Journal of Big Data

RESEARCH Open Access

# Library adoption in public software repositories

Rachel Krohn and Tim Weninger*

*Correspondence:
tweninger@nd.edu
Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, USA

**Abstract**
We study the the spread and adoption of libraries within Python projects hosted in public software repositories on GitHub. By modelling the use of Git pull, merge, commit, and other actions as deliberate cognitive activities, we are able to better understand the dynamics of what happens when users adopt new and cognitively demanding information. For this task we introduce a large corpus containing all commits, diffs, messages, and source code from 259,690 Python repositories (about 13% of all Python projects on Github), including all Git activity data from 89,311 contributing users. In this initial work we ask two primary questions: (1) What kind of behavior change occurs near an adoption event? (2) Can we model future adoption activity of a user? Using a fine-grained analysis of user behavior, we show that library adoptions are followed by higher than normal activity within the first 6 h, implying that a higher than normal cognitive effort is involved with an adoption. Further study is needed to understand the specific types of events that surround the adoption of new information, and the cause of these dynamics. We also show that a simple linear model is capable of classifying future commits as being an adoption or not, based on the commit contents and the preceding history of the user and repository. Additional work in this vein may be able to predict the content of future commits, or suggest new libraries to users.

**Keywords:** Information adoption, Software libraries, GitHub, Python, StackOverflow, Classification, SVM, Modelling, Git, Repository, Commit, Software development, Cognitive science, Text mining

## Introduction

The availability of rich online resources is an important source from which people discover and consume new information. Because of the scale and influence of the online marketplace of ideas there has been a particular focus on information diffusion dynamics [17] as well as information cascades of social hashtags, memes, and online news content [3, 4], including a recent focus on how these processes might be corrupted to spread false or misleading content [15]. In these studies, some piece of information (e.g., a hashtag, image, post) is typically tracked through a social or digital system. These large scale studies are usually sufficient to understand how far some information spreads and may be able to predict the future spread of information [7]. However, the question that many really want answered is, did the piece of information have any effect on the user? That is, did it change a user's behavior [6]?

It is important to distinguish between the spread of information and its adoption. Information can be spread and promoted, particularly in online spaces, with little thought given to the quality or accuracy of the content itself. Indeed, most information is shared on social media without the user ever reading the content of the post [14]. On the other hand, to adopt an idea one must incorporate the new information into their personal knowledge base so that it is reflected in their beliefs and behavior [44].

Online software repositories offer a unique setting for the examination of information adoption because the adoption of a software library requires a level of interaction beyond the simple sharing a social media post. Contrary to much of the data and systems used to collect and understand information adoption in the recent literature, software development processes result in an enormous volume of specific and well-documented signals of information adoption. This paper presents a thorough analysis of social software development patterns with the ultimate goal of developing a fine grained understanding of socially-mediated information adoption behaviors.

We make two main contributions in the present work. Based on a comprehensive analysis of user behavior, we show that library adoption commits are followed by higher than normal commit activity within the first 6 h. This implies that adoption is associated with increased cognitive effort. We also show that a simple linear model is capable of classifying commits as containing a library adoption or not, based on recent user and repository history.

## Related work

### *Information adoption and diffusion*

Information adoption refers to the process of accepting new knowledge assets into a personal knowledge base. The information source is often other people, or users in the case of online information adoption. Because of the complex mechanisms inherent in the information adoption process, researchers commonly rely on models to capture and simulate this behavior.

The processes by which humans adopt new ideas and information is rooted in cognitive science [31], which differentiates a person's understanding of language, for example, into receptive and productive vocabularies [37]. A *receptive vocabulary* are those words that a person understands when they are spoken or read. A *productive vocabulary* are those words that a person uses to express themselves in their own speech or writing. The information that a person reads represents a different cognitive engagement than the information that they produce. In this context we say that a person *adopts* a new word or phrase if they receive it and then use it for the first time, i.e., when it is transferred from their receptive vocabulary to their productive vocabulary [9, 10, 16].

Previous works rely on small user samples to examine information adoption in a particular context. For example, to determine the effect of argument quality or source credibility on information adoption in online communities, a sample of users could be surveyed about their recent activities [45]. A similar methodology can also be applied to adoption of opinions from online shopping reviews [8]. Researchers have also utilized this approach in the context of electronic word-of-mouth on social media [13]. Though this survey approach draws directly from users' experiences, the simplified models and small sample sizes limit the applicability of results.

Other researchers seek to model adoption at the user level. The Independent Cascade Model relies on cellular automata to track the state of each individual in the network over time [18, 19]. Diffusion thresholds [20] have been incorporated in this paradigm for greater simulation power [29]. Additional features, like real world nodes, user attention spans, and topic "stickiness" seek to mitigate the original model limitations. Overall, the user-based approach offers higher resolution than population-level models, but at the cost of increased complexity. This complexity makes approximation models for efficient computation attractive, particularly for large datasets [30]. The present work uses an adoption graph based on the Independent Cascade Model to help define the features of the adoption classifier.

Consideration of information adoption can reveal new approaches to existing problems in social networking. Determining influential users is a common challenge, and most approaches rely on the network structure or user attributes, like follower counts. One proposed alternative method incorporates temporal dynamics to produce a ranking of influential Twitter users based on information diffusion principles [33]. In this present work, we consider commit events temporally to trace all adoption events through users and repositories.

### *Big data analysis*

Existing machine learning techniques can be applied to a wide array of problems, such as using patient and population-level data in health informatics [24], applying deep learning models to AI tasks [38], or leveraging clustering algorithms to improve agricultural yield estimates [36]. For example, text mining can be used not only to extract libraries from source code, but also to determine the key words in a health study [26] or infer product characteristics from user reviews [39]. Often specialized analysis techniques are required because of the volume of data to be analyzed and the scalability problems that result [27, 41]. Processing frameworks for cluster computing, such as MapReduce, Hadoop, or Spark, allow for parallel processing of large datasets [32, 40, 47]. More recently, cloud computing offers researchers the capability to perform machine learning on big data without dedicated hardware [21].

In this work, we examine information adoption from the perspective of libraries used in public Python repositories on Github using both existing and novel analysis and machine learning techniques. In this study we rely on an SVM to predict future adoption activity based on commit features. By examining each commit individually in the context of the user's previous actions and the repository's history, we can determine whether or not an adoption event occurs in the commit. This is analogous to existing work that seeks to identify anomalies in datasets—in our work, an adoption can be viewed as an anomaly in the committing user's behavior. For example, classification techniques have been applied to user access behaviors on a secured system to identify abnormal behavior patterns as a malignant user [34]. Data profiles created by clustering algorithms have been used to detect anomalies in real-time data based on both the content and context of the information [22].

### Present work

In the present work, we measure adoption by observing changes in the information created by online users. Specifically, we focus on the adoption of software libraries within public Python software repositories. We tackle this question in a practical, quantitative way and make two primary contributions: (1) a fine-grained analysis of user behavior during and surrounding an adoption event as compared to normal activity, and (2) the development of a user model that predicts future adoption behavior. Additionally, we introduce a large dataset consisting of all the commit activity for roughly 250,000 Python repositories.

Our data comes from a large set of Github repositories that use the Python programming language. Within this data we identified users and their commits. Our "unit of information" is a software library $\ell$ (e.g., `numpy`, `pandas`, `tensorflow`) that is imported into a Python module through an `import` statement. For example, the `random` submodule of `numpy` can be imported using a command such as `from numpy import random` or `import numpy.random as rnd`. There are dozens of variations on Python import statements. We assume that each `import` statement is written deliberately and therefore define a library by its full library path as written. Therefore, submodules like `import numpy.random` are considered distinct from `import numpy` in the present work. The appeal of focusing on Python libraries is that they provide a clearly defined and rich information unit that is *adopted* rather than shared. To adopt the library, the user must understand its mechanics and be able to incorporate its functionality to produce new source code.

Although the two are commonly conflated, it is important to distinguish between Git and Github. Git is a source code management system allowing individual users to pull updated code from the shared software repository, merge them into a local codebase, and commit changes back to the shared repository. GitHub is a social and online hosting service for Git repositories that also provides an online mechanism for accessing, sharing, and discussing millions of repositories. The confluence of Github's online social system and Git's complete history of pull and commit behavior provides an unprecedented source of information on adoption behavior [28].

Public Github repositories only represent a subset of all Git repositories. Generally, Git is a specific and unique domain representing a single type of highly specialized activity—software creation. As such, we must temper conclusions to represent a case study of this particular domain. It will be interesting to see in future work if these findings generalize to other information contexts.

## Methods

### Data collection

We explore information adoption through a case study of software libraries in Python repositories found on Github. We collected repositories as follows. First, we issued a query to the Github search API for repositories written primarily in Python. Github returned repository IDs of the 1000 most popular Python projects on the site. We then found all Github users who made at least one commit to a repository in this set, and retrieved all of their Python repositories. This breadth-first style crawling was repeated, culminating in 259,923 repositories with 89,311 contributing Github users.

Of these, we were able to clone 259,690 Python repositories to disk; the remainder were made private or deleted between the time we crawled their repository address and performed the clone operation. Each cloned repository includes all files, branches, versions, and a complete edit history. These repositories $\mathcal{R}$ constitute about 13% of all Python repositories on Github as of July 2018.

Each repository contains files, commits $c$, and users. Each commit is marked with a timestamp $t$, a message, and a `diff` specifying lines that were added, edited, or deleted within each file. A commit also contains a name and email address defined by the user's local Git configuration. This configuration may or may not be correlated with a user's Github account; we make no attempt to associate the Github user account with the Git email or username. Because names may be ambiguous, we uniquely identify a user $u$ by their email address. A single user may contribute to multiple repositories. In total, we collected 170,413 unique Git users. An important caveat, however, is that users may change their Git configurations to alter their email address. It is unclear exactly how often this happens, but individuals may be represented as multiple users in our dataset.

We define two types of adoption. A *user-to-user adoption* occurs when the user $u$ pulls data from a repository, sees some new library $\ell$ committed by another user $v$, and then commits new code containing $\ell$. In this case we say that $u$ adopted $\ell$ from $v$. Note the pull and commit of library $\ell$ need not be consecutive. A *resource adoption* occurs when the user reads about $\ell$ from some other resource and commits code containing $\ell$ for the first time. It can be difficult to distinguish between user-to-user and resource adoptions because much of modern programming is aided by online resources like StackOverflow [48]. To incorporate these exogenous effects, we augment the Github repository data with the complete history of StackOverflow. In either case, an adoption is a global event, and a user may only adopt each library once; the repository committed to has no effect on whether the library was adopted or not.

## Library extraction

For each Python file in each repository, we retrieved its full history and examined all edited lines of each commit. An edit can be an addition marked by (+) or a deletion marked by (−). For each addition and deletion, we extracted any newly added or deleted Python libraries included in the commit. All library information is considered a unique package; for example, `from numpy import random` denotes the library `numpy.random`, which is considered a separate, albeit, closely related, library from `numpy.linalg` and even the top level library `numpy`, etc.

In total we extracted 29,060,288 commits over 331 months from August 1, 1990 through February 28, 2018. Of these, 15% contained at least one library addition and 6% contained at least one library deletion. Figure 1 (top) shows the growth in the number of commits made in our dataset per month including commits with at least one library addition or deletion. The number of libraries increased significantly over the past decade. Figure 1 (middle) plots the total number of libraries added and deleted as well as the number of unique libraries added and deleted per month.

We caution the reader not to conclude any trends of overall Python or Github usage from these plots. Although out dataset is large, it is biased towards more popular repositories, which are biased towards being older; this bias results from the first 1000 Python

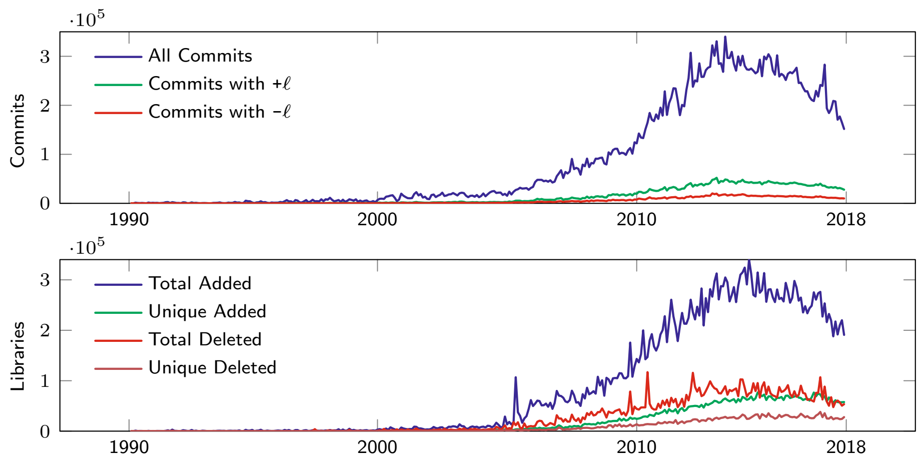

**Fig. 1** Top: monthly commits containing library additions or deletions. Middle: Monthly library additions and deletions. Caution: plots represent our sample of Python Github repositories; sampling is biased towards popular repositories and not representative of overall usage trends

repositories returned by the Github search API. Fortunately, this sampling bias will not effect the adoption analysis, since the dataset contains all commits by each of the included users.

## Library adoption

Based on their Git history, we model a user's receptive vocabulary $R_u$ and productive vocabulary $P_u$ as those libraries that the user has seen and those libraries that the user has used, respectively ($P_u \subseteq R_u$). When $u$ encounters a new library $\ell$ from new code obtained via a pull/merge of some repository, they may chose to ignore that new library or they may chose to consider the new library but not use it: $R_u \leftarrow \ell$. Occasionally, however, the user will choose to adopt that library by using it in a commit: $P_u \leftarrow \ell$. We assume that importing a library implies a conscious choice by the user to add that library's functionality to the repository.

Formally, we define a *library adoption* as any library $\ell \notin P_u$ committed by user $u$ to repository $r$ at some time $t$, such that $R_u \leftarrow \ell$ via a pull, merge, or clone operation from $r'$ (where $r'$ may or may not be $r$) at some time $t' < t$.

It follows that $\ell$ must have been added to $r'$ by some other user $v$ before $t'$. Therefore we say that $u$ adopted $\ell$ from $v$. In the present work we focus on these user-to-user library adoptions, and therefore use the term "adoption" to mean a user-to-user library adoption as defined here.

Simply put, a library adoption occurs when a user encounters some library and later uses it for the first time. This definition makes some assumptions. First, we assume that the adopting user received the library information through a pull, merge, or clone operation. Second, we assume that the adopting user is consciously aware that they are using the library. Finally, we assume the user has shared all of their commits publicly and has not been privately using the library.

The illustration in Fig. 2 captures the basic adoption behavior as time moves from left to right. Let $P_u = \{\emptyset\}$ initially. This figure illustrates that $u$ pulled updates from $r'$ and $r''$

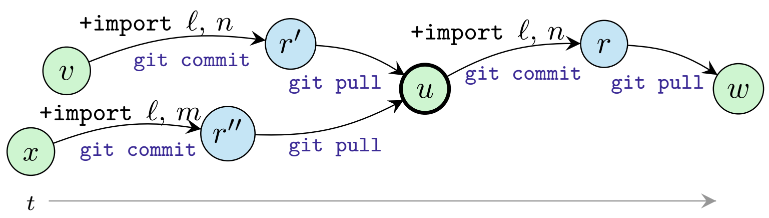

**Fig. 2** Example of how Git events like commit and pull spread information

containing libraries $\ell$, $n$, and $m$. Later $\ell$ and $n$ were adopted by $u$ as demonstrated by $u$'s commit containing +`import` $\ell$, $n$ to $r$. Conversely, despite pulling code containing $m$, $u$ did not adopt $m$ in this commit. Figure 2 also demonstrates that the libraries $\ell$, $n$, and $m$ were introduced to repositories $r'$ and $r''$ by users $v$ and $x$.

### *Adoption event extraction*

To create a dataset for adoption analysis, we extracted all library adoption events from the GitHub data using the following procedure. First, the added and deleted libraries were extracted from each commit, as discussed in "Library extraction" section above. Then, we stepped through all commits in time-sorted order, tracking the set of libraries used by each user. If a user commits a library that they have not previously used, this means the user *adopts* that library. Using this approach, we can obtain a dataset containing all library adoption events, for all users, across all cloned repositories. This adoption data is then used for the behavior change analysis.

During this data extraction process, we also track the libraries used in each repository and the number of commits adding each library. Similar values are tracked for each library, across all repositories. The supplementary data created during this temporal analysis is used to create the featureset outlined in Table 1, which is used for the adoption classification task.

## Adoption graph

If we consider a library adoption to be a directed edge between users, the collection of all adoptions forms a directed, edge-labeled multigraph. We call this edge-labeled multigraph the *adoption graph*, since each edge represents a user's adoption of a new library from another user [2]. The example multigraph illustrated in Fig. 3 represents the adoption graph from the events in Fig. 2. We represent the user-to-user adoption of $\ell$ as an edge $v \xrightarrow{\ell} u$ where the edge direction denotes the flow of information.

In the dataset collected from Github, we identified 512,097 commits containing at least one library adoption. These commits yielded 4,891,070 adoption-edges labeled by 248,141 unique libraries between 68,692 user-nodes. From this adoption graph we summarize various user roles and behaviors by their graph properties. The outdegree of a node represents how often a user's library usage was adopted by another user, akin to a user's leadership and innovativeness [46]. Conversely, the indegree of a node represents a user's propensity to adopt new libraries from other users, literally their *adapt*-ability [11, 44].

**Table 1 Feature set for the library adoption model**

| | |
|---|---|
| Commit features (C) | |
| C1 | # libs added by user |
| C2 | # libs updated since last commit |
| C3 | —C1 ∩ C2— |
| User features (U) | |
| U1 | Size of productive vocab $P_u$ |
| U2 | Size of receptive vocab $R_u$ |
| U3 | Time since last commit |
| U4 | Intra-commit duration in last 10% of commits |
| U5 | # Repos committed |
| U6 | # Repos committed in last 10% of commits |
| U7 | % commits with added libs |
| U8 | % commits with added libs in last 10% of commits |
| User–Library pair features (P) | |
| P1 | # times user has seen $\ell$ |
| P2 | # times user has seen any library |
| P3 | P1/P2 |
| P4 | # times user has seen $\ell$ in last 10% |
| P5 | # times user has seen any library in last 10% |
| P6 | P4/P5 |
| Library features (L) | |
| L1 | # commits adding $\ell$ |
| L2 | # users who have committed $\ell$ |
| L3 | # repos containing $\ell$ |
| L4 | Time since last commit of $\ell$ |
| L5 | Avg time between last 10% of commits adding $\ell$ |
| L6 | Avg time between last 10% of commits adding $\ell$ |
| StackOverflow features (S) | |
| S1 | # posts containing $\ell$ |
| S2 | # views of posts containing $\ell$ |
| S3 | # posts containing $\ell$ created in last 30 days |
| S4 | # views of posts containing $\ell$ created in last 30 days |

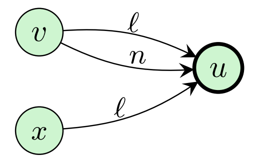

**Fig. 3** Example adoption multigraph

The in- and out-degree of the adoption graph forms a power law distribution with $k = 1.225$ and $k = 1.187$ respectively (not illustrated here). Such a long-tailed indegree distribution is surprising because of the cognitive effort required in adopting a new library, but the extreme cases are relatively few: only 17% of users have more than 100 adoptions and only 6% of users have more than 500 adoptions. The long-tailed outdegree distribution implies that there are a handful of users that are responsible for most of the library cross-pollination throughout Github.

### StackOverflow

A reasonable argument can be made that many Python libraries are not adopted by viewing pulled source code, but rather by searching the Web [42]. StackOverflow is one of the primary sources of programming guidance, so we compare adoption behavior against the popularity of related posts on StackOverflow. For this task, we identified 279,212 top-level libraries by removing submodule descriptors: `numpy.random` and `numpy` are both condensed as the top-level library `numpy`. Only 19% of adopted libraries also appeared in at least one post on StackOverflow.

Figure 4 plots, for each top-level library, how many views it received on StackOverflow against the number of commits adopting it. Some libraries like `error` and `help` have almost no adoptions, as we define them, but hundreds of millions of StackOverflow views. Conversely, `homeassistant`, an open-source home automation library, has hundreds of adoptions but only a dozen views on StackOverflow. We show a small positive correlation (Pearson $R = 0.16$, p $< 0.001$) between StackOverflow Views and Adoption Commits. However, its overall effect is tempered by the finding that a relative-few (22%) adoption events were of a library that appeared on StackOverflow.

## Results and discussion

### Behavior change near adoption events

Software systems are constantly changing. Python is a fast-evolving language with new libraries being invented frequently. Some libraries are in vogue only for a short period while others become ubiquitous and even change the culture of the language [43]. This process relies on individuals to adopt the new library and spread it to other repositories [1]. However, often the adoption of new technology comes with certain risks. Users implementing a new software library for the first time may not fully comprehend its use, or they may inadvertently introduce bugs into the system.

In this section our goal is to identify what change in behavior, if any, occurs when a user adopts a new library. For example, if a user adopted a new library that permitted the creation of a new feature or represents a marked change in the system, then we may

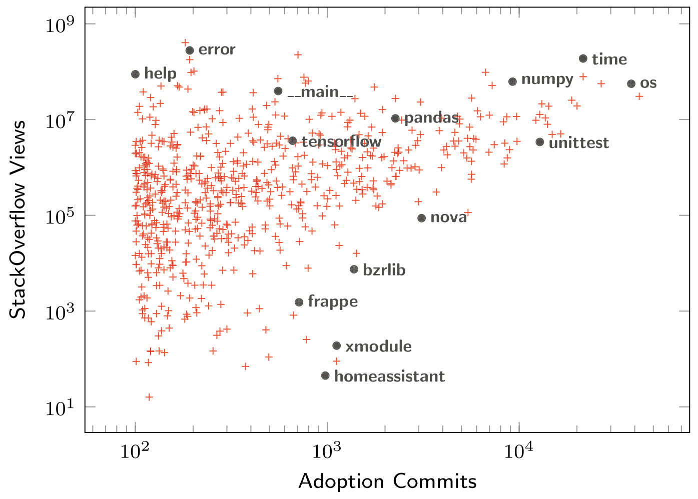

**Fig. 4** Adoption commits and StackOverflow views for top-level libraries with at least 100 adoptions

see increased activity before and after the adoption event [12]. Alternatively, if an adoption event has a high cognitive cost, then we might expect a decrease in activity after the adoption event [5]. Perhaps the adoption of a new library introduces a bug into the software system, which must be immediately fixed resulting in additional commits immediately surrounding the adoption event [25, 35].

Without interviewing individual users during adoption commits, it is difficult to ascertain the motivation for specific behaviors. To get a basic understanding of user behavior near an adoption event, we compare the frequency of commits immediately before and after an adoption event to the normal activity rate. We define three different activity rates: (1) the commit frequency for non-adoption sessions; (2) the commit frequency for non-adoption sessions where a library is added, but not adopted; and (3) the commit frequency during sessions where a library is adopted.

The first two rates are baselines, defined as follows. For each commit by each user, realign the commit time to $t = 0$ and compute the relative time difference (positive or negative) for all surrounding commits by the same user. These relative times are stacked into 5 min bins. The result demonstrates typical user activity surrounding each commit type. For the third activity rate, adoption commits are stacked separately.

Figure 5 shows all non-adoption sessions (orange) and the non-adoption sessions with added libraries (red). Non-adoption sessions with added libraries is necessarily a subset of all non-adoption sessions, thus there is slightly less activity. Both baselines are roughly symmetric around the central commit, indicating business as usual. The rise in activity directly before and the fall in activity directly after a commit indicates, generally, that activity surrounds activity.

The main result of this section is indicated by the blue line in Fig. 5. This represents commits (of any type, adoption or not) that occur within 6 h of an adoption event. We find that there are relatively few adoption sessions and that the activity preceding an adoption event is rather small. However, following an adoption the user, on average, becomes much more active than the pre-adoption baseline. These residual effects could represent various kinds of user behavior. We speculate that this surge of activity is due

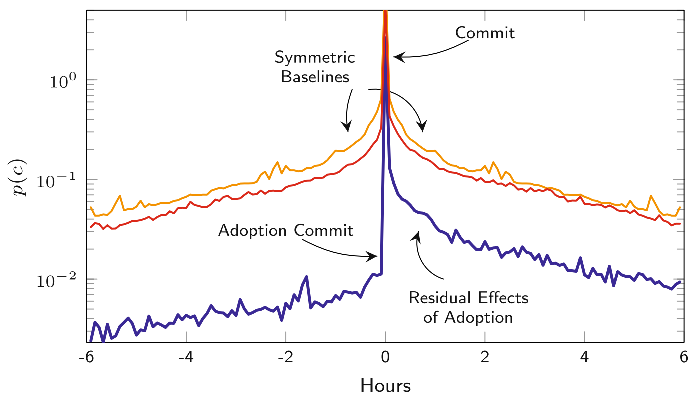

**Fig. 5** Activity rates for two baseline sessions (red and orange) versus adoption session rate (blue). The baseline sessions are roughly symmetrical surrounding the central commit indicating normal rates of commit activity. The residual activity directly following an adoption is significantly higher than the activity preceding the adoption, indicating anomalous behavior

to feature construction using newly adopted libraries, and bug fixes caused by the new libraries. Further investigation in needed to study this phenomenon.

## Adoption model

Our next task is to predict future adoptions by users [49]. To accomplish this task we must model the process by which users read, understand, and adopt information, as well as their receptive and productive vocabularies. We present a straightforward but surprisingly effective predictor that learns a model of user behavior. The goal is to model how a user will behave when shown new information—in this case new Python libraries. Will the user employ this new information to solve their programming challenges or will they ignore (or choose to not use) the new information?

The remainder of this section outlines the adoption model, and the results obtained from it. Additional details can be found in Appendix A.

### *Model data and features*

We transform the stream of Git pulls and commits into a classification task in the following way. For each commit we create a training instance for each library added in the commit as well as all libraries added in Git pull operations since the user's last commit (an example of this transformation is given in Appendix A). Note that this is a classification task and not a simulation task, i.e., we do not simulate future user commits. Rather, given some commit-library pair, we predict whether it is an adoption or not.

Each instance is described by a comprehensive set of features that imbue the model with the state of the user's receptive and productive vocabulary, recency, and the state of the repository at the time of the commit.

We define 5 feature categories: Commit (C), User (U), User–Library Pair (P), Library (L), and StackOverflow (S). Table 1 describes the features used in the model. Commit features describe information related to the libraries used in the current commit and updated since the user's last commit. User features describe previous user behavior. User–Library pair features encode information about how often the user has seen or previously interacted with a library. Library features describe the use of the library throughout the entire population. StackOverflow features denote the popularity of the library on the Web. Recency is encoded through the inclusion of features that only consider the last 10% of relevant commits, or the last 30 days of StackOverflow history.

### *Training and testing methodology*

We used the SVM implementation of SKLearn's SGDClassifier to train and test the model. Specific parameter values and training data preparations, including negative sampling, are outlined in Appendix A. Model performance is measured by area under the ROC curve (AUC). We plot the (mean) avg AUC and its 95% confidence interval for the 10 random days.

Our first task was to choose the proper amount of training data. Based on a series of tests using various training data set lengths (see Appendix A), the optimal training set interval is 1 month; we therefore use a training interval of 1 month for all further experiments.

### *Model tuning*

In addition to the training interval, we also tuned the training algorithm's parameters. Based on learning rate and regularizer experiments (see Appendix A), the L1 regularizer and a learning rate of 0.0001 produced the best results. Following experiments leave the learning rate set at 0.0001 and change the regularizer to L1.

Next we investigated the effect of the training data negative sampling rate. Because negative examples (non-adoption events) greatly outnumbered positive (adoption) events, we applied the common negative sampling strategy to the training set [23]. Experiments led us to select a negative sampling ratio of 2:1 (2 negative instances for each positive instance) for all following experiments (see Appendix A).

### *Feature ablation tests and model performance*

The overarching goal of this paper is to understand and model information adoption through the lens of library adoption in public software repositories. The modelling portion of this task is not complete without a thorough understanding of the information provided by the various features used to model the overall system. To do this we perform feature ablation tests; these tests purposefully hold out one or more features or feature sets in order to gauge their relative effectiveness and impact.

Using the parameters tuned in the experiments presented in Appendix A, we again performed classification tests for 10 random days. Figure 6 shows the performance of the ablation tests. These results show that the Commit features hold, by far, the most information: feature sets including Commit features perform significantly better than those without. The results also show that the User, User–Library pair, and Library feature sets carried only a modest amount of information. The User–Library pair feature set holds slightly more information than either the User or Library sets, suggesting that previous interactions between a user and library are more predictive than the user or library's overall history. Inclusion of the StackOverflow features do not significantly improve results, particularly if the feature set is already rich.

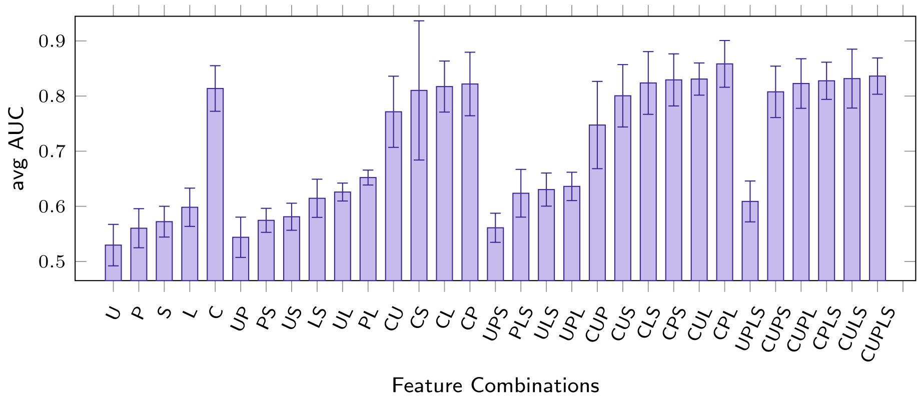

**Fig. 6** Feature ablation tests for predicting library adoptions with 95% confidence intervals. Results are ordered within feature categories from lowest to highest performance. Commit features (C) provide a large amount of information. The reduced feature set CPL has the highest performance on average, followed closely by the full feature set CUPLS

Why do Commit features carry so much information? Features C1 and C2 (from Table 1) denote the number of libraries added by the user in the current commit and the number of libraries that have been pulled by the user since the last commit respectively. Both of these activities are necessary for a library adoption to occur, since by our definition a library is only adopted if it is first received and later committed by a user. A large number of added libraries (C1) provides a greater chance for an adoption. However, because there is one testing instance per library-commit pair, a single adoption co-located with many non-adoptions could produce a high false positive rate. Overall, the number of libraries added within the commit contains a large amount of information about the number of adoptions. Indeed we find that the number of libraries added in the commit is strongly positively correlated (Pearson $R = 0.51$, $p < 0.001$) with the number of adoptions in the commit.

Similar to the results regarding user behavior near adoption events (Fig. 5), these results imply that adoptions are not isolated events. Instead, they occur during highly active periods and often include several other adoptions and additional commit activity.

As seen in Fig. 6, many feature sets achieve an AUC score of 0.8 or higher, indicating that these classifiers correctly identify adoptions 80% of the time. In fact, using Commit features alone is sufficient for this level of prediction. Including other features improves results, with a maximum AUC of approximately 0.85 for the CPL feature set. One month of training data is sufficient for this level of accuracy; therefore, the model only requires recent history for successful predictions.

## Main findings

We used an extensive dataset of commits, pulls, and pushes for public Python repositories to study library adoption in a network of users. First, we examined the activity rates of users during normal activity or surrounding an adoption event. As shown in Fig. 5, normal activity rates are higher than those before and after an adoption, suggesting that adoption is associated with some cognitive cost. Additionally, the activity rate immediately following an adoption is significantly higher than the baseline activity preceding the adoption. This implies that adoption a library is associated with a higher than normal cognitive effort, and this effort induces a temporary change on the user's behavior. The increased activity also suggests that library adoptions are not isolated events, and instead are followed by a flurry of commits. This anomalous behavior may be the result of feature construction or bug fixes induced by the adopted library, but further study is required to ascertain the true cause of this behavior.

Second, we trained a classifier to predict future library adoptions based on commit, user, library, and StackOverflow features. A single month of training data is sufficient to accurately predict 85% of library adoptions. As seen in Fig. 6, most of the model's power comes from the Commit features, which depend on the set of libraries committed by the user, and the set of libraries made visible to the user by pull operations. This again implies that adoptions are not isolated events, but instead tend to occur with other commit activity.

### Limitations

The present work is not without limitations. Though the dataset is large, it covers only a subset of the public Python repositories. Therefore, conclusions should not be taken as more than a case study of this particular domain.

Our commit analysis assumes that all library usage is purposeful and intentional, and that the user is making a conscious decision to add the library. In reality, users may be copying code from other sources, instead of directly importing the library. Even in these cases the library is still adopted by the user, they may just not be aware of it. This possibility should be considered when evaluating results.

## Conclusions

Analyzing the adoption of libraries within public software repositories has been a methodological challenge for many years. We introduced a new setting for studying cognitive behavior in a digital environment where it is possible to observe, in detail, the exposure to and adoption of new information. We introduced a large dataset of Git activity containing a thorough accounting of changes in `imported` libraries of Python repositories. This corpus allowed for the identification of library adoptions at the user and repository level.

Analysis focused on two concepts. First, we showed that user-to-user adoptions generated a higher than normal residual effect immediately following the adoption. User activity rates spike immediately following an adoption. Further study is needed to understand the exact mechanisms causing these dynamics. Second, we created a model that classifies commit-library pairs as adoptions or not. This model relies on not only the commit data, but also the history of the user and library in question. Data from StackOverflow was also incorporated. We showed that adoptions rarely occur in isolation; the inclusion of many libraries within a single commit is a strong indicator of new adoptions. This could suggest a common development strategy, but further study is required to determine the true cause of this phenomena.

### Future work

We alluded to many avenues for further study. The foremost task is to identify the specific causes, if any exist, of the residual effects of an adoption event. Perhaps the adoption of a new library results in buggy software that needs to be frequently fixed or reverted. Further examination of the commit data could reveal additional user activity patterns. Are most commits following an adoption additions, or deletions? Does a single adoption tend to induce further library adoptions? Is there a correlation between user session characteristics and adoption events? The usage of the newly-adopted library could also be examined. Do users tend to use new libraries often following the adoption, or is the initial usage an isolated event? Are newly adopted libraries used throughout the code, or only in a few places? The motivation for a library adoption could also be examined. Is the adopted library used to replace or upgrade existing functionality? Or is it adopted to create new features in the project? Further study of the Git data could answer these, and many more, questions about library adoption.

The analysis methodology used in the present work could also be applied to other contexts. For example, a similar methodology of information adoption could also be applied to language usage across the public Web. Do these residual effects occur in other adoption scenarios? Does adoption of a new word or term induce a similar activity spike? For example, when someone learns a new word do they tend to learn other new words simultaneously? Are newly-adopted words used frequently, or do users ease into their usage? The acceptance of bots in online communities could also be studied from the perspective of information adoption.

Finally, this data and methodology can be easily adapted to other areas of inquiry. For example, a similar methodology could be applied to other programming languages that use libraries, or to study ever-changing programming conventions across compiler versions. What impact does adopting a changed convention have on user activity patterns? Are certain changes easier to accept, with a lower cognitive cost? Beyond information adoption, this dataset could be used for more general studies on cooperative software development behavior. For example, the impact of team size on the development process could be examined.

Beyond additional research opportunities, the classifier could potentially be modified to serve as a library recommendation system. Based on a user's development history, and the history of the repositories they contribute to, a prediction system may be able to suggest potential libraries to the user. This could be a useful tool for new developers unfamiliar with the project or programming the language, since it would tend to suggest libraries already in use by connected users.

**Abbreviations**

$\mathcal{R}$: set of cloned repositories; *c*: set of commits for a repository; *u*, *v*: users; $\ell$: a Python repository; $R_u$: user *u*'s receptive vocabulary, or the set of libraries they have seen; $P_u$: user *u*'s productive vocabulary, or the set of libraries they have used; C: the set of commit features; U: the set of user features; P: the set of features defined by a user and library pair; L: the set of library features; S: the set of StackOverflow features; AUC: area under the ROC curve.

**Acknowledgements**

Not applicable.

**Authors' contributions**

RK collected and analyzed the GitHub adoption data, including development of all required software. TW played a pivotal role in guiding the study, and in results interpretation. Both authors read and approved the final manuscript.

**Funding**

The authors gratefully acknowledge support by the Defense Advanced Research Projects Agency (DARPA #W911NF-17-C-0094).

**Availability of data and materials**

The raw GitHub data is about 8 TB. The import diffs and full adoption graph are available from the author on reasonable request.

**Competing interests**

The authors declare that they have no competing interests.

## Appendix A: adoption model

### Data transformation

Consider the example below containing Git activity for a single user $u$. Let the productive vocabulary $P_u = \{\emptyset\}$ and the receptive vocabulary $R_u = \{\emptyset\}$ initially. Line 1 shows that $u$ imported a library $n$ organically; $n$ was not introduced to the user previously via a Git pull, so $n$ is not an "adoption" by our definition. After line 1 the receptive vocabulary and productive vocabulary of the user $R_u = P_u = \{n\}$. Lines 2 and 3 indicate that the user pulled source code containing newly included libraries $\ell$, $m$, and $p$. After line 3 $R_u = \{\ell, m, n, p\}$ and $P_u = \{n\}$. Finally, line 4 indicates that the user committed code with

$q$ and $\ell$. Because $\ell$ was introduced to the user via a Git pull event in line 2, $\ell$ is *adopted* by $u$ in line 4. On the other hand, because $q \notin R_u$ before line 3, $q$ is not a adoption by our definition. After line 4 $R_u = \{\ell, m, n, p, q\}$ and $P_u = \{\ell, n, q\}$.

| *t* | Event | Newly included libraries |
|---|---|---|
| 1 | Git commit | +`import` $n$ |
| 2 | Git pull | +`import` $\ell, m$ |
| 3 | Git pull | +`import` $p$ |
| 4 | Git commit | +`import` $q, \ell$ |

The time-ordered stream of Git pulls and commits are used to create the training data for the adoption classification model. For each commit, we create a training instance for each library added in the commit as well as all libraries added in Git pull operations since the user's last commit. Therefore, each commit results in one or more training instances, each corresponding to a single added or pulled library.

Returning to the example above, the commit at line 4 will generate 4 instances corresponding to $\ell$, $m$, $p$, and $q$—one instance for each unique library that was in a Git pull or commit since the user's last commit. Each instance is labeled as being adopted or not adopted in the current commit. Of the 4 instances corresponding to the commit on line 4, $p$, $q$, and $m$ are labeled as "not adopted," and $\ell$ is labeled as "adopted". Not all libraries that a user receives will necessarily be adopted; this is the behavior we aim to predict.

## Model framework

We used the SVM implementation of SKLearn's SGDClassifier to train and test the model. We limited hyperparameter exploration and instead focused on model data, using default values unless specified otherwise. This includes L2 regularization, and a learning rate of 0.0001. Negative instances (libraries received but not adopted) greatly outnumbered positive (adoption) instances, so we performed negative sampling to rebalance the training set. Unless otherwise specified, we maintained all positive training instances and randomly selected 2 negative training instances for each positive instance—a 2:1 negative sampling ratio. We did not perform any sampling on the test data. All tests were repeated 10 times, once for each of 10 random days; the same random days are used across comparable tests.

Model performance is measured by area under the ROC curve (AUC). This metric indicates how capable the model is at distinguishing between the binary classes: adoption and non-adoption events. The AUC always falls between 0 and 1, with 1 indicating perfect classification. We plot the (mean) avg AUC and its 95% confidence interval for the 10 random days used for testing.

## Training data interval

Before tuning model parameters or performing ablation tests, the first task was to choose the proper amount of training data. We assumed more training data is better. To test this assumption we selected 10 random days between February 1, 2017 and February 1, 2018. We generated hold-out testing data from Git activity for 30 days after each random date. We also generated training data sets of various lengths from Git activity immediately prior.

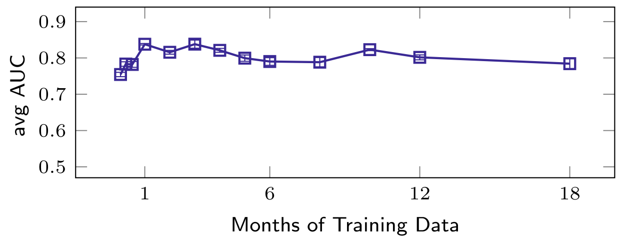

**Fig. 7** Model performance and 95% confidence intervals (which may be occluded by the marker) on 1 month of testing data as a function of the training data interval. 1 month of training data performs the best on average

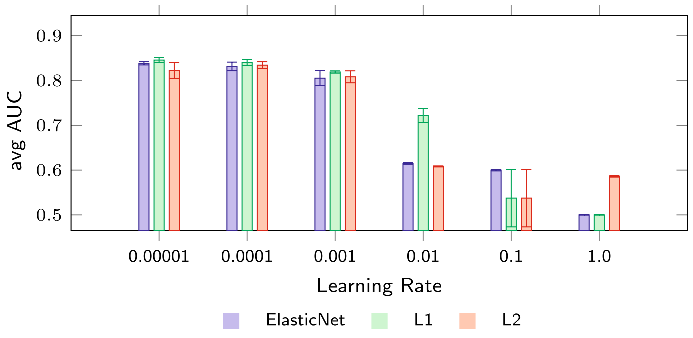

**Fig. 8** Model performance and 95% confidence interval as the learning rate and regularizer are varied. A learning rate of 0.0001 and the L1 regularizer yields the best performance

The results for various training intervals are plotted in Fig. 7. We show that model performance rises as the training interval increases from 1 day to 1 month. Beyond this point, performance decreases as more training data is included. The cause for this performance deterioration is unclear, but we suspect it is due to out-of-date information. Users may change their development behavior over time, and smaller training intervals are not diluted by old commit data. Overall the optimal training set interval is 1 month; all further experiments therefore use a training interval of 1 month, immediately preceding the 30 days of testing data.

## Classifier parameter tuning

To tune the classifier's training algorithm parameters, we varied the learning rate from 0.00001 to 1.0, and tested the L1, L2, and ElasticNet regularizers. All other settings were left at the default values. Figure 8 shows that the L1 regularizer performed the best, and a learning rate of 0.0001 yielded the most consistent and highest performance. All following experiments therefore use a learning rate set at 0.0001 and the regularizer set to L1.

## Negative sampling

Further tuning experiments examined the effect of the training data negative sampling rate. Figure 9 illustrates the model performance over various negative to positive label

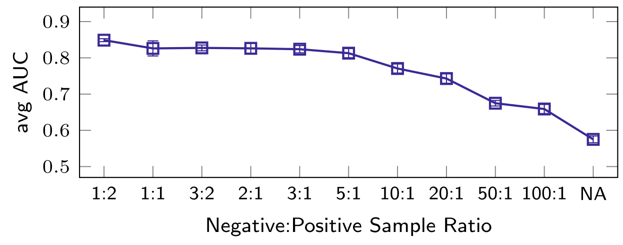

**Fig. 9** Model performance and 95% confidence interval as a function of the negative sampling rate applied to training data. Best avg AUC performance is found with a 1:2 negative-to-positive instance rate

ratios; NA represents no sampling. The avg AUC score indicates that performance decreases as the ratio of negative instances increases. Although the ratio of 1 negative to 2 positive instances (1:2) shows the highest avg AUC, other performance measures like accuracy, precision, and recall (not shown) are consistently higher with a ratio of 2:1. In the following experiments we leave the negative sampling ratio unchanged at 2:1.

Received: 7 March 2019 Accepted: 29 April 2019
Published online: 07 May 2019

## References

1. Anderson JR. A spreading activation theory of memory. J Verb Learn Verb Behav. 1983;22(3):261–95.
2. Bakshy E, Karrer B, Adamic LA. Social influence and the diffusion of user-created content. In: Proceedings of the 10th ACM conference on electronic commerce, ACM; 2009. p. 325–34.
3. Bakshy E, Messing S, Adamic LA. Exposure to ideologically diverse news and opinion on Facebook. Science. 2015;348(6239):1130–2.
4. Berger J, Milkman KL. What makes online content viral? J Market Res. 2012;49(2):192–205.
5. Burghardt K, Alsina EF, Girvan M, Rand W, Lerman K. The myopia of crowds: cognitive load and collective evaluation of answers on stack exchange. PLoS ONE. 2017;12(3):e0173,610.
6. Centola D. The spread of behavior in an online social network experiment. Science. 2010;329(5996):1194–7.
7. Cha M, Mislove A, Gummadi KP. A measurement-driven analysis of information propagation in the flickr social network. In: Proceedings of the 18th international conference on World Wide Web, ACM; 2009. p. 721–30.
8. Cheung CM, Lee MK, Rabjohn N. The impact of electronic word-of-mouth: the adoption of online opinions in online customer communities. Internet Res. 2008;18(3):229–47.
9. Cole J, Ghafurian M, Reitter D. Is word adoption a grassroots process? An analysis of Reddit communities. In: Social, cultural, and behavioral modeling. Springer, Washington, D.C., LNCS, 2017. p. 236–41. http://www.david-reitter.com/pub/cole2017wordadoption.pdf
10. Cole J, Ghafurian M, Reitter D. Word adoption in online communities. IEEE Trans Comput Soc Syst. 2019;6(1):178–88. https://doi.org/10.1109/TCSS.2018.2889493.
11. Czepiel JA. Word-of-mouth processes in the diffusion of a major technological innovation. J Market Res. 1974;11:172–80.
12. Danescu-Niculescu-Mizil C, West R, Jurafsky D, Leskovec J, Potts C. No country for old members: User lifecycle and linguistic change in online communities. In: Proceedings of the 22nd international conference on World Wide Web. ACM; 2013. p. 307–18.
13. Erkan I, Evans C. The influence of ewom in social media on consumers' purchase intentions: an extended approach to information adoption. Comput Hum Behav. 2016;61:47–55.
14. Glenski M, Pennycuff C, Weninger T. Consumers and curators: browsing and voting patterns on reddit. IEEE Trans Comput Soc Syst. 2017;4(4):196–206.
15. Glenski M, Weninger T, Volkova S. Identifying and understanding user reactions to deceptive and trusted social news sources. In: ACL. 2018. p. 176–81.
16. Goel R, Soni S, Goyal N, Paparrizos J, Wallach H, Diaz F, Eisenstein J. The social dynamics of language change in online networks. In: International conference on social informatics. Berlin: Springer; 2016. p. 41–57.
17. Goel S, Watts DJ, Goldstein DG. The structure of online diffusion networks. In: Proceedings of the 13th ACM conference on electronic commerce. ACM; 2012. p. 623–38.
18. Goldenberg J, Libai B, Muller E. Talk of the network: a complex systems look at the underlying process of word-of-mouth. Market Lett. 2001a;12:211–23.

19. Goldenberg J, Libai B, Muller E. Using complex systems analysis to advance marketing theory development: modeling heterogeneity effects on new product growth through stochastic cellular automata. Acad Market Sci Rev. 2001b;9(3):1–18.
20. Granovetter M. Threshold models of collective behavior. Am J Sociol. 1978;83(6):1420–43.
21. Hashem IAT, Yaqoob I, Anuar NB, Mokhtar S, Gani A, Khan SU. The rise of "Big Data" on cloud computing: review and open research issues. Inform Syst. 2015;47:98–115.
22. Hayes MA, Capretz MA. Contextual anomaly detection framework for big sensor data. J Big Data. 2015;2(1):2.
23. He H, Garcia EA. Learning from imbalanced data. IEEE Trans Knowl Data Eng. 2008;9:1263–84.
24. Herland M, Khoshgoftaar TM, Wald R. A review of data mining using big data in health informatics. J Big Data. 2014;1(1):2.
25. Huelser BJ, Metcalfe J. Making related errors facilitates learning, but learners do not know it. Mem Cogn. 2012;40(4):514–27.
26. Huh JH. Big data analysis for personalized health activities: machine learning processing for automatic keyword extraction approach. Symmetry. 2018;10(4):93.
27. Jacobs A. The pathologies of big data. Commun ACM. 2009;52(8):36–44.
28. Kalliamvakou E, Gousios G, Blincoe K, Singer L, German DM, Damian D. The promises and perils of mining github. In: Conference on mining software repositories. ACM; 2014. p. 92–101.
29. Kempe D, Kleinberg J, Tardos É. Maximizing the spread of influence through a social network. In: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM; 2003. p. 137–46.
30. Kimura M, Saito K. Tractable models for information diffusion in social networks. In: European conference on principles of data mining and knowledge discovery. New York: Springer; 2006. p. 259–71.
31. Labov W. Principles of linguistic change, volume 3: cognitive and cultural factors, vol. 36. New York: Wiley; 2011.
32. Landset S, Khoshgoftaar TM, Richter AN, Hasanin T. A survey of open source tools for machine learning with big data in the hadoop ecosystem. J Big Data. 2015;2(1):24.
33. Lee C, Kwak H, Park H, Moon S. Finding influentials based on the temporal order of information adoption in twitter. In: Proceedings of the 19th international conference on World wide web. ACM; 2010. p. 1137–8.
34. Lee S, Huh JH. An effective security measures for nuclear power plant using big data analysis approach. J Supercomput. 2018; 1–28.
35. Ma W, Chen L, Zhang X, Zhou Y, Xu B. How do developers fix cross-project correlated bugs? A case study on the Github scientific Python ecosystem. In: 2017 IEEE/ACM 39th international conference on Software engineering (ICSE), IEEE. 2017. p. 381–92.
36. Majumdar J, Naraseeyappa S, Ankalaki S. Analysis of agriculture data using data mining techniques: application of big data. J Big Data. 2017;4(1):20.
37. Melka F. Receptive vs. productive aspects of vocabulary. Vocabulary. 1997;33(2):84–102.
38. Najafabadi MM, Villanustre F, Khoshgoftaar TM, Seliya N, Wald R, Muharemagic E. Deep learning applications and challenges in big data analytics. J Big Data. 2015;2(1):1.
39. Ngoc TNT, Thu HNT, Nguyen VA. Mining aspects of customers review on the social network. J Big Data. 2019;6(1):22.
40. Ngu HCV, Huh JH. B+-tree construction on massive data with hadoop. In: Cluster computing. 2017. p. 1–11.
41. Reed DA, Dongarra J. Exascale computing and big data. Commun ACM. 2015;58(7):56–68.
42. Vasilescu B, Filkov V, Serebrenik A. Stackoverflow and Github: associations between software development and crowdsourced knowledge. In: International conference on social computing, IEEE, 2013. p. 188–95.
43. Venkatesh V, Bala H. Technology acceptance model 3 and a research agenda on interventions. Decis Sci. 2008;39(2):273–315.
44. Vilpponen A, Winter S, Sundqvist S. Electronic word-of-mouth in online environments: exploring referral networks structure and adoption behavior. J Inter Advert. 2006;6(2):8–77.
45. Watts SA, Zhang W. Capitalizing on content: information adoption in two online communities. J Assoc Inform Syst. 2008;9(2):3.
46. Webster CM, Morrison PD. Network analysis in marketing. Australasian Market J. 2004;12(2):8–18.
47. Wu X, Zhu X, Wu GQ, Ding W. Data mining with big data. IEEE Trans Knowl Data Eng. 2014;26(1):97–107.
48. Yang D, Martins P, Saini V, Lopes C. Stack overflow in github: any snippets there? In: 2017 IEEE/ACM 14th international conference on mining software repositories (MSR), IEEE. 2017. p. 280–90.
49. Yang J, Leskovec J. Modeling information diffusion in implicit networks. In: 2010 IEEE 10th international conference on data mining (ICDM), IEEE. 2010. p. 599–608.