DREU Final Report
University of Wisconsin - Madison

Rebekah Manweiler

May 18 - Aug 3, 2018

1 Abstract

GitHub, a social coding platform, has become a rich source of information. Researchers seek to understand the structure, growth, and influence of the social environment it has become. As part of a DARPA Challenge to model the GitHub environment, this project aims to model the users of GitHub. We created a descriptive cognitive model classifying three types of users on GitHub, core developers, active users, and passive users, affirming previous research. From this cognitive user model we were able to create a predictive model of user behavior. Specifically, we created a model that, when given a repository, the user that owns the repository (the owner), and the user that performed an action on the repository (the actor), could predict the type of action performed on the repository by the actor. We also determined that the actor's information was more predictive of the action taken than the owner's information.

2 Introduction

2.1 Project Background

This summer I worked with Professor Nicole Beckage and Professor Joe Austerweil at the University of Wisconsin - Madison. They were working on a DARPA Challenge whose goal was to model different aspects of the Software Social Network (SSN) GitHub. At the start of my research experience they were working on the baseline challenge which focused on "simulating [the] social structure and temporal dynamics of key processes, as well as individual, community and population level behavior on GitHub" as stated in the DARPA baseline challenge description. They have been given GitHub user data and repositories, and five high level research questions. These research questions are further specified with 31 different questions and related metrics.

- RQ1: How do users **engage** in the development of technologies and how does their engagement evolve?
- RQ2: How do users **contribute** to the evolution of technologies; how much, how quickly, and how evenly?
- RQ3: How do users' levels of **trust and reputation** impact the rate of innovation growth?
- RQ4: How do contributions to one technology **influence** the development of other technologies?
- RQ5: How quickly do technologies become **popular** and how does their popularity evolve over time?

My work this summer was centered around this project. I used the MySQL GHTorent database to model users and their actions on GitHub. I began narrowing my project focus with the list of metrics from the main project. I categorized each question by determining if it focused on a user, repository, or community, and if it focused on a general model or comparative model. The questions and related metrics I selected for my project scope were questions I categorized as general user model questions and are listed below.

- (27 from RQ1) How soon do users engage with GitHub over time after joining?
  Metric - Diffusion delay of user actions (excluding Fork and Watch events) since the creation of the user account.

- (30 from RQ1) Do users who initially engage with repositories continue to contribute over time?
  Metric - Probability that users will continue to contribute to the repository given how many previous interactions with that repository the user had.
- (31 from RQ1) How much activity is required before the users become committed project contributors?
  Metric - The ratio of the probability that a user's next action will be an Issues event to the probability that it will be a Push event as a function of how many previous interactions the user has had with a repository.
- (5 from RQ2) What are typical patterns of activity observed for developers on GitHub?
  Metric - Distribution of total events by week day/hour.
- (17 from RQ2) Do users contribute across many repositories?
  Metric - Number of unique repositories that users contribute to.
- (24 from RQ2) What are the basic characteristics of the developers' population?
  Metric - Distribution over user activity for all users. Top K most active users (total number of actions per user).

My goal with these questions is to understand how people are using GitHub and how their usage changes over time as well as the different kinds of GitHub users and the differences in their behavior. We already have an intuition for the different kinds of GitHub users like students, professors, small businesses, and companies, and we can imagine how they might use them differently as a starting point. For instance, students may use their GitHub accounts more sporadically and more exclusively for school projects with other students while developers at a company may use their accounts more regularly in conjunction with a software development framework and for multiple ongoing projects. But, I want to know if we can systematically detect and categorize those differences to generate a mathematical basis for all the current and potential users of GitHub and from there be able to more accurately model and predict user behavior.

2.2 GitHub

GitHub is a network of public 'repositories' or projects which can contain anything from documents to images. GitHub is generally used as a platform to share and manage coding projects including software for games or applications, libraries for coding languages, manuals and specifications for software or hardware, and much more. GitHub was built off of the document source control system, Git, which allows users to manage and edit different versions of documents over time. GitHub stores a copy online of any folder or file system that is being monitored by Git: these are a user's repositories. Repositories and local folders are updated by using commands like 'push' or 'pull'. GitHub is largely used by groups or teams of users that are all working on the same project. Users will all have a local copy of the repository which they can work on and edit, and then when they want to share their changes with the group they can 'push' their changes to the online repository and tell other users to 'pull' their changes from the online repository. Users can also add comments to their changes to give notes or explanations for their changes. In this way, groups can easily edit and share a project.
Additions to this environment has allowed it to take on an aspect of social media. Users on GitHub can follow the activity of other users, watch the activity on a repository from many users and receive notifications for activity, star certain projects to bookmark them for later or receive more notifications on repository activity, fork a users project to create their own local copy, create pull requests for repositories they do not own to ask for their changes to be included in a project, and several more. These additions create an environment for users to interact with each other while developing and managing software.

2.3 Prior Research

In the recent years, research into the behavior of users on social networking sites like Twitter and Facebook has sparked a similar interest in social coding platforms like GitHub and Stack Overflow. These websites allow software developers to collaborate on projects, ask questions or solve problems, and share code easily, creating a community rich in resources and ideas. Research on these social software networks is largely focused on the users and their behavior, and is spread between many levels of community: from individual users to entire ecosystems.
Starting at a high level and studying the structure of these ecosystems, Blincoe et al. found that most ecosystems revolve around a single project that facilitates and supports software development such as libraries or frameworks, and that those center projects are highly interconnected to other ecosystems [4]. Within projects, work done by Loyola and Lima to understand the population of contributors for a given project has shown that the distribution of contributors per project follows a power-law distribution [9] and that the number of contributors on a project impacts the quality of the project [10]. They also found that the number of followers per user follows a power-law and observed low levels of reciprocity from users with many followers to users with few followers [9]. Looking further into the behavior of users and specifically the interactions between users, Dabbish and Marlow found that GitHub users have many reasons for seeking out information about other users [12] and when they do they make a "rich" set of inferences about the user and form an extensive impression of that user [6]. Their research has also identified what kind of information users utilize when forming their impressions and how they use their impressions. More on user behavior, Badashian et al. studied what it means to be influential on GitHub and have shown that users with many followers are not necessarily influential. They also discovered that there are two different types of influence found on GitHub: popularity or fame, and quality of content [2]. The last level of research focuses on categorizing user behavior and identifying different types of users. An example researched by Blincoe et al. is popular users and how they build their influence and use it within GitHub [5]. Another by Sheoran et al. is users that watch projects, coined 'watchers', and how they affect projects and how they transfer into regular contributors [15].
With these examples we find users being categorized into broad types based on behavior or based on a perceived role that they have with a given project. Matragkas et al. sought to understand the biodiversity of the user population on GitHub, determined how users could be categorized or structured within the context of a project, and if the size of a project affected its inherent structure. They found that GitHub projects of all sizes consist of three different types of users: 'core developers', 'active users', and 'passive users' [13]. Bincoe et al. described this user-type hierarchy as a "series of concentric circles" where users in the center are 'core developers' surrounded by 'maintainers' who add modules and plug-ins to a project, surrounded by a diverse sea of 'patchers' who fix bugs and report errors [5]. Understanding GitHub users and their behavior informs researchers about many things including how to best distribute information and tools to users, how to improve user's experiences, how to improve software created in GitHub and how to more widely distribute it [6]. This knowledge could also give them insights into general user behavior and user cognition as the field continues to expand.

3 Project Approach and Goal

The first goal of this project was to understand the population of Github users, and to do this I took a very similar approach as Matragkas et al. in their 2014 paper on user biodiversity. In this paper they use an unsupervised machine learning algorithm [1], the k-means clustering algorithm, to identify user types, categorize users from several project communities, and determine what user characteristics define each user type. In their work they sought to understand user behavior in terms of the user's role in a single active project, therefore if there was a user participating in multiple projects that user would be realized as a different user for each project [13]. Instead, I wish to uncover a more general behavior for users over all of their projects and actions. It is possible that some users have the ability to drastically change their behavior and actions for different projects, but I believe that if there are such users they will be few and would blend in with other highly skilled users that do not drastically change their behavior. The second goal of this project after discovering the user types is to categorize the users and determine what user characteristics define each user type, similarly to Matragkas. The final goal of this project will be to predict user behavior based on the user types. Formally this means using a supervised machine learning algorithm [13] to learn the relations between a given user's information and an action that they took on a given repository.

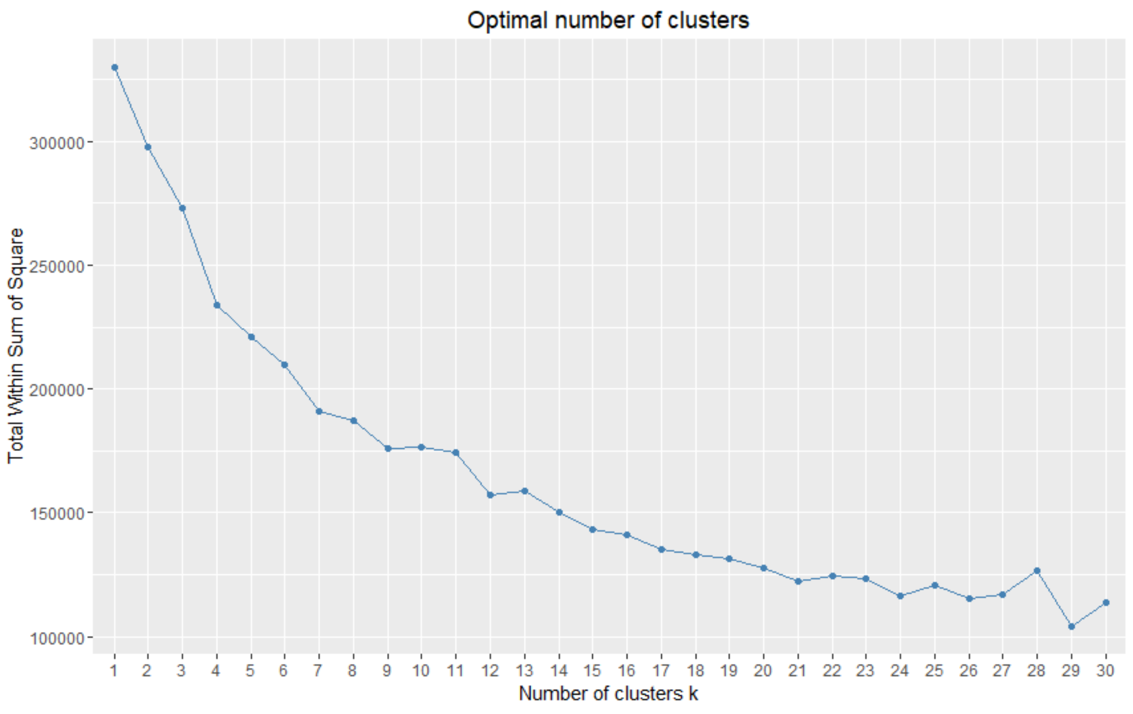
4 Methods and Results

The data used in this project was from the GHTorrent database accessible by my mentors. I was not given access to this database and was instead provided 10,000 users from approximately 100,000 random events in the database that were stripped of any identifying information. All event dates were between December 1 of 2014 and March 1 of 2015 for a time range of approximately 3 months. Each event contained information about which repository the event was done on, the user the repository belonged to (the owner), the user who did the event (the actor), and the date, time, and type of the event. Another list was provided for me of users from the event list and information about them including and not limited to a user id, the number of public repositories they owned, the number of followers they have, the number of users they were following, their country code, the day they created their account, etc. In total there were 47 different user variables.

4.1 Goal 1: Realize User Types

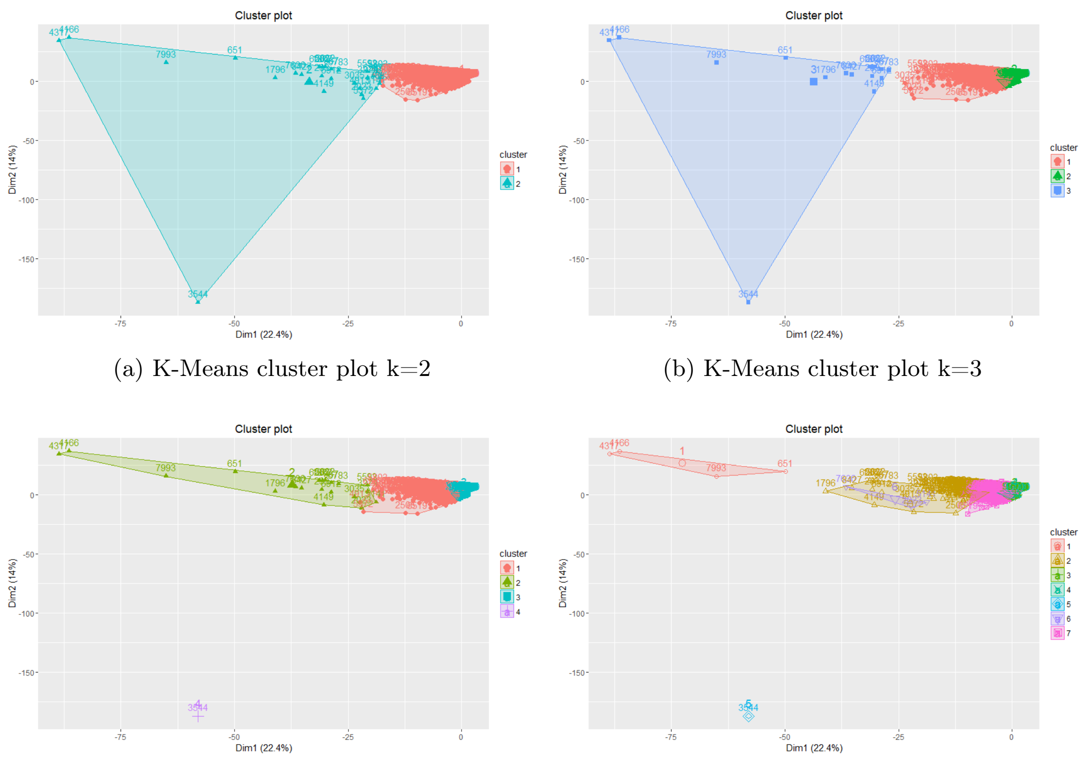
For the first goal of this project I began by cleaning the user data. I removed any variables with large amounts of NA entries and any users with NA entries. After removing 6 variables with NA entries, there was no need to remove any users. After scaling all non-factor variables, I began the k-means cluster analysis. The k-means clustering algorithm partitions the data into k clusters based on the 'closeness' or distance of the data points [11]. There are many different measures of distance available to use for this algorithm, but the default which I used was the Euclidean distance. Similarly there are several versions of this algorithm available and the default used was the Hartigan and Wong algorithm [16]. Since this algorithm requires a set number of clusters as input, I ran several iterations varying k between 1 and 30 using the elbow method. The elbow method plots k verses the total within sum of squares. The lower the total within sum of squares, then the tighter the clusters are together. The best way to analyze this plot is to select the 'elbow' point, meaning the point at which the marginal reduction of the within sum of squares lessens noticeably or where the plot begins to taper toward a minimum [7]. This can be a non-trivial task and it is possible that the plot will be too ambiguous to determine an 'elbow' point if the within sum of squares continues to get smaller as k gets larger. For this reason I chose a relatively large range of iterations to be sure that if there is an elbow it would be seen. At the same time if k is too large the clusters will have less interpretability, so the goal will be to find the best k somewhere in the middle.

Figure 1: Plot of K-means elbow method analysis: K from [1,30]



As can be seen from Figure 1, there is no distinct 'elbow' point but there are several points where the within sum of squares lessens significantly compared to the previous point. These points are 2, 3, 4, 7, 9, and 12. Figure 2 shows four cluster plots of k equal to 2, 3, 4 and 7.

Figure 2: Four cluster plots from four different iterations of the k-means algorithm



(a) K-Means cluster plot k=2

(b) K-Means cluster plot k=3

(c) K-Means cluster plot k=4

(d) K-Means cluster plot k=7

I determined which k value I would use for the rest of my project from the plots shown in Figure 2. I decided that I would not use k equal to 4 or 7 because the fourth cluster would be represented by only one user and therefore chose k equal to 3. The added benefit of choosing k of 3 is that this is the same number of clusters used in the Matragkas paper which will allow me to easily compare my findings with theirs.

4.2 Goal 2: Interpreting User Types

The output of the k-means clustering algorithm where k is 3 included three centroids. Centroids are the central points of each cluster that best represent the entire cluster and can be seen in the cluster plots of Figure 2 as the larger points numbered with the cluster number [11]. Each centroid contained an average value for each user variable that was representative of the variable values for the entire cluster. Below in Table 1 is a small subset of centroid variables which I chose before viewing the results. I predicted these variables were important and would be telling of the differences between the user types. The cluster number is the number of the cluster that the centroid comes from as seen in Figure 2b where k is 3. The users column is the number of users in that cluster accompanied by the percentage of users in the cluster out of the total number of users. The columns push, pull, watch, issue, and fork are all events, and their values are the average number of times the given event would occur for that centroid. These values are not the values of a single user but are theoretical values that a user of that type may have. The columns following, watchers, and repos (short for repositories) are the average number of followers, watchers, and repositories for that centroid respectively. Similarly these values are theoretical and not the values of a single user.

Table 1: Centroid values for selected variables

Cluster Number	Users	Push	Pull	Watch	Issue	Fork	Following	Watchers	Repos
1	969 (.0969%)	36.93	2.39	9.59	1.86	1.94	68.21	579.36	31.65
2	9015 (.9015%)	4.61	0.198	0.895	0.21	0.41	4.84	32.8	4.92
3	16 (.0016%)	85.25	57.28	10.81	11.75	32.62	884.875	7158.68	224.75

Looking at Table 1 at a high level, we can see that the second cluster contains about 90% of all the users and that on average these users have very little activity (within a three month time span) and have fewer followers, watchers, and repositories compared to clusters 1 and 3. Cluster 3 on the other hand contains only 0.0016% of the population and on average have very high activity and significantly more followers, watchers, and repositories compared to clusters 1 and 2. Lastly, the first cluster seems to be somewhere in between 2 and 3 on all variables. It appears that sometimes the activity is skewed more towards the behavior of the second cluster like in pull, issue, and fork, while in other activities they are skewed more towards the behavior of the third cluster like in push and watch. These user types are strikingly similar in nature to those described by Matragkas et al. and can accurately be described using the same names. The second cluster are the passive users who do not have much activity and make up for a large majority of the population, the third cluster are the core developers who are extremely active in all areas and make up a small minority, and the first cluster are the active users who have increased activity in development and communication areas but are otherwise similar to the passive users in others.

4.3 Goal 3: Predict User Behavior

Now that the user types have been realized and interpreted, I could use them as input for a model to try and predict user behavior. To do this, I used the list of events originally given to me by my mentors. As stated previously, each event in this list contained the actor's user id, the owner's user id, the repository id, the event type, the event date, and the event time. In cleaning the data I found that there were many owner ids that were not in my list of users. I realized that the list of users I received was a list of users that were actors, meaning that owners in the event list that were not also actors in the event list were not included in my user list. I was unable to get a list of the users I was missing and was forced to remove all events where the owner id was unknown. This left me with approximately 80,000 events to use for my model.
The model I chose to use, as suggested by my mentor, was the Random Forest classification algorithm. The random forest algorithm is a supervised machine learning algorithm also classified as an 'ensemble learning method' which means that this algorithm utilizes multiple learning algorithms or multiple iterations of a single learning algorithm to improve accuracy [14]. In this case, the random forest algorithm constructs many decision trees and takes the mode classification as the predicted output to correct the tendency of the decision trees to over-fit the data [3]. The algorithm I used was based on Breiman and Cutler's original algorithm in Fortran, and was used in supervised learning mode to classify the event types given in the event list [8].
In order to correctly use the algorithm, I first needed to make sure that all variables given as input were sufficiently independent. I looked at correlations between all the actor variables and removed one from each pair of variables correlated higher than 80% as recommended by my mentor. I did the same for the owner variables, and then the same for the remaining actor and owner variables together. I found that the variables I removed from the actor list were almost exactly the same variables removed from the owner list. I also found that most of the remaining actor and owner variables were highly correlated. I decided to determine if this was the case because the owner was the same user as the actor for many events or if there is truly a correlation between the actor and the owner. I found that in 99.5% of the events the actor and the owner are the same user, and in cases where the actor and owner are not the same user the actor and owner variables did not correlate highly. This creates a problem because my event data is skewed toward users that only act on their own repositories, but I can still create a model to predict the event type based on user information including the user type.
With this new information in mind, I decided to investigate the differences between the actor and the owner, and to see if either alone with the repository information and event information was more predictive of the event type or if they are equally as predictive. To do this I would train one model using the actor information and another using the owner information and compare them. Before doing this, my mentors recommended that I create a naive model to serve as a baseline for accuracy when analyzing

the other two models. We decided that one of the easiest models to create would be one that only ever predicts one event type, and since push events occurred most often in our data we decided that our naive model would predict every event to be a push event. We were able to construct a confusion matrix for this model manually and can be seen in Table 2. In a confusion matrix the rows are all the events that the model predicted and the columns are the true event types. By design this naive model never predicted any create, delete, fork, issue, pull, or watch events so all of those rows contain zeros, therefore the only row with values is the push row (again, because that was the only event type predicted). For the columns, each value in the column is the number of true column event types categorized as the row type. As an example, the issue column contains all of the true issue events and, because of the nature of this model, all of them were categorized as push events. If some of the issue events had been categorized as delete events then the delete entry of the issue column would be non-zero. Finally, the bolded diagonal are the number of correctly predicted event types.
After creating the naive model and calculating the overall accuracy, I could move on to creating the actor model (Table 3) and the owner model (Table 4). Comparing the accuracies, the actor model with an overall accuracy of 82.2% is more accurate than our baseline model, and the owner model with an overall accuracy of 81.6% is also more accurate than the baseline model but is less accurate than the actor model. And, if we compare the accuracies using the confidence intervals we can see that the actor model is significantly more accurate than the owner model.

Table 2: Confusion Matrix for the naive baseline model
Overall Accuracy: 72.4%

	Ground Truth							
Predicted	Create	Delete	Fork	Issue	Pull	Push	Watch	*Specificity*
Create	**0**	0	0	0	0	0	0	*0%*
Delete	0	**0**	0	0	0	0	0	*0%*
Fork	0	0	**0**	0	0	0	0	*0%*
Issue	0	0	0	**0**	0	0	0	*0%*
Pull	0	0	0	0	**0**	0	0	*0%*
Push	16569	2790	40	1314	693	**57628**	512	*72.4%*
Watch	0	0	0	0	0	0	**0**	*0%*
Sensitivity	*0%*	*0%*	*0%*	*0%*	*0%*	*100%*	*0%*	

5 Discussion and Conclusion

In this report I have shown the research that I conducted with Professor Beckage and Professor Austerweil at the University of Wisconsin. I achieved my goals of creating a descriptive model of GitHub users, analyzing the model and comparing my results to previous work, and creating predictive models of user behavior. Because this research experience was only ten weeks long, there were parts of this project where I wish I had been able to go into further detail and further analysis. For instance, I would have liked to get more user data to investigate the user type models with k greater than 3, understand how the model changed as k increased, and determine when we lose interpretablility at larger values of k. I would have liked to include analysis on more centroid values to get a better understanding of the behavior of each user type, and gotten a better understanding of the distributions for each variable to determine if the values were actually significantly different. I would have liked to go back to the original data and retrieved the missing owners from my user list to have a more complete sample of both users and events, and to give me more options in creating the predictive models. And finally, I would have liked to try a different machine learning algorithm to improve my model accuracies. Overall, I am pleased with the work I have done and am excited to continue to work in the field of modeling in the future. I would like to thank both of my mentors, all of the faculty I met at UW, all of my fellow students, and the CRA-W for giving me this amazing opportunity.

Table 3: Confusion Matrix for the actor model
Overall Accuracy: 82.2%
Accuracy 95% Confidence Interval: (0.8193,0.8247)

	Ground Truth							
Predicted	Create	Delete	Fork	Issue	Pull	Push	Watch	*Specificity*
Create	**9195**	1432	6	65	84	2266	45	*93.8%*
Delete	52	**343**	0	0	1	75	2	*99.8%*
Fork	0	0	**24**	0	1	2	2	*99.9%*
Issue	29	3	0	**446**	1	137	2	*99.7%*
Pull	6	2	0	0	**46**	15	0	*99.9%*
Push	7277	1010	8	803	560	**55122**	249	*54.8%*
Watch	10	0	2	0	0	11	**212**	*99.9%*
Sensitivity	*55.5%*	*12.3%*	*60%*	*33.9%*	*6.6%*	*95.7%*	*41.4%*	

Table 4: Confusion Matrix for the owner model
Overall Accuracy: 81.6%
Accuracy 95% Confidence Interval: (0.8139,0.8193)

	Ground Truth							
Predicted	Create	Delete	Fork	Issue	Pull	Push	Watch	*Specificity*
Create	**8904**	1283	3	77	83	2328	54	*93.9%*
Delete	156	**460**	0	0	1	130	1	*99.6%*
Fork	0	0	**7**	0	0	0	1	*99.9%*
Issue	32	7	0	**436**	8	160	2	*99.7%*
Pull	7	3	0	2	**37**	14	0	*99.9%*
Push	7457	1037	26	797	563	**54982**	320	*53.4%*
Watch	13	0	4	20	1	14	**134**	*99.9%*
Sensitivity	*53.7%*	*16.4%*	*17.5%*	*33.1%*	*5.3%*	*95.4%*	*26.1%*	

References

[1] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin. *Learning From Data: A Short Course.* AMLBook, 2017.

[2] A. S. Badashian and E. Stroulia. Measuring user influence in github: the million follower fallacy. In *Proceedings of the 3rd International Workshop on CrowdSourcing in Software Engineering*, pages 15–21. ACM, 2016.

[3] I. Barandiaran. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, 20(8), 1998.

[4] K. Blincoe, F. Harrison, and D. Damian. Ecosystems in github and a method for ecosystem identification using reference coupling. In *Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on*, pages 202–211. IEEE, 2015.

[5] K. Blincoe, J. Sheoran, S. Goggins, E. Petakovic, and D. Damian. Understanding the popular users: Following, affiliation influence and leadership on github. *Information and Software Technology*, 70:30–39, 2016.

[6] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on computer supported cooperative work*, pages 1277–1286. ACM, 2012.

[7] D. J. Ketchen and C. L. Shook. The application of cluster analysis in strategic management research: an analysis and critique. *Strategic management journal*, 17(6):441–458, 1996.

[8] A. Liaw, M. Wiener, L. Breiman, and A. Cutler. Package 'randomforest'. *R Documentation*, 2018. https://cran.r-project.org/web/packages/randomForest/randomForest.pdf.

[9] A. Lima, L. Rossi, and M. Musolesi. Coding together at scale: Github as a collaborative social network. In *Icwsm*, 2014.

[10] P. Loyola and I.-Y. Ko. Population dynamics in open source communities: an ecological approach applied to github. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 993–998. ACM, 2014.

[11] D. MacKay. An example inference task: clustering. *Information theory, inference and learning algorithms*, 20:284–292, 2003.

[12] J. Marlow, L. Dabbish, and J. Herbsleb. Impression formation in online peer production: activity traces and personal profiles in github. In *Proceedings of the 2013 conference on Computer supported cooperative work*, pages 117–128. ACM, 2013.

[13] N. Matragkas, J. R. Williams, D. S. Kolovos, and R. F. Paige. Analysing the 'biodiversity' of open source ecosystems: the github case. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 356–359. ACM, 2014.

[14] L. Rokach. Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1-2):1–39, 2010.

[15] J. Sheoran, K. Blincoe, E. Kalliamvakou, D. Damian, and J. Ell. Understanding watchers on github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 336–339. ACM, 2014.

[16] R. C. Team. R: K-means clustering. *R Documentation*, 1999. https://stat.ethz.ch/R-manual/R-devel/library/stats/html/kmeans.html.