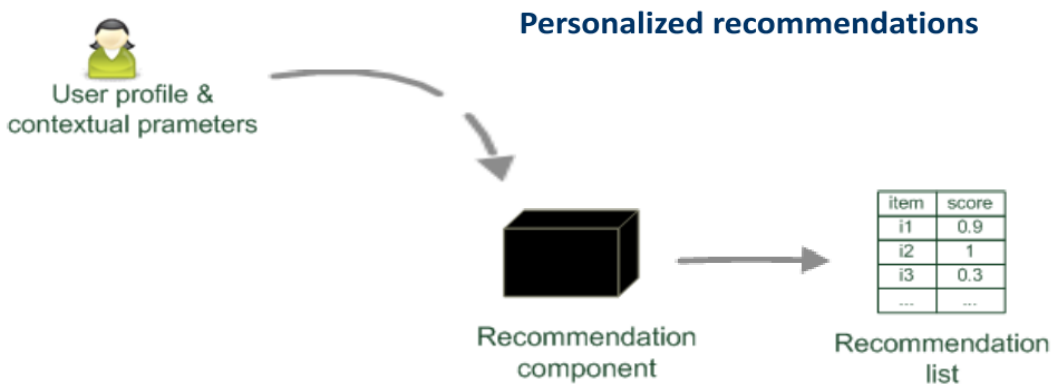

Figure 2.2: Personalized Recommender System

One of the generic approaches to designing of a recommendation system is the technique of users personalizing what they want to see whenever they use a platform. The users filter the entire information based on their contextual parameters and taste. The result of this search filtering gives the users a taste of what they want but the challenge this approaches faces is that users are also faced with a very large amount of search results. Apart from the two above listed approaches in recommendation system design, there are four most efficiently used paradigms in recommender systems design:

1. Content based recommendation
2. Collaborative filtering recommendation
3. Hybrid recommendation
4. Knowledge based recommendation.

2.1.4 COLLABORATIVE FILTERING RECOMMENDATION

Collaborative filtering is one of the most popularly used techniques for recommenders systems that anchors its fundamental principle of recommendation and prediction on the activities and ratings of other users in the system. In this method, “users’ opinions can be selected and

aggregated in such a way as to provide a reasonable prediction of the active user's preference" (Ekstrand, 2011). In essence, if a group of users accept the relevance of an item, they might also agree on the relevance of other items.

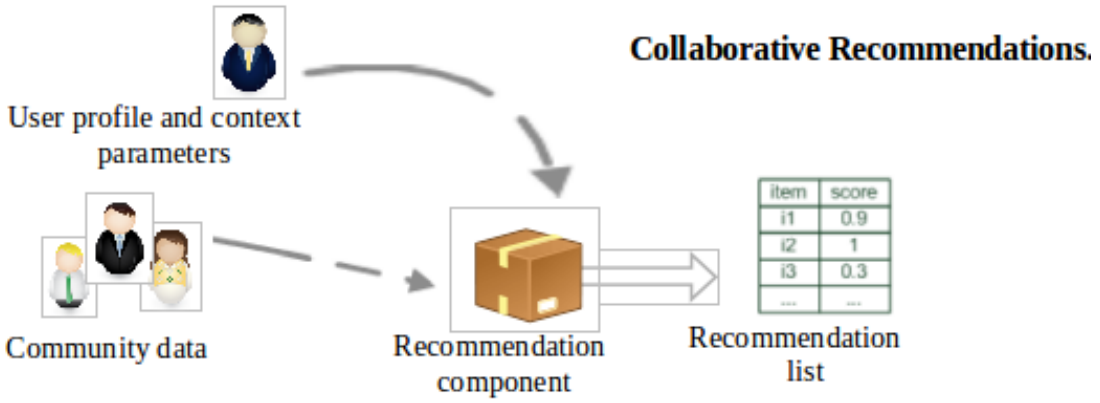

Figure 2.3: Collaborative Filtering Recommender System

A lot of collaborative filtering algorithms in use operate by generating prediction of user-to-item preferences, and ordering the items according to predicted user preferences leading to recommendation. The prediction and the ratings might be on the same scale but in most cases, the prediction is done on different scale and its sole importance is for item ranking. As such, the prediction value makes more sense to a provider but less sense to the users as they are concerned with ranked items that meets their needs. According to Ekstrand (2011), "the recommendation task can be seen as information retrieval problem in which the domain of items (the corpus) is queried with the user's preference profile". A literature review on collaborative filtering would be more appreciated if a little discourse is made on baseline predictor models. These models are good for non-personalized baselines and when used with sophisticated algorithms, they provide personalized baselines good for user rating predictions. This baseline prediction for user X and item i can be represented by $b_{x,i}$. The simplest baseline method is predicting the average rating amongst all ratings in the platform: $b_{x,i} = \mu$ (μ is the overall average

rating). With further modification, baseline can be modified by computing a predicted value for the average rating by the user X or average predicted value for item "i" :

$$b_{x,i} = \bar{r_x} \text{ or } b_{x,i} = \bar{r_i} \quad (1)$$

General baseline predictor is of this form

$$b_{x,i} = \mu + b_x + b_i \quad (2)$$

b_x and b_i are user baseline predictor and item baseline predictor respectively. Additional enhancement could cause the baseline predictor ratings be closer to global mean with user or item having few ratings. Other baseline predictors exists but are not discussed in this work.

2.1.4.1 METHODS OF IMPLEMENTING COLLABORATIVE FILTERING

1. **User-User K-neighbourhood model:** This model considers the similarity between a user "i" and all other users, then gets the k-nearest neighbours based on this similarity measure and computes the predicted rating for "i" using a weighted summation of the k-nearest neighbor ratings (Samaneh, Mehta, & -Rahul, 2013.).

$$P_{u,r} = \bar{r_u} + \frac{\sum_{u\prime \in N} s(u,u\prime)(r_{u\prime j} - \bar{r_{u\prime}})}{\sum_{u\prime \in N} |s(u,u\prime)|} \quad (3)$$

   As user database grows, the model suffers scalability. Hence other approaches were also considered.

2. **Item-Item k-neighbourhood model:** Unlike the user-user k-neighbour, this model pays attention on the items and not the users. In this model the application of baseline predictor is very important for normalizing the matrix on which similarities check is carried out.

$$r_{x,i} = b_{xi} + \frac{\sum_{j\prime \in N(i,x)} s_{ij} \cdot (r_{xj} - b_{xj})}{\sum_{j\prime \in N(i,x)} s_{ij}} \quad (4)$$
$$where\, b_{xi} = \mu + b_x + b_i$$

   Other methods include the use of Stochastic gradient descent, regularized stochastic gradient, alternative least square. More sophisticated methods includes applying deep

learning algorithms for collaborative filtering, applying neural networks and other machine learning algorithms for making predictions and recommendations.

2.1.5 CONTENT-BASED RECOMMENDER SYSTEMS

Content-based recommendation systems learn from user profiles and item features descriptions as previously rated by a user. This learned information is used to model the user profile and structure the user interests and adapt them for recommendation for new interesting items. The recommendation is done simply by matching up the attributes of the content against the attributes of the user profile. Content based recommender systems are linked to information retrieval and filtering research that tries to determine similar items. One weakness of this approach in recommendation materializes when there is little or no description available about the item and the accuracy of recommendation depends on these descriptions or annotations. Another weakness also appears when the recommendation system always brings up items that the user is already conversant with, as such content based recommenders do not explore interests of users besides those expressed in their rating record (Adomavicius & Tuzhilin, 2005).

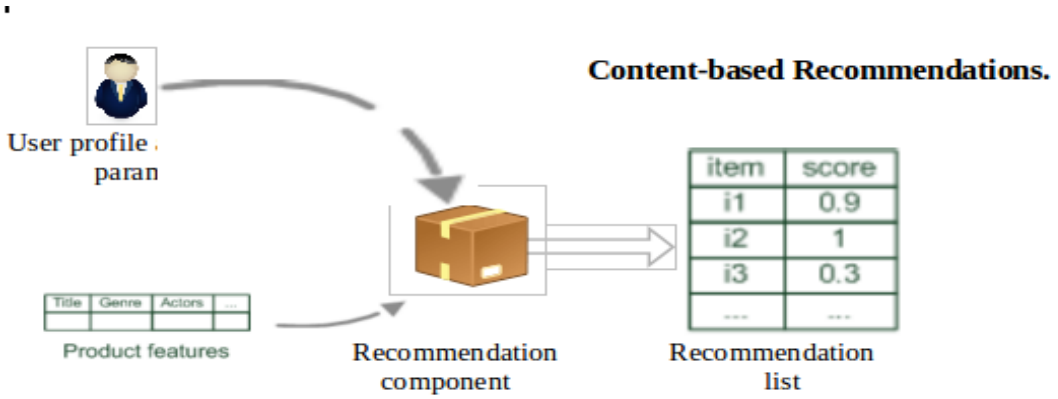

Figure 2.4: Content Based Recommendation System

2.1.6 KNOWLEDGE-BASED RECOMMENDATION SYSTEMS

Knowledge-based recommendation systems get their basic idea from the expert system field where systems become 'intelligent' based on encoded knowledge. This type of recommendation systems use knowledge that is encoded on them by human experts on input data to generate recommendations, thereby giving advice to users on actions or decisions to make. The user profile could be of any structured form provided it provides support for such inference. The knowledge used can also take any form that can also support the targeted prediction(Burke, 2002).

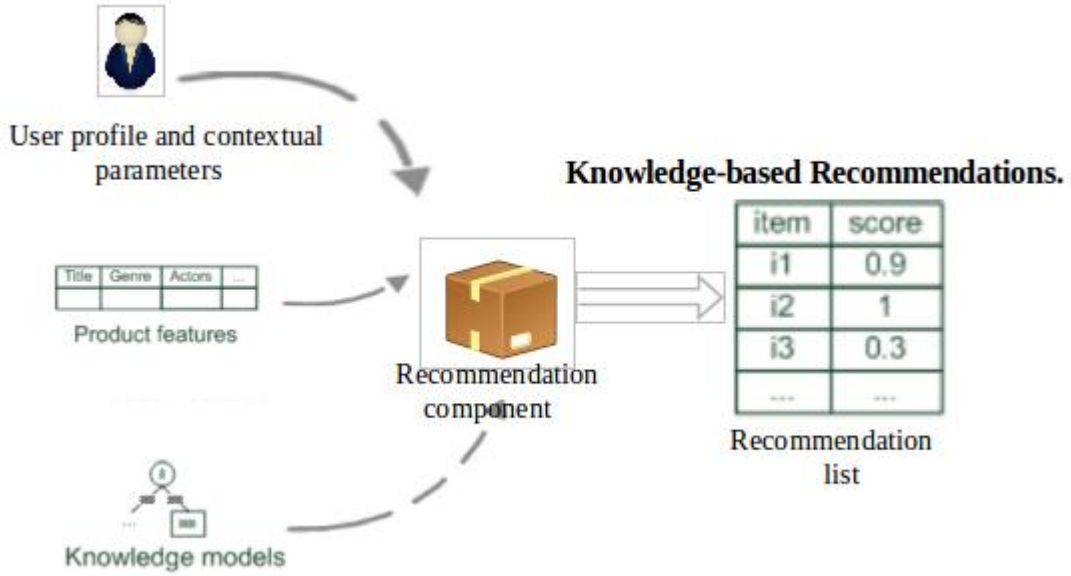

Figure 2.5: Knowledge-based Recommender System

One class of this system uses case-based reasoning and this class has greater implementation in the commercial world than the academic or research area. Entree (Burke, 2003.), a restaurant recommender system recommends new restaurants to users based on similar restaurants known or liked by the user. As a knowledge based system, users are allowed to navigate the system by stating their preferences, hence redefining their filter criteria. In this case, the users get recommendations based on their peculiar knowledge domain and the activities of others are not taken into consideration (Bouraga, Jureta, Faulkner, & Herssens, 2014; Burke, 2003). According to Bouraga and Jureta (2014), knowledge-based systems can be used to solve the problem of cold start since no large data set is required, the grey sheep problem of new items is also avoided

and the knowledge domain is noise free. But the biggest challenge is knowledge representation which might be solved by knowledge domain and the involvement of expertise in knowledge representation. More detailed examples and evaluation of these types of systems can be found in Ricci (2012) .

2.1.7 HYBRID RECOMMENDATION SYSTEMS

Hybrid recommendation systems make a combination of one or several recommendation techniques to make inferences or predictions. This takes away many of the drawbacks that come with using single technique. The most common approach is the combination of collaborative filtering with other techniques to resolve new user/item ramp-up challenge. Some algorithms that adopt this approach may not be classified as hybrid because their collaborative and content-based predictions do not depend on each other (Mark et al., 1999) although some researchers like Burke (2003) consider such approach to be a weighted classification of a hybrid recommender system.

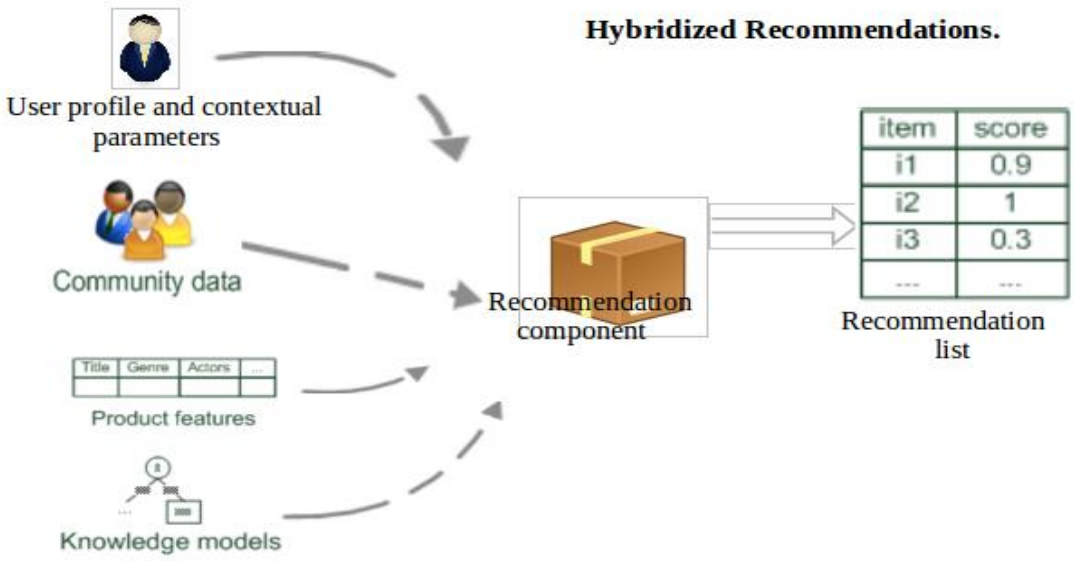

Figure 2.6: Hybrid Recommender System

2.1.7.1 CLASSIFICATION OF HYBRID RECOMMENDATION SYSTEMS

Burke in 2002 suggested a classification for hybridized recommender algorithms. He classified them into:

1. **Weighted hybrid:** In this hybrid recommender, the score for the prediction is computed from results of all available recommender techniques available in the system. P-Tango system is an example of such recommenders. All parts of the system are brought together to share in the recommendation process in a straight-forward manner.
2. **Switching hybrid:** In this hybrid system, a switch is made from one recommender technique to another based on some criterion. Switching hybrid is always sensitive to weakness and strengths of constituent recommenders and this introduces another complexity since the criteria for switching must be determined.
3. **Mixed hybrid:** Mixed hybrid is used where the recommendations from more than one technique are to be presented together. It avoids the new item start-up problem. Some implementations of mixed hybrid is PickAFlick by Burke et al. (2000), ProfBuilder by Wasfi (1999) and many others not listed in this work.
4. **Feature Combination hybrid:** Treating collaborative information simply as additional feature data linked with each example and using content-based techniques over the augmented data is the core idea behind the operation of Feature Combination hybrid.
5. **Cascade hybrid:** This is a staged recommendation technique in which one technique is used to get a coarse prediction, the next refines the output until an optimal prediction is made at last. Using cascade hybrid, the system is always prevented from using lower-priority technique on items well differentiated by the first.
6. **Meta-level hybrid:** For this type of hybrid recommender system, contributing and actual recommenders exist but the former completely replaces the data for the latter, not just part of it (Jae-wook, 2005).

Going by the context-specific property of trust in Chapter One, a new approach in Hybrid Recommender Systems has evolved and that is trust-ware recommender systems. The sections below discusses trust aware recommender systems, trust network, trust representation and the

state of the art in trust-aware recommenders for social coding platforms as this covers the general scope of this work.

2.2 TRUST REPRESENTATION AND TRUST METRIC

Trust as one of the integral players in life and our society and in our day-to-day interactions both while using the social media and in physical relationship. It is dependent on the individual decision and behaviour either to trust or to distrust. Quantification of trust value is generally done in ranges <a, b>, where a, b (a<b) are integer or real numbers. This is best suited for verbal trust representation.

The simplest trust metric has three inputs: a directed graph, a designated root node of the trust usually referred to as source, and a target node. This forms the basis of other trust metrics except in some cases where bipartite graphs are considered instead of a directed graph. The choice is always to determine whether the target node is trustworthy. An edge in the graph from source (s) to target (t) is an indication of belief that 's' believes that 't' is trustworthy. The simplest of trust metrics makes evaluation of trust by considering whether target "t" is reachable from source 's'. In the absence of a path from source down to sink, there is no reason to believe that t is to be trusted in view of available data. Noting that "the simplest trust metric is also the weakest with respect to attacks. If an attacker is able to generate an edge from any node reachable from the seed to a node under his control, then he can cause arbitrary nodes to be accepted. As the size of the reachable subgraph increases, the risk of any such attack increases as well . All trust metrics include the three inputs described above: the trust graph, the seed node, or trust root, and the target. Some trust metrics add more detail to these inputs. Trust edges may contain some conditions or restrictions, for example 't' is himself trustworthy, but cannot be trusted to vouch for other nodes. In addition, the target may be a richer assertion than merely whether a certain node is trustworthy. For example, edges may additionally identify a maximum dollar value, and the target may be an assertion that the target node can be trusted with a transaction of some dollar value" (Levien, 2004) .

In Netrvalova, Safarik, and Republic (2012), the model represented trust values in form of continous interval <0,1> , where 0 represented complete distrust and 1 represented complete blind trust. Indecisiveness was found in the middle while other trust values were found in between these ranges. Interpersonal trust representation which has been discussed earlier can also be represented in Netrvalova's model considering "n" subjects represented as set S={s1,s2,…..s_n}. The personal trust between each subject is introduced as

$$t_{ij} = t(s_i, s_j),\ t_{ij} \in \langle 0,1 \rangle,\ \text{where } i,j = 1,\ldots,n,\quad i \neq j,$$
$$\text{and} \sum_{j=1, j\neq i}^{n} t_{ij} = 1 \qquad (1)$$

(5)

supposing that both t_{ij} , and t_{ji}, exist. A directed weighted graph is used for such personal trust representation in the social network where nodes represents\ subjects and edges represent trust relations between the subjects and the weights are the trust values. Another representation that follows this approach too is the phenomenal trust representation by Netrvalova, Safarik, and Republic (2009), which defines trust subject to phenomenon. Subjects are consider as set S of n elements and m exclusive products of any kind is represented as set of P elements which constitute the phenomenon. Trust of subject "x_i" to product "p_k" is denoted as

$$t_i^k = t(s_i, p_k),\ t_i^k \in \langle 0,1 \rangle, \qquad (6)$$

where: "i" = 1….n and k=1,…..m. Phenomenal trust matrix was used to represent the phenomenal trust. In a decentralized environment where there is no centralized quality control measure, assuring the quality of content created by a user becomes a daunting task and as such, the best strategy is to allow the users to give a rating within a range <0,1> about the user and content they can trust. These trust statements and their equivalent values can be aggregated from the trust network to represent the relationship of the users.

2.3 TRUST MODELS

SUNNY, a model that gives a direct probabilistic interpretation to confidence in social trust networking is one of the first models that considered confidence level in a trust network (Kuter & Golbeck, 2005, 2010; Ziegler & Golbeck, 2015). The model first gives a formal representation of a trust network and produces a Bayesian Network appropriate for use in probabilistic reasoning. The model uses similarity measures which are calculated over the previous ratings of users in the trust network to produce a probabilistic model for the Bayesian network. On the confidence values of each node in the Bayesian network in the sink node, SUNNY computes estimates of the lower and upper bounds. These estimates enable SUNNY to determine whether to include a leaf node in a Bayesian Network in the final trust or to leave it. SUNNY, with the help of some probabilistic logic sampling techniques, calculates confidence value of a source to a sink node in a B-network. The upper and lower bound computed by SAMPLE-BOUNDS (a subroutine in the SUNNY algorithm) are used first by invoking the subroutine with no decisions made and this produces upper and lower bounds on the source's confidence and this is usually the maximum and minimum values respectively. These bounds are used in a hill climbing search to actually finalize if a leaf node will be excluded or included. At the end of this decision, the model uses a backward search from the leaf nodes back to the source to compute the trust value for the source. "At each iteration during this search, the trust value of a node is computed based on the trust values between that node and its immediate parent and the trust values of the parents in the sink node that are already computed in the search" (Kutter et al 2015). Dorri Nogoorani and Jalili (2012) in Uncertainty in probabilistic trust models, analyzed the uncertainty of probabilistic trust models and quantified them in the form of confidence intervals. Beta and HMM (Hidden Markov Model) trust models were used to achieve their uncertainty factors. The confidence intervals were calculated using the bootstrapping method from (Efron & Tibshirani, 1986)

Another trust model is the RN-trust model proposed in Taherian, Amini, and Jalili (2008). The basic concept of this model is the use of a resistive network idea in simulating a trust network. To achieve this aim, Taherian et al. modelled a trust relationship between two individuals in the network by a resistor such that the value of the resistor between two persons is inversely

proportional to the trust value. In doing this, the trust network must first be changed to a resistive network. A mapping function is used in mapping a trust network to a resistive network and this is done by taking a trust label as input and giving an equivalent resistance as output. Current flow in this resistive network is likened to the trust relation from one user to another. If no resistor exists between two users, the model argued that both of them complete trust. If there was a resistor between two users x and y, then the amount of current flows from x to y decreases. Hence, "the trust values have reverse relation with the values of the resistors. The more the trust values, the less the values of the resistors. If there is not full trust between two nodes in the trust network, no resistor might be placed between corresponding nodes in the resistive network. On the other hand, if we have low trust between two nodes in the trust network, we must consider a resistor with relatively big resistance between the related two nodes in the corresponding resistive network" (Taherian, Amini, & Jalili, 2008). This model also considered the trust network to be a direct graph as with Golbeck (2008). To achieve the asymmetric relation property of trust, an ideal diode and the logarithmic function is also introduced to switch the values between the intervals [0,1]. A general summary on the properties of this model includes the fact that all resistors must be >= 0. The inferred trust between two users cannot be greater than 1. In the absence of any path between two nodes, the trust inferred must be equal to 0. Assuming all the paths between two nodes passes through a node, the value of trust in the central node from which the paths pass must be greater than or equal to the trust value between the two users. Increasing the trust value between two corresponding users, must not decrease the inferred trust value between the two nodes and vice versa. Creating a new path from a source to a sink in the absence of an intersection with older paths, the trust value between the source and sink must be increased. The aggregated time complexity for this algorithm is said to be $O(V^3)$.

Trust inference in on-line social networks has also been explored by Papaoikonomou, Kardara and Varvarigou, 2015). Their work focuses on predicting the level of trust between two nodes using the ratings they have given for a set of items and also proposing an edge sign prediction using a semi-supervised approach and the concept of transfer learning from (Raina, Battle, Lee, Packer, & Ng, 2008). The approach used in the work was influenced by the advances in deep

learning (especially the Restricted Boltzmann Machine and the Autoencoder which are the building architecture of the model) where a number of layers as with neural networks is used to arrive at a better representation of the input. The Restricted Boltzmann Machine is applied on the ratings by the users in order to give each of the user ratings a binary low-dimensional code indicative of preferences. Autoencoders are used on the user codes to perform the actual classification task. In its simplest form, an RBM consists of binary units that are arranged in a set of layers, called the “visible” and “hidden” layers. Observed data are fed to the visible layer, while the units in the hidden layer takes note of statistical regularities. In doing so, an RBM approximates the input data distribution and can be tagged a generative model. The binary visible layer of the RBM was replaced with a layer of conditional multinomial units in order to permit the RBM to consume rating data. An autoencoder neural network is simply an unsupervised learning algorithm that tries to approximate the identity function by applying non-linear transformations to the input. The autoencoder architecture determines the effect it exerts on the data. For example, autoencoders with fewer hidden units than visible can be regarded as a non-linear dimensionality reduction technique.

Nordheimer, Schulze, & Veit, 2010) in trustworthiness in networks: A simulation approach for approximating local trust and distrust values assumed that a trust network provides a simple system for rating in which users are allowed to make only positive trust statements and they came up with MoCaTrust which is an extension of SimTrust. The assigned trust value by a trusted user represents a subjective probability that a trusted user will behave in a particular expected manner. These values are exclusively between the interval [0,1]. This concept is to interpret trust as the connection probability between two nodes (that is, the probability that a start node connects to the target node in a network). In the absence of a direct trust value, the model argues that it can compute trust value as the total connection probability. It describes the probability in view that there is an underlying indirect connection between the nodes. Monte Carlo approximation method is then introduced to obtaine the required solution by repetitions of random tests and by carrying out statistical analysis of the obtained results. The required solution is a state of trust network which is determined whether the start node connects to the target node in this state. An estimator

for the trust value is obtained simply by repeated independent experiments and the computing of the mean value of the function. Creating a network state requires generating a (0,1) uniform distributed random number as with Monte Carlo. The generated number is then compared with the respective trust value. There is a failure and a removal of edge from the if the random number is greater than the trust value. The reverse is the case if the randomly generated number is lower than the trust value. This is continued for a number of repetitions alongside with the generated function φ(y).

$$\hat{Pr}(v_i, v_j) = \frac{1}{N}\sum_{i=1}^{N} \phi\,(\bar{y}_i) \qquad (7)$$

Raph Levien in his PhD thesis in 2004 devised an Attack Resistant trust metric which is a quantitative framework for evaluating the attack resistance of trust metrics.

Massa Paolo in Trust-aware recommender systems proposed a trust metric that is able to propagate trust over the trust network and to estimate a trust weight that could be used instead of the popularly used similarity weight. (Massa & Avesani, 2007, 2009) exploited trust information explicitly given by the users, as with many other literatures. In context of recommender systems, the expressed trustworthy level expressed by individuals could be liked to considering the importance and relevance attached to the ratings provided by a given user. TaRS has two input datasets which include the ratings matrix(ratings of items given by users) and trust matrix (all community trust statements).

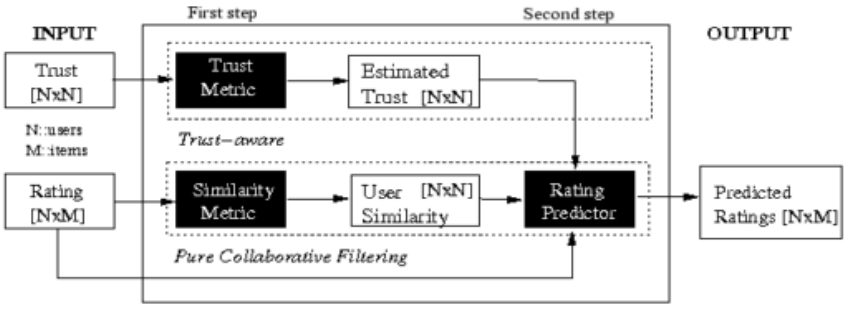

Figure 2.7: Architecture of Trust-Aware Recommender Systems

The trust matrix extends the traditional collaborative filtering systems. Going with normal collaborative filtering, the other two steps remains unchanged: neighbourhood selection using similarity metrics and rating prediction. From the diagram above, Massa et al. introduced a new dimension to neighbourhood selection and how weights are computed. The weights can be computed from the user similarity assessment for the case of traditional collaborative filtering or with the use of trust metric. TaRS uses these two metric modules to produce estimated trust matrix and the user similarity matrix: row "i" contains the neighbours of a user "i" and the column "j" represents a weight in interval [0,1]. The trust metric module in this model computes local and global trust. MoleTrust was used for computation of local trust whereas PageRank was used for the global trust metric.

Abderrahim and Benslimane (2015) in "Towards improving recommend system: A social trust-aware approach" used quality of recommendations given by peers and quality of social links with peers in user-to-user relationships to derive the degree of trustworthiness between users. The user's quality of connection is modelled after the concept of friend-of-a-friend (FOAF) or web of trust in the case that it allows users to create connect statements about themselves and who they know or wish to connect to. This level of trustworthiness is represented in a scale of 0(weak affinity) to 1 (for Strong Affinity). To infer and compute trust for users not directly connected, they applied tidal trust model. Transitive affinity values were inferred using

$$U_{u,v}^{LoA} = \frac{\sum_{p=0}^{n} \begin{cases} \left(U_{p,v}^{LoA} * U_{u,p}^{LoA}\right) & if\ U_{u,p}^{LoA} \geq U_{p,v}^{LoA} \\ \left(U_{u,p}^{LoA}\right)^2 & if\ U_{u,p}^{LoA} < U_{p,v}^{LoA} \end{cases}}{\sum_{p=0}^{n} U_{u,p}^{LoA}} \quad . \qquad (8)$$

The quality of a user's recommendation which denoted the degree of satisfaction of v regarding recommendations of u allows detecting useful recommenders that are known for making reliable recommendations. This was calculated by dividing the total number of reliable recommendations

that user “u” provides to a user “v” by the total number of recommendations that “u” has given on the trust network.

$$U_{u,v}^{QoR} = \frac{RR_{u,v}}{TR_u} \tag{9}$$

The user’s quality of trust which denotes the degree of trustworthiness between users “u” and “v” can be calculated using their social connections and interactions. It is measured with. The model combines both user based and web service based collaborative filtering to calculate trust value from user's and web services point of view.

$$U_{u,v}^{QoT} = \gamma \times U_{u,v}^{QoR} + (1-\gamma) \times U_{u,v}^{QoC} \tag{10}$$

The Advogato max-flow trust metric proposed by Levien and Aiken is another trust metric which is introduced to discover how users are trusted among themselves in an on-line community. One centralized community server is used in computing the trust and considered relative to a seed of users that enjoy supreme trust. This metric can also be applied to arbitrary agents, which may require computing personalized lists of trusted peers. Local trust metrics inquire into a trust seed and compute sets of agents trusted by members of the trust seed. In the case of Advogato, the input is supposed to be equal to the total members as well as a subset of the complete set of users known as the seed. The output is a characteristic function supposed to map each member to a Boolean value which signifies his trustworthiness. Ziegler and Golbeck’s, (2015) AppleSeed trust metric is a converse of Advogato. Advogato was inspired by maximum network flow computation, AppleSeed was motivated by spreading activation models which were first proposed to use semantic memory to simulate human comprehension. Some assumptions in terms of searches in contextual network graphs, trust propagation, spreading factor rank normalization, backward trust propagation and nonlinear trust normalization require modifications to tailor them for local group trust metrics. Having made the modifications, AppleSeed allows ranking of agents just like Advogato but this values are assigned in Boolean values indicating presence or absence of trust. Nodes are accessed only when they are reached by the energy flow. Trust ranks that correspond to energy in Advogato are initialized to 0 and any unknown node is also initialized to

0. “Incoming flow for x is hence determined by all flow that predecessors “p” distribute along edges (p, x)”.

Tidal trust is another very popular algorithm for trust propagation in networks having continuous rating systems. In her PhD thesis, Golbeck made an extension of the semantic web Friend-of-a-Friend (a system where users on a scale of [0,10] annotate the friends they trust) obtained direct trust explicitly by previously building a Web-of-Trust. Her propagation algorithm followed some of the properties of trust and this gave rise to some of her findings;

- The more trusted a neighbour is, the more coincidence in the trust ratings of the target user and the neighbour.
- (b) Shorter paths are more desirable than longer ones. In order for the algorithm to avoid a maximum length of the paths, she incorporated a variable path length. The inferred trust rating is given by

$$t_{is} = \frac{\sum_{j \in adj(i) | t_{ij} \geq max} t_{ij} t_{js}}{\sum_{j \in adj(i) | t_{ij} \geq max} t_{ij}} \quad (11)$$

For this formula to give the best result, the path to be followed from source to sink must be decided on and this is set as the maximum trust threshold. This threshold is used to discard edges with lower trust values. Adjacent nodes to the source store the ratings the source assigned to them and the neighbour records the maximum of strength path leading to it. The depth of the path from the source to the sink found first is recorded as the maximum allowable depth until any path with lower depth is found since it takes a BFS (breath first search). At the completion of the search, the trust threshold is established by taking the maximum strength of the trust paths leading to the sink.

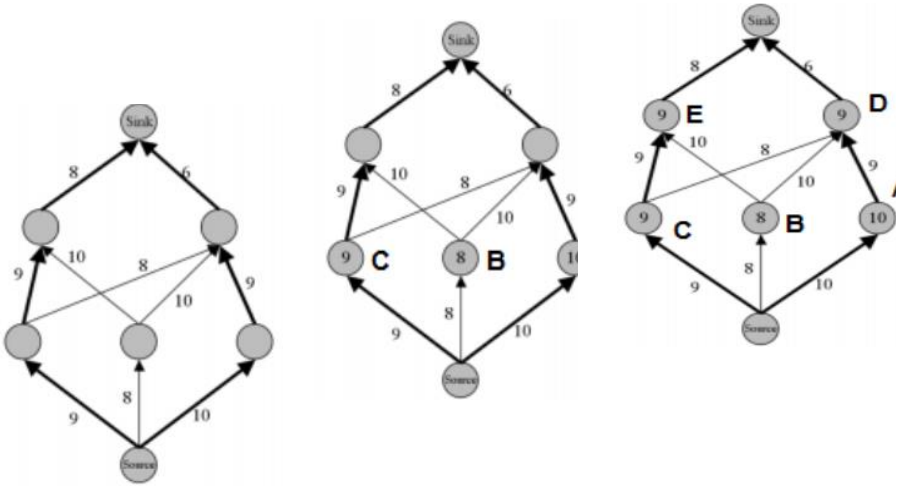

Figure 2.8: Tidal Trust Graph

The figure above is a simple graph of users with trust values represented as edge weights. Tidal trust has two main parts;

- Finding a path to the sink from the source while rating the nodes on the way to the sink; and
- Aggregating the trust values backwards towards the source once the sink has been found.

After the level 0 rating (i.e. rating from the source to its neighbours), the nodes in level 1 rates the ones in level 2 but the method of their ratings is quite different from that of level 0. Every node from level 1 assigns its rating to the level 2 nodes, and the rating value is the minimum of his rating and the his trust towards that neighbour. If any node has more than one predecessor in level 1, rating value changes to the maximum of the ratings that his predecessors gave to it. The process can be summarized to be

```
ratings ← min(rating,DirecttTustValue)
if pred > 1
    rating ← max(ratings)
```

This continues until the sink is found and a threshold value (“the maximum value of the nodes in the last level of the graph that are connected to the sink and have some trust towards the sink”) is set. At the determination of the threshold, every node apart from the ones having direct trust value to the sink, calculates its trust value towards the sink with the formula above where max is the threshold value. This formula is a weighted average of trust between them and their neighbours having trust values towards the sink and trust that their neighbours have towards the sink.

Mole trust is also similar to tidal trust. The only difference is the determination pattern of the threshold and search pattern for the sink. Mole trust doesn’t stop looking for the sink once the threshold value is found, but it continues further by exploring some paths until a particular depth d is reached. At that point it then calculates its trust value using the weighted average manner but with a different aggregation pattern.

Table 2.2: Summary of some common trust algorithms

Paper	Web of trust	Rating-based similarity	Profile-based similarity	User-item rating	Implicit trust value	Explicit trust value
Massa and Avesani 2004	X	X			X	
Massa and Avesani 2007	X				X	
Avesani et al 2004	X	X				X
Weng et al 2008	X					X
Ziegler and Golbeck 2007			X	X		X
Golbeck 2006			X	X		X

Paper	Web of trust	Rating-based similarity	Profile-based similarity	User-item rating	Implicit trust value	Explicit trust value
Golbeck 2005	X					
Victor et al 2008b	X					X

2.4 STATE OF THE ART ON RECOMMENDATION OF REPOSITORIES ON GitHub

In Repository Recommendation on GigHub for Requirements Elicitation Reuse, Lisette et al. (2015) proposed the creation of project profiles that have useful attributes for requirement engineering and this can help achieve a better recommendation of projects. Using readme file perspective, techniques finding meaningful assets in texts must be carefully designed, bearing in mind the requirements the engineer needs (Lisette et al., 2015). Some of the issues introduced in the work are that readme files are not categorized in GitHub repositories. A readme may contain information about features of the software, requirements for use, and installation guide. The second issue introduced in the work is that rating of GitHub projects are done at user end function not by their perspectives separately. They also proposed a natural language processing (NLP) based approach for recommendation. The steps included are retrieval activity, the filter activity, discovering activity and preference activity. The retrieval activity was used to automatically extract readme files from projects by passing queries using the GitHub API. This was called corpus R. The filter activity used the corpus R using the POS-tagging techniques to transform unstructured corpus R into nouns, proper nouns and other important assets, and called corpus T. The discovering activity also uses another NLP-processing approach to unveil frequent words that may attract user attention as a result of no user preferences for readme texts. Proper nouns from corpus T were used to form corpus NP. The frequency of words was calculated using TF-IDF (term frequency-inverse document frequency) weighting and a word cloud visualization technique was applied to display the data. The preference activity was aimed at getting user preferences, and ‘Zillow’, which is a real estate term, was found in 61 of the readme texts. This they called it corpus S. The clustering activity organizes the corpus S for recommendation. K-means and K-

medoids algorithm was used to generate clusters of projects in similar groups. This was done in collaboration with the silhouette approach for finding similarity.

Georgios and Claudia in Matching GitHub developer profiles to job advertisements proposed a pipeline that automates the process of using GitHub developer profiles to suggest matching job advertisements to developers based on signals from their activities on GitHub (Hauff & Gousios, 2015). The aim was to reduce the task of employers manually going through the profiles of employees in this case developers. The work was mainly concerned in translating both the developer profiles and available job advertisements into the Linked Open Data (LOD) space. Applying available algorithms, the setup also provides a means of determining the similarity between a job advertisement and a GitHub developer profile. The general pipeline includes three main components, which are extraction of concepts, weighting of concepts and matching. The extraction of concept uses the DBPedia Ontology, which is made up of two NLP processes, named entity recognition (NER) and name entity disambiguation (NED). The extracted advertisement concepts and the developer profiles are converted into weighted vectors W_j and W_d. If a concept from the developer concept is found in the extracted advertisement concepts, the weight is converted to a value above zero. Since not all concepts are important, TF-IDF is used, which gives low weight to concepts that have presence in many documents and gives high value to concepts that rarely appear across the entire corpus of documents. These weights are then computed separately as corpus for developer profiles and corpus for job advertisements. Cosine similarity, which is one of the widely used vector space models for determining similarity, is employed at this stage to determine the similarity between the devised job and the developer profile vectors. Each developer can then be ranked based on similarity on each of the job vectors.

CHAPTER THREE
RESEARCH METHODOLOGY

3.1 GitHub API EXPLORATION FOR DATA EXTRACTION

Like many social web platforms, GitHub's developer website offers sound documentation on its APIs, example code on how to use the APIs and the terms of service guiding the use of the APIs and much more. The APIs provide users with more or less everything that would be needed to build a rich user experience without any shortage of possible applications which could be built with the APIs not excluding applications like GitHub itself. Although there are a plethora of these APIs, this work focuses only on the few APIs calls needed for data retrieval and collection of data for creating a trust network that associates developers to developers and developers to projects based on their collaboration behaviour on GitHub. The most important primitives for GitHub are users and projects. A user is expected to have one or more code repositories that he or she created or have been forked from another user. As part of the activities of users on GitHub, a forked project could turn out to be a completely different project either hosted on GitHub or elsewhere. Apart from forking a project and creating a project, a user likewise can also bookmark or star projects and turn out to be *stargazers* of the project. By being a stargazer, you are signifying interest in the project and just like bookmarking a webpage, it appears in a bookmark list as a reminder for you to be aware of what is happening to that project. In addition to forking a project, users can also contribute to a forked project by committing their changes to their own local repository and sending a flag to the owner called the *head-repository* to accept the changes; this is referred to as a *pull-request.* Once a project is created, the author of the project creates a list of project members as those who have permission to make changes to the repository. These members are trusted developers whom the author can recommend to other projects. These project members also could be added to be contributors to the project based on their ongoing activities like accepted pull-requests, commit issues raised, commits and many other of activities. Throughout the remainder of this project, our focus is going to be primarily on members

who have accepted pull-requests and are contributors, members who have forked an old project, members who are stargazers and also project members.

3.1.1 MAKING A GitHub API CONNECTION

GitHub, having a social web property, also implements Oauth and using the GitHub API involves creating an account followed by either creating an application to use as the consumer of the API or creating a personal access token linked directly to your user account.

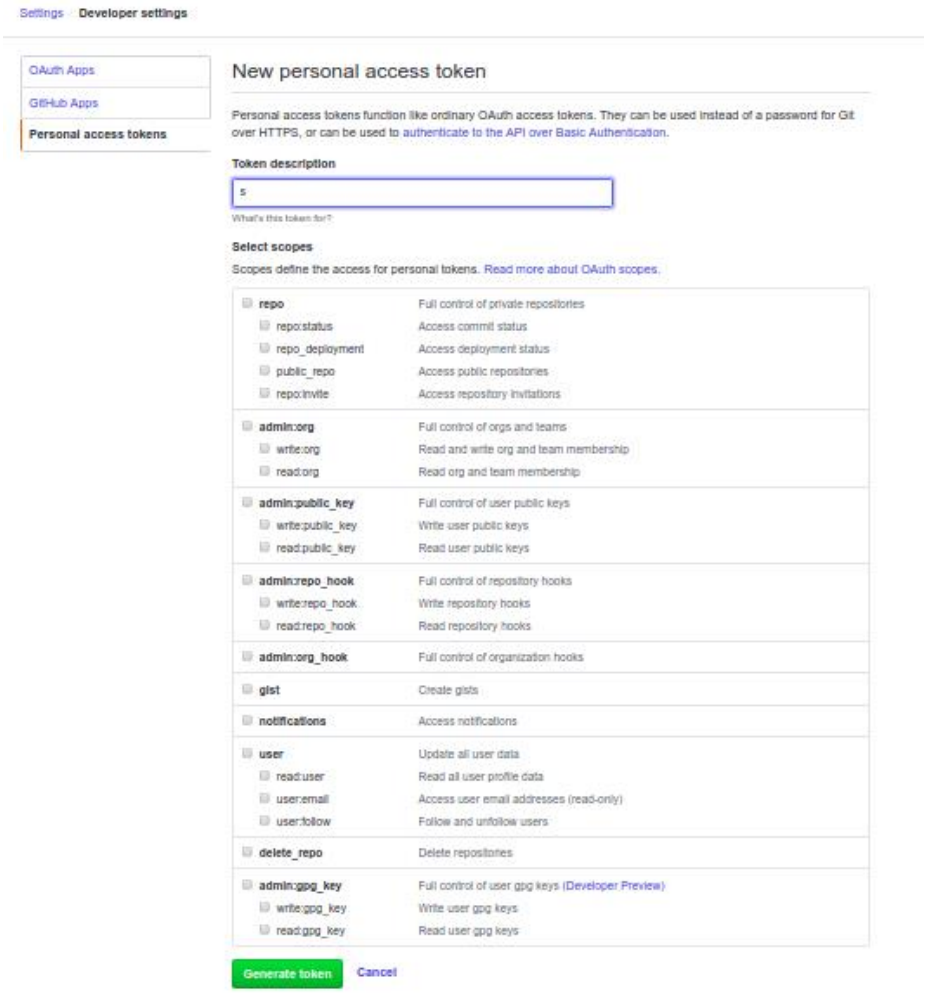

Figure 3.1: Creating a personal API token to use in accessing the GitHub API for data extraction and collection

In order for one to make this call programmatically, PyGitHub, a module used in Python for accessing GitHub data could be used.

```
from github import Github

# XXX: Specify your own access token here

ACCESS_TOKEN = '2a1ca1f031d45c4cb69d91d89d3a3a715525f9d9'

# Specify a username and repository of interest for that user.

USER = 'nkorojoseph'
REPO = 'thesiswork'

client = Github(ACCESS_TOKEN, per_page=100)
user = client.get_user(USER)
repo = user.get_repo(REPO)
print('repo%s'%repo.name)
print(user)

print(repo.language, user.login)
# Get a list of people who have bookmarked the repo.
# Since you'll get a lazy iterator back, you have to traverse
# it if you want to get the total number of stargazers.

stargazers = [ s for s in repo.get_stargazers() ]
print ("Number of stargazers", len(stargazers), stargazers)
```

Figure 3.2: Getting stargazers from a repository using PyGitHub.

The returned data is mainly a Json dump file which could be saved to a noSQL database or also use a Json module to format the output. Most of these API calls were not carried out in this work, since Gousios (2015), Gousios and Spinellis (2012) and GitHubarchive have a free explorable dataset in the Google BigQuery, although one has to pay for a Google cloud service to have a complete access to the data. Most of the datasets used in this work are a combination of BigQuery Standard and Dialect SQL queries from both GitHubarchive and ghtorrent datasets.

3.2 TEXT FEATURE EXTRACTION

Text analysis is one of the most widely applied fields where machine learning algorithms are highly relevant. However, the raw text which is a sequence of symbols cannot be directly fed to the algorithms owing to the fact that “the algorithms expect numerical feature vectors with a fixed size rather than the raw text documents with variable length” (Scikit-learn.org, 2009). In information retrieval, TF-IDF is a well-known method to evaluate the importance of a word in a document. TF-IDF

is a very good way of converting the textual symbolic representation of text documents into sparse features or vector space model (VSM) which the machine learning algorithms could use. VSM in this instance should not be confused with the mathematical vector spaces but in this context “is a space where text is depicted as a vector of numbers instead of its original string textual representation; it represents the extracted features from the document” (Chris, 2012). In order to extract these features, there are common ways to extract numerical features from atext document, which are:

Tokenizing: In a character sequence from a document unit, tokenizing chops it up into a pieces called tokens and throws away certain characters like punctuation and white spaces which are in most instances the token separator An example is:

- **Input**: installation, of the system, computer, laptop and digital system
- **Output**: installation, of , the, system, computer, laptop, and, digital, system
- **Counting**: This counts the occurrences of tokens in a document.
- **Normalizing and weighting** using important tokens whose occurrence in the sample document is a majority.

Features in this instance refer to each individual token occurrence frequency, whether normalized or not. For all token frequencies, its vector for a given document is considered a multivariate sample. A matrix with one row per document and a column per token occurrence forms the corpus of documents. Vectorization is therefore a general process of turning a collection of text documents into numerical feature vectors. Tokenization, counting and normalization makes up the ‘Bag of Words’ representation for a corpus.

In a large corpus, there are common words that are always very present (like “the”, “a”, “is”, “an”, “for”) which carry little information about the contents of the document, whereas the rare but most interesting terms in the document are always shadowed by the common words. If we were to feed the algorithms with both categories of word, the meaningful information will be shadowed by the less

important but most frequently occurring words. TF-IDF which means (term frequency * inverse document frequency) converts these features into floating point values best fitted for the classification algorithms. Since most documents will use a very small subset of the words used in the corpus, the resulting matrix will have many zeros representing some features. In order to speed up algebraic operations and also represent such a matrix in memory, sparse representation of such a matrix is needed. Thus in this work scikit-learn NLP module is used to handle these processes ranging from tokenization, vectorization, TF-IDF and sparse matrix representation. Going with the explanations and definitions on TF-IDF, we define the term-frequency as a function for counting the presence of a vocabulary:

$$tf(t,d) = \sum_{x \in d} fr\,(x,t) \tag{1}$$

where $fr(x,t)$is a simple function defined as :

$$fr(x,t) = \begin{cases} 1, if x = t \\ 0, otherwise \end{cases} \tag{2}$$

What the $tf(t,d)$ returns is the number of times term t is present in document d. The document vector can now be represented as

$$\overrightarrow{v_{d_n}} = (tf(t_1,d_n), tf(t_2,d_n), tf(t_3,d_n), \ldots\ldots, tf(t_n,d_n),) \tag{3}$$

noting that the $tf(t_n,d_n)$ will eventually be normalized.

It is obvious that each dimension of the document vector is represented by the term of the vocabulary. This also draws attention that the most occurring terms does not imply a best discriminator of a document. The idf (inverse document frequency) weighting in combination with the tf solves this problem. Idf is defined as

$$idf(t) = log \frac{|D|}{1+|\{d:t \in d\}|} \tag{4}$$

where $|\{d: t \in d\}|$is the number of documents where 't' appears, when the term-frequency function satisfies$tf(t,d) \neq 0$. The addition of 1 helps avoid zero-division.

The TF-IDF is then

$$tf - idf(t) = tf(t,d) * idf(t) \quad (5)$$

the impact of this formula is that a high weight of the TF-IDF calculation is reached when there is a high term frequency in the document (local parameter) and a low document frequency of the term in the whole collection (global parameter).

3.3 PROPOSED ALGORITHM

Trust-aware recommender systems follow the maxim "tell me your friends and I will tell you who you are" and this concept is the underlying method used in accomplishing the task of recommending developers to a new project. The proposed approach in this work takes a new turn from the conventional trust aware recommendation systems explored in the literature review. The existing methods of trust-aware recommendation centre on enhancing collaborative filtering for item recommendation using user-item rating matrix. But this approach tries to take the approach of content-based recommendation system by using item description and developer profiling in building preferences and also takes the approach of the collaborative recommendation systems by exploring developer behaviour on GitHub and building a trust level based on how trustworthy the user could be to an existing project. Having identified that there is no explicit trust rating in GitHub activities, implicit methods are adopted in this work.

The proposed approach could be classified into phases

3.3.1 PHASE 1

1. Data extraction.

2. Old projects Readme file extraction.
3. New project Readme file preparation.

3.3.2 PHASE 2

In this phase, a stemmer which would be used to reduce each word to its root form in the readme files is first generated. The use of this stemmer is to help reduce the noise in each document. Along with a punctuation removal function, all words are ensured to be in their root form. Example, should a word like “cat” be identified, other variations of the word like (“cats”, “catlike”, “catty”, “cat’s”) will all be stemmed down to “cat”. Before stemming, the text needs to be tokenized to break the document into tokens separated by white-spaces. Then we applied the TF-IDF Vectorizer to get the term frequency – inverse document frequency of each text in the document. To perform similarity check between the new project readme files and all already existing readme files (old project readme files), we applied cosine similarity between the two vectors from the output of the previous stage (I.e TF-IDF vectors). Cosine similarity between two vectors (or two documents in vector space) is a measure that calculates the cosine of the angle between them. The is a measure for orientation and not magnitude, it can be seen as a comparison between documents on a normalized space because we’re not taking into consideration only the magnitude of each word count (TF-IDF) of each document, but the angle between the documents. This is dot product of the $cos\theta$:

$$\vec{a}.\vec{b} = \|\vec{a}\|\|\vec{b}\|cos\theta \quad (6)$$

$$cos(\theta) = \frac{\vec{a}.\vec{b}}{\|\vec{a}\|\|\vec{b}\|} \quad (7)$$

Let us assume a document has a word “play” which appears 200 times and another document has the same word “play” appearing 50 times. The Euclidean distance between them will be greater but the angular distance between them will still be small because both documents are pointing to the same direction, and this matters most when comparing documents.

Algorithm 1.0: Programming languages similarity

```
input: OldReadme files.
Output: Similarity value list
0 def cosine_sim(text1, text2):
1        tfidf ←- vectorizer.fit_transform([text1, text2])
2  for files in range(length(oldreadmefiles)):
3      sim ←- cosine_sim(oldreadmefiles[ i ] , newreadmefile[0])
4  return similarity values
```

With the above algorithm, the cosine similarity is computed and its values are stored in similarity values. Since in this work we are trying to consider most related or similar project, the project with the maximum similarity value is extracted with the algorithm below;

where Mostsimproj represents the most similar project whose trusted project members are to be

Algorithm 2.0: Similar project identification

```
input: similarity values.
Output: Most similar project
1 for values in range(length(simivalues)):
2   if simivalues[i] > maxsim:
3  maxsim ←- simivalues[i]
4  Mostsimproj ←-file name of most similar project
```

recommended to the new project.

3.3.3 PHASE 3

In phase 3, most of the work is done using graph edge analysis. In terms of a recommendation system, an item in this case and henceforth is regarded as a project or project owner. A developer is a trusted and tagged competent if he or she meets the following criteria:

- Starts the project;
- Active commits;
- Has accepted pull request(s); and
- Has forked the project.

In this work, since modeling of distrust is out of the scope of the work, if a developer does not meet these four requirements he is categorically not worth trusting for a new project. The importance of this plays out when considering the experience level of a developer. This will be explained further in the next phase. To explore each user action, each of the dataset involved in the four criteria are converted to a graph. The graphs represent user-project relationship where vertices (nodes) represent users and projects and an edge shows connection for each of the criterion. With the four graphs set, the algorithm below checks for an edge existence in each of the graphs. If an edge exists, the edge is added to a new graph which represents the most influential and trusted developers along with the projects they are involved in. The output of this algorithm is another graph called the trusted developers graph.

Algorithm 3.0: Project-Developer Trust Graph

```
input:  users, projects and connection files.
Output: Trusted developer graph
0 G ←- nx.DiGraph()
1 for user in totalusers:
2    for project in totalprojec:
3        if F.has_edge(i,j) and W.has_edge(i,j) and P.has_edge(i,j) and
         C.has_edge(i,j):
4            G.add_edge(i,j)
```

where F,W,P,C are all forks, watchers, pull requests and commits graphs respectively. The more a developer connects in this trust graph G, the more value he gets when calculating his experience level.

3.3.4 PHASE 4

The users in the graph G are referred trusted users, but that a user has met these criteria does not show his or her relevance to the new project which he or she is to be recommended. In order to put this into consideration, an experience level which is the sum of all the projects participated in the trusted developers graph G.

Let $u_p = \{p_1, p_2, \ldots\ldots\ldots p_n\}$ represent the projects a user "u" has participated in, such that $p \subset P$ where P is the set of all the projects in the trust network.

Let $u_l = \{l_1, l_2, l_3, \ldots\ldots\ldots\ldots, l_n\}$ represent the languages a user "u" has used based on his list of projects such that $l \subset L$ where L is the set of all programming languages that were used in projects in the trust graph.

User experience level can then be computed by

$$E(u_p) = \frac{\sum U_p}{\sum P} \tag{8}$$

The algorithm below does the computation of the experience level, the final experience level is a normalized value to leave the range between [0,1].

Algorithm 4.0: Developer Experience level

```
input: nodes from trust graph
Output: developer experience level
0usercount ←- Counter(occur_users)
1 j ←- dict(usercount)
2 for key,value in j.items():
3    normUsercount[key] ←- value/length of j
```

3.3.5 PHASE 5

Along with all projects in GitHub is a list of the programming languages used in the project. A further exploration into the code used in any project reveals the number of lines each programming languages occupies and this tells a lot about the behaviour of developers who has worked on that project and also serves as a good value for classification of projects based on programing languages. Although lines of codes are not considered in this work, this and other activities could be good markers for developer behaviour analysis.

This phase first tries to build a user profile for each developer based on the programming languages found in the projects in which he is a trusted developer (see edge connection in graph G).

Algorithm 5.1: Building a developer profile

```
Input: Trusted Users and connected projects
Output: Developer profile
1.for user, project in zip(occur_users, occur_projects):
2.   if user in QUserExp:
3.       QUserProj[user] ←- project
4 read in projLang.csv as reader
5 for row in reader:
6 prolang.append(row)
7 for lang in prolang:
8 langdictionary['p'+lang[ 0 ] ] ←- lang[1:]
9 for key,value in QUserProj.items():
10    for key1,value1 in dicti.items():
11        if value ←- key1:
12            QUsersProfile[key] ←- value1
```

Here a comparison is made between the languages of the new project to which a developer from the most similar project is to be recommended and the user profile is built. A set function returns unique items; hence, a set was used to return intersection of the programming languages and this is further normalized to put the final value between [0,1]. The developer programming language relevance or similarity score is computed as

$$R(u_l) = \frac{\sum u_l \cap N_l}{\sum N_l} \quad (9)$$

such that N_l represents the set of languages to be used in the new projects.

The totalRel which is in general referred to as the project direct trust value to a developer is then computed as

$$T(p,u) = R(u_l) + E(u_p) \quad (10)$$

The rank of a developer to the new project is a summation of the developer's relevance score and the experience level. The algorithm below carries out the language relevance check.

Algorithm 5.2: Developer programming language skill relevance

```
input: Programming Languages.
Output: Language similarity.
0 for lang in newprolang:
1    newdicti[lang[0]] ←- languages
2  for key,value in QUsersProfile.items():
3    lang_sim ←- set(value).intersection(set(newdicti['pnew']))
4    langsim[key] ←- len(lang_sim)/len(set(newdicti['pnew']))
5  for key,value in QUserExp.items():
6    totalRel[key] ←- QUserExp[key] + langsim[key]
```

Developers are ranked based on the values in “totalRel” dictionary. This is similar the conventional recommender systems in which predicted ratings are using for item ranking which is to be used as the recommendation set.

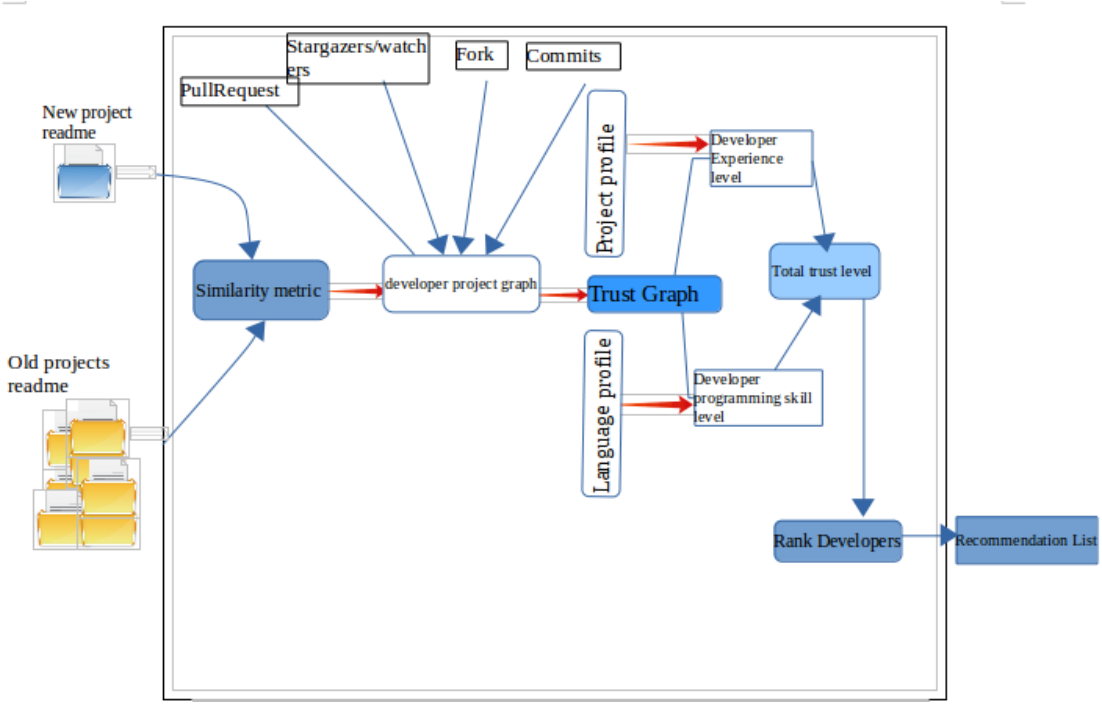

Figure 3.3: Proposed Architecture of ProjectTrust

CHAPTER FOUR
EXPERIMENTATION AND RESULT

4.1 WORD CLOUD VISUALIZATION OF README FILE SIMILARITY

The use of graphs, info-graphics, charts and more are good ways of data visualization which gives valuable insight about important information at a glance, but these methods are lacking when it comes to giving insight about raw data in text formats. The use of word clouds is a stunning visualization method for highlighting important textual data points and conveying crucial information about dull text data.

Word clouds, sometimes called tag clouds are “an image composed of words used in a particular text document or subject, in which size of each word indicates its frequency or importance” (“what is word cloud - Google Search,” 2017, p. www.google.com). Word clouds work in a simple way, the more a word appears in a text document, the bigger it appears in the word cloud image. Understanding that word clouds are good ways of gaining insight into text, it could also be adopted in gaining insight about the similarity between two readme files and thus word cloud is adopted in this project as a means of visualizing the result of the first two phases of the project which includes finding similarity between readme files. To visualize this similarity, the word cloud of the new readme file is generated and at the end of the similarity check, the word cloud of the most identified similar old project readme file is also generated. For example, a new project readme file generates a word cloud with words like *(Akka, doc, io, GitHub, https, java, questions make , Scala, package, chat and so on)* being the most dominating and most occurring words. At the end of the similarity check, a project “p18” was identified to be the most similar project. The word cloud of project “p18” shows that words like *(“akka”, “https”, “package”, “io”, “scala”, “applications”, “chat” and others)* also appear to be most frequent or occurring words but at different strengths as shown in the image below.

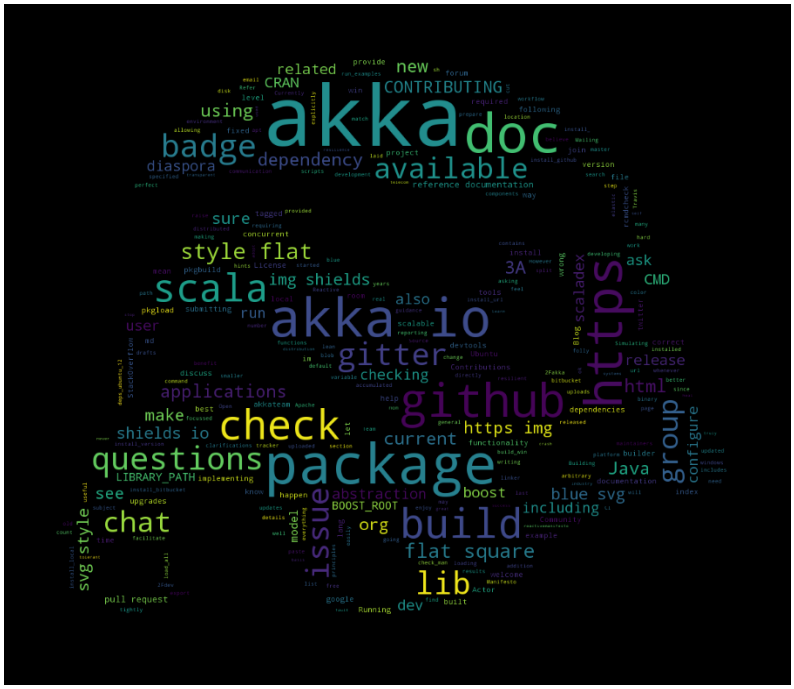

Figure 4.1: Word Cloud of New ReadMe File

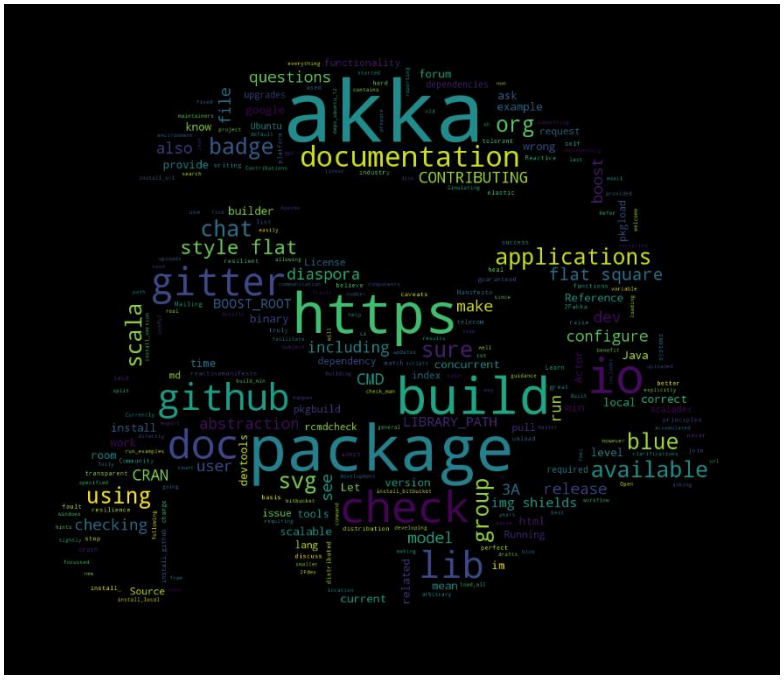

Figure 4.2: Word Cloud of the readme file of the identified most similar old project.

4.2 RECOMMENDATION EVALUATION

In the absence of any benchmark method for this new approach in trust recommendation for social coding platforms, since to the best of my knowledge this is the first method proposed, a cross validation approach similar to a machine learning test and training data approach is used to check the accuracy of the recommended developers to a project. To achieve this cross validation, five projects and their readme files were kept in another folder as test_readme with their content a little different from their root project readme file. The developers involved in each of the projects in the test dataset were also extracted by introducing project members GitHub data. Knowing that the algorithm makes a recommendation based on the most similar readme, each test project tries to identify their root project readme as the most similar project and makes recommendation of developers to the test project.

The results of this five trials are tabulated below.

Table 4.1: Recommendation List Accuracy

Project	Actual developers	Recommended developers	Accuracy
P1	[['u9', 'u6', 'u2', 'u4', 'u1']	['u2', 'u1', 'u6', 'u4']	0.8
P2	['u12', 'u7', 'u5', 'u6', 'u4', 'u1', 'u13']	['u1', 'u6', 'u4', 'u5', 'u13']	0.7143
P3	['u4', 'u24', 'u14']	['u4']	0.33
P4	['u5', 'u6', 'u2', 'u4', 'u29', 'u1', 'u8']	['u2', 'u1', 'u6', 'u4', 'u5']	0.7143
P5	['u84', 'u7', 'u79', 'u9', 'u2', 'u4', 'u11', 'u28', 'u1', 'u15', 'u13', 'u1o']	['u4', 'u1o', 'u11', 'u1', 'u13']	0.5

4.3 CORRELATION BETWEEN TRUSTED DEVELOPERS WITH PROGRAMMING LANGUAGES AND WORK EXPERIENCE

Correlation between datasets is a relationship measure of how well the data are related. The most common of this relationship measurement is Pearson product-moment correlation or PPMC but commonly referred to as a Pearson correlation. This shows a linear relationship between two sets of data. It shows the strength of a linear association between two variables, and is denoted by "r". PPMC tries to draw a line of best fit through the data of two variables and r (which is the correlation coefficient) indicates the distance of these data points to the line of best fit. The Pearson correlation coefficient, r, can take a range of values from +1 to -1. A value of 0 indicates that there is no association between the two variables. A value greater than 0 indicates a positive association; that is, as the value of one variable increases, so does the value of the other variable. A value less than 0 indicates a negative association; that is, as the value of one variable increases, the value of the other variable decreases.

$$r = \frac{n(\sum xy) - (\sum x)(\sum y)}{\sqrt{[n(\sum x^2) - (\sum x)^2][n\sum y^2 - (\sum y)^2]}} \quad (1)$$

Just as stated earlier in Chapter Three, recommending all trusted developers without proper consideration of their experience level and programming language level is a big drawback which might lead to recommending developers who are experienced but with little programming language relevance to the new project, or the reverse might be the case. The experiments with the five projects shows there is a high correlation between developers' experience level and the trusted developers in graph G while there is a low or reverse correlation between trusted developers and their programming language similarity. This is an obvious fact during recruitment, since employers will always prefer to hire developers with broader experience in preference to those with a large programming skill set but who have worked on few projects. For the five tested projects, the value of 'r' for experience level was always higher than programming language similarity.

Table 4.2: Correlation values of experience level, programming language similarity and trusted developers

Project	Experience correlation (r)	P-language sim correlation (r)
P1	0.8863	0.3032
P2	0.2090	-0.1440
P3	0.2290	-0.0002
P4	0.87759	0.1689
P5	0.8996	-0.1634

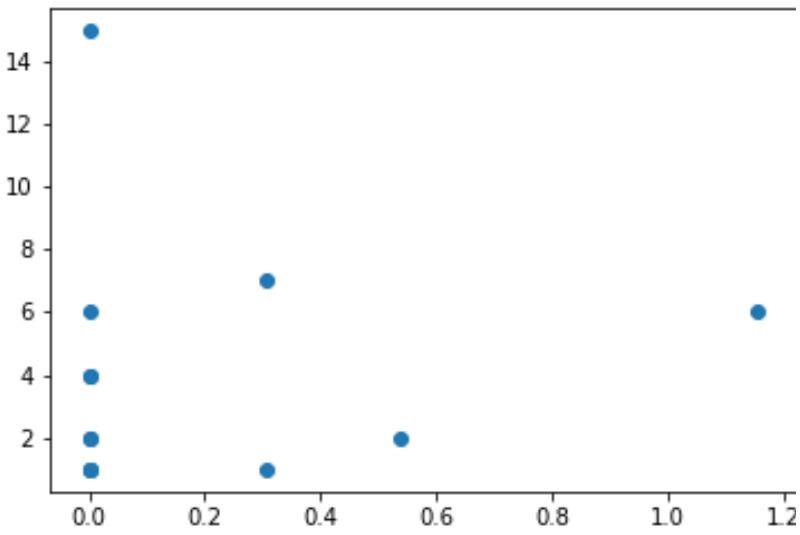

Figure 4.3: Correlation between experience level and trusted developers for project p1

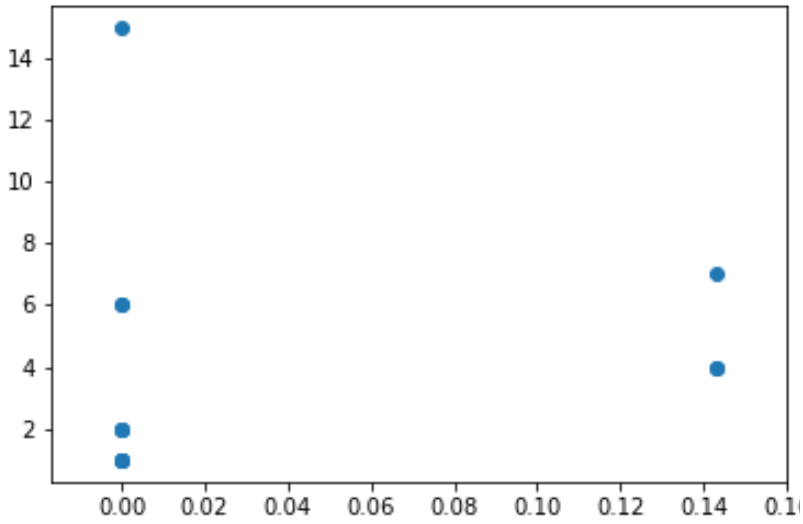

Figure 4.4: Correlation between language similarity and trusted developers for project p1

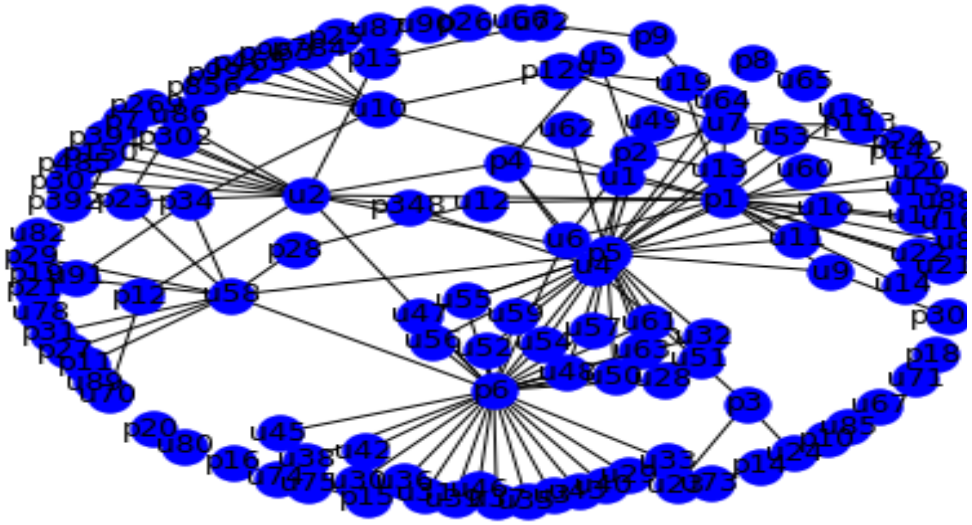

Figure 4.5: Sample Trust Graph

CHAPTER FIVE
SUMMARY, RECOMMENDATION AND CONCLUSION.

5.1 SUMMARY

This work focuses on recommending existing developers to a new project based on their social coding behaviour in GitHub. In this thesis, ProjectTrust, a trust-aware content-based recommendation model to estimate trust between projects and developers based on users' interaction was developed. The thesis investigated an existing graph-based trust inference algorithm like tidal trust. An extensive literature review of trust inference algorithms and developer to projects recommendations alongside state of the art methods were discussed. A natural language processing approach was looked into and was found to be a good tool for text feature extraction in GitHub readme files. Vector space model like cosine similarity were also looked into for similarity check. Experience level and skill set level were all looked into. The work finally presents a new trust inference recommendation approach for social coding platforms like GitHub.

5.2 CONCLUSION

As part of a contribution in the field of recommendation systems, a recommendation system based on project-user trust behaviour is developed in this work. Though not statistically sound due to lack of data, experiments showed that ProjectTrust has some promise in trust-aware recommender systems for social coding platforms. It also showed that considering developers' experience level rather than programing languages skill set is more beneficial in building developer recommendation engines. Experiments from this research also suggest that project-developer trust could be a solution for a platform like GitHub in profiling of developers for proper recommendation to a newly created project.

5.3 FUTURE WORK

The work done in this thesis has brought about a set of open questions which might suggest better recommendation listing and trust computation. One of such suggestions is the evaluation of number of lines of codes a developer sends as a pull-request to a project which could be a good measure of how much skilled and experienced a developer could be. Although computing similarity based on similarity in readme file of projects is a good approach, computing project similarity based on intersecting developers is one of such questions open for future work. The work considered mainly four out of more than 21 GitHub Events. Other GitHub events could be explored and this might suggest stronger project to developer direct trust computation, thereby leading to better recommendation.

REFERENCES

Abderrahim, N., & Benslimane, S. M. (2015). Towards Improving Recommender System: A Social Trust-Aware Approach. *International Journal of Modern Education and Computer Science*, *7*(2), 8–15. https://doi.org/10.5815/ijmecs.2015.02.02

Adomavicius, G., & Tuzhilin, A. (2005). Toward the Next Generation of Recommender Systems : A Survey of the State-of-the-Art and Possible Extensions, *17*(6), 734–749.

Bouraga, S., Jureta, I., Faulkner, S., & Herssens, C. (2014). Knowledge-Based Recommendation Systems. *International Journal of Intelligent Information Technologies*, *10*(2), 1–19. https://doi.org/10.4018/ijiit.2014040101

Burke, R. (2014). Hybrid Recommender Systems: Survey and Experiments. Retrieved from https://pdfs.semanticscholar.org/5880/b9bc3f75f4649b8ec819c3f983a14fca9927.pdf

Burke, R. (2002). Hybrid Recommender Systems: Survey and Experiments. *Springer*. Retrieved from https://pdfs.semanticscholar.org/5880/b9bc3f75f4649b8ec819c3f983a14fca9927.pdf

Burke, R. (2003). Knowledge-based recommender systems. *Researchgate*. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.21.6029&rep=rep1&type=pdf

Carrasco, A. L. (2012). Master of Science Thesis Towards Trust-aware Recommendations In Social Networks, (June).

Chris, S. (2012). Machine Learning: Text Feature Extraction (TF-IDF) - Part I - DZone Web Dev. Retrieved November 5, 2017, from https://dzone.com/articles/machine-learning-text-feature

Dorri Nogoorani, S., & Jalili, R. (2012). Uncertainty in probabilistic trust models. *Proceedings - International Conference on Advanced Information Networking and Applications, AINA*, 511–517. https://doi.org/10.1109/AINA.2012.73

Efron, B., & Tibshirani, R. (1986). Bootstrap methods for standard errors, confidence intervals in measure of statistical accuracy. *Statistical Science*, *1*(No. 1), 54–75.

Ekstrand, M. D. (2011). Collaborative Filtering Recommender Systems. *Foundations and Trends® in Human–Computer Interaction*, *4*(2), 81–173. https://doi.org/10.1561/1100000009

ericsink.com. (2017). A History of Version Control. Retrieved September 2, 2017, from http://ericsink.com/vcbe/html/history_of_version_control.html

Francesco, R., Lior, R., Bracha, S., & Paul B., K. (2014). *Recommender system handbook*.

Golbeck, J. (2008). COMPUTER SCIENCE: Weaving a Web of Trust. *Science*, *321*(5896), 1640–1641. https://doi.org/10.1126/science.1163357

Gousios, G. (2015). The GHTorent dataset and tool suite The GHTorent Dataset and Tool Suite, (June). https://doi.org/10.1109/MSR.2013.6624034

Gousios, G., & Spinellis, D. (2012). GHTorrent: GitHub's data from a firehose. *IEEE International Working Conference on Mining Software Repositories*, (Section II), 12–21. https://doi.org/10.1109/MSR.2012.6224294

Hauff, C., & Gousios, G. (2015). Matching GitHub Developer Profiles to Job Advertisements Matching GitHub developer profiles to job advertisements, (May). https://doi.org/10.1109/MSR.2015.41

Jannach, D., Dortmund, T., & Friedrich, G. (2013). Tutorial: Recommender Systems. Retrieved from http://ijcai13.org/files/tutorial_slides/td3.pdf

Kuter, U., & Golbeck, J. (2005). SUNNY : A New Algorithm for Trust Inference in Social Networks Using Probabilistic Confidence Models. *Network*, *2*(2), 1377–1382. https://doi.org/10.1145/1754393.1754397

Kuter, U., & Golbeck, J. (2010). Using probabilistic confidence models for trust inference in Web-based social networks. *ACM Transactions on Internet Technology (TOIT)*, *10*(2), 8. https://doi.org/10.1145/1754393.1754397

Levien, R. L. (2004). Attack Resistant Trust Metrics Chapter 1 Introduction. *Draft Available at Httpwww Levien Comthesiscompact*, 1–27. https://doi.org/10.1.1.9.7825

Lisette, R., Portugal, Q., Casanova, M. A., Li, T., Sampaio, J. C., & Leite, P. (2015). GH4RE: Repository Recommendation on GitHub for Requirements Elicitation Reuse. Retrieved from http://www-di.inf.puc-rio.br/~casanova//Publications/Papers/2017-Papers/2017-CAiSE_Forum-Portugal.pdf

Mark, C., Anuja, G., Tim, M., Pavel, M., Dmitry, N., & Matthew, S. (1999). Combining content-based and collaborative filters in an online newspaper. Retrieved from https://web.cs.wpi.edu/~claypool/papers/content-collab/content-collab.pdf

Massa, P., & Avesani, P. (2007). Trust-aware recommender systems. *Proceedings of the 2007 ACM Conference on Recommender Systems RecSys 07*, *20*, 17–24. https://doi.org/10.1145/1297231.1297235

Massa, P., & Avesani, P. (2009). Computing with Social Trust, 259–285. https://doi.org/10.1007/978-1-84800-356-9

Netrvalova, A., Safarik, J., & Republic, C. (2009). Phenomenal Trust Model.

Netrvalova, A., Safarik, J., & Republic, C. (2012). Trust model for social network, (1).

Nordheimer, K., Schulze, T., & Veit, D. (2010). Trustworthiness in Networks: A Simulation Approach for Approximating Local Trust and Distrust Values. *Trust Management Iv*, *321*, 157–171. https://doi.org/10.1016/j.jnca.2011.03.005

Papaoikonomou, A., Kardara, M., & Varvarigou, T. (2015). Trust Inference in Online Social Networks. *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015 - ASONAM '15*, 600–604. https://doi.org/10.1145/2808797.2809418

Raina, R., Battle, A., Lee, H., Packer, B., & Ng, A. Y. (2008). Self-taught Learning: Transfer Learning from Unlabeled Data. Retrieved from http://ai.stanford.edu/~hllee/icml07-selftaughtlearning.pdf

Ricci, F. (2012). Part 15 : Knowledge-Based Recommender Systems. Retrieved from https://www.ics.uci.edu/~welling/teaching/CS77Bwinter12/presentations/course_Ricci/15-KnowledgeBased.pdf

Samaneh, S., Mehta, M., & -Rahul, M. (n.d.). Collaborative Filtering Recommender Systems.

Scikit-learn.org. (2009). 4.2. Feature extraction — scikit-learn 0.19.1 documentation. Retrieved November 5, 2017, from http://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction

Taherian, M., Amini, M., & Jalili, R. (2008). Trust inference in web-based social networks using resistive networks. *Proceedings - 3rd International Conference on Internet and Web Applications and Services, ICIW 2008*, 233–238. https://doi.org/10.1109/ICIW.2008.41

what is word cloud - Google Search. (2017). Retrieved November 9, 2017, from https://www.google.com.ng/search?ei=Y5MEWtXCJ83TkwWZorCwBw&q=what+is+word+cloud+&oq=what+is+word+cloud+&gs_l=psy-ab.3..0l4j0i22i30k1l6.57.554.0.1261.5.4.0.0.0.0.293.584.2-2.2.0....0...1.1.64.psy-ab..4.1.290....0.1j8DH8_WHOk

Ziegler, C., & Golbeck, J. (2015). *Propagation Phenomena in Real World Networks* (Vol. 85). https://doi.org/10.1007/978-3-319-15916-4