Cross-Platform Software Developer Expertise Learning

by

Norbert Eke

A Thesis submitted to

the Faculty of Graduate and Postdoctoral

Affairs in partial fulfilment of

the requirements for the degree of

Master of Computer Science

in

Computer Science with Specialization in Data Science

Carleton University

Ottawa, Ontario

The undersigned recommend to

the Faculty of Graduate and Postdoctoral

Affairs acceptance of the Thesis

Cross-Platform Software Developer Expertise Learning

Submitted by **Norbert Eke**

in partial fulfilment of the requirements for the degree of

Master of Computer Science

Dr. Olga Baysal, Thesis Supervisor

Dr. Majid Komeili, Internal Examiner

Dr. Diana Inkpen, External Examiner

Dr. Ahmed El-Roby, Chair of Defence

Dr. Michel Barbeau, Department Chair

Carleton University

2020

Abstract

In today's world software development is a competitive field. Being an expert gives software engineers opportunities to find better, higher-paying jobs. Recruiters are always searching for the right talent, but it is difficult to determine the expertise of a developer only from reviewing their resume. To solve this problem expertise detection algorithms are needed. A few problems arise when expertise is put into application: how can developer expertise be defined, measured, extracted or even learnt? Our work is attempting to provide recruiters a data-driven alternative to reading the candidate's CV or resume.

In this thesis, we propose three novel topic modeling based, robust, data-driven techniques for expertise learning. Our extensive analysis of cross-platform developer expertise suggests that using multiple collaborative platforms is the optimal path towards gaining more knowledge and becoming an expert, as cross-platform expertise tends to be more diverse, thus creating opportunities for more effective learning by collaboration.

I dedicate this thesis to my grandmother, who taught me so much about life, and my mother, whom I love so much.

Acknowledgments

I, Norbert Eke, would like to express my sincere gratitude to my amazing supervisor, Professor Olga Baysal, for her continuous guidance, advice, and friendly discussions throughout my thesis. Our joint vision for this project, her continuous efforts in providing me with valuable feedback and support made this work successful.

I am very grateful to my family - my parents, siblings, grandparents, aunts and uncles. They gave me a tremendous amount of support by motivating, encouraging me and keeping me optimistic in difficult moments.

I am thankful to my friends, and Diana Lucaci for their patience, optimism, support, and encouragement when I needed it the most, especially during the writing process.

I would like to acknowledge the advice of several professors: Prof. Majid Komeili and Prof. Ahmed El-Roby from Carleton University, as well as Prof. Abram Hindle from the University of Alberta. I would also like to be grateful for the help of my colleagues in the Data Science Lab at Carleton University, and the 20 students from COMP5117 (Mining Software Repositories), Fall 2019 semester who volunteered for the *Expertise Ground Truth Human Annotation Study.* Without their professional opinion on expertise, the ground truth study would have not been possible.

Table of Contents

List of Tables

List of Figures

Nomenclature

Abbreviations

This thesis uses some common abbreviations existing in the domain of computer science. The following table lists the abbreviations and their meaning :

Abbreviation	Name
GB	Giga-Byte
MB	Mega-Byte
SO	Stack Overflow
GH	GitHub
CSV	Comma-Separated Values
GDPR	General Data Protection Regulation
LDA	Latent Dirichlet Allocation
RQ	Research Question

Chapter 1

Introduction

In this first chapter we introduce *developer expertise* in Section 1.1, then uncover the motivation behind this work in Section 1.2. Motivation will be followed up by our *research questions* that we seek to find answers to in Section 1.3, then our *contributions* will be following in Section 1.4, and lastly a *thesis structure* will conclude this chapter in Section 1.5.

1.1 Developer Expertise

In today's world, one can argue that software development is a competitive field. Technology giants such as Google, Facebook, Amazon, Microsoft and Apple have virtually unlimited resources to achieve their goals by releasing newer, faster, and better software than their direct competition. One might ask what is their secret? We subjectively believe that one of their secrets lies in the hard-working, talented people that they hire. Ian Sommerville writes in a book about *Software Engineering* that bad hiring decisions are one of the main risk factors when aiming to build a successful software [1]. Smaller companies and start-ups do not start with equal chances against the technology giants, as they do not possess the same financial and human resources. We argue that in order to maximize the likelihood of building the best possible software, the key is in *hiring the right expert developers* who are a perfect fit for their position. The technology giants have thousands, even millions of applicants each year. For instance, in 2014 it was reported that Google received about three million applications per year [2], and only the best of the best (*the experts*) get hired. Becoming an expert developer is probably the goal of every one of those applicants. Finding the right expert for a job opening is certainly the goal of every recruiter and HR manager, but it is difficult to determine the actual expertise of developers just from reviewing their CVs or resumes.

1.2 Motivation

A few problems arise when the concept of expertise is put into application: how does one know if they are an expert in something, how can developer expertise be defined, measured, extracted or even learnt? The main motivation of this work is answering some of the above questions that come up as barriers to the implementation of expertise-driven applications. Previous research work had similar motivations, as the problem of finding experts, defining and learning developer expertise from data is a popular problem in software engineering. Mockus and Herbsleb [3] presented one of the first tools attempting to quantify expertise performance. Baltes and Diehl [4] noted how difficult it is to propose objective quantitative measures of expertise. Teyton et al. [5] attempted to detect third party library experts in open source software communities. The authors were motivated by the fact that modern software depends on numerous third-party libraries and developers require substantial expertise to maintain their software systems.

One might ask, why research developer expertise when social platforms such as *LinkedIn*[1] or job search engines such as *Indeed*[2] exist. *LinkedIn* recommends job

[1] `https://www.linkedin.com`

[2] `https://www.indeed.com`

advertisements to users, facilitates faster and more convenient ways to apply for positions, and allows recruiters to post job opportunities and have applicants apply for positions. *LinkedIn* users can enter their previous work experiences, display their top skills, and most importantly the online platform will compute similarity scores between pairs of users and job advertisements, then rank the user compared to other candidates and recommend top candidates for the recruiter to hire. One might say *LinkedIn*'s recommender systems act as an automated expert candidate detector, but that is not the case. One essential factor on *LinkedIn* is that users self-report their education, previous experience, and top skills. This means that users can lie about their education or previous work experiences, exaggerate their skills, neglect some skills, and even add skills that they do not possess. Thus, *LinkedIn*'s endorsements are as accurate as the information self-reported by users, and it is questionable how much a recruiter should rely on the candidates recommended by *LinkedIn* actually being the top candidates.

1.2.1 Developer Expertise in Collaborative Platforms

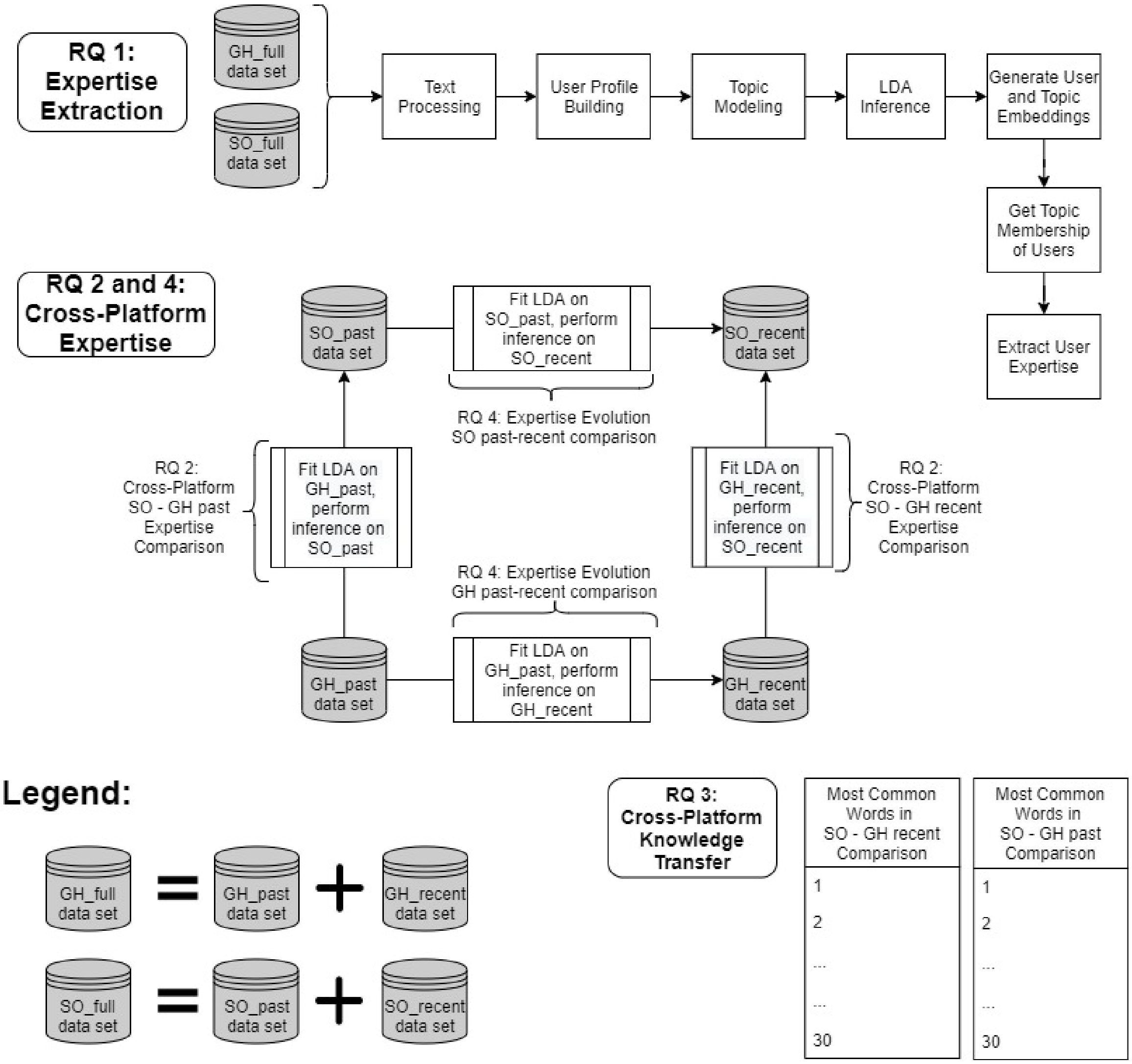
Greene and Fischer [6] have expressed similar concerns about *LinkedIn*, and suggested expanding the candidate search to developer-oriented platforms, such as GitHub[3] and Stack Overflow[4], noting that it could become valuable for the recruitment process. Greene and Fischer also mentioned that identifying and hiring candidates with proper skills is necessary for a software project to succeed. Hauff and Gousios [7] had very similar motivations in their work suggesting matching job advertisements to developers, based on developer activity extracted from their GitHub profiles. Tian et al. [8] had very similar intentions when they proposed the recommendation of expert developers using user behaviour and contributions on GitHub and Stack Overflow.

[3] `https://github.com/`

[4] `https://stackoverflow.com/`

One can notice a pattern of leveraging data from collaborative platforms, such as GitHub and Stack Overflow, to learn the expertise of developers based on their software development related activities on the platform. These collaborative platforms are online communities containing question answering, discussion and shared source code related activities [9]. These platforms provide a way for developers to communicate and collaborate with their peers while gaining and sharing knowledge. Tian et al. [8] state that the success of question answering platforms highly depends on the number of expert users who contribute regularly by answering questions, suggesting more efficient algorithms and pointing out flaws in source code. Greene and Fischer [6] explained that collaborative platforms are an excellent source of data for identifying the right candidate for a job opening, as content on such platforms tend to contain large amounts of technical information, thus developer interest and expertise can be inferred with the help of text mining algorithms and language models. Zhang et al. [10] confirmed Greene and Fischer's claims, as they stated that user behaviour is a rich source of data about the software development process. Tian et al. [8] explored the benefits of cross-platform expert recommendation systems, and they referred to GitHub and Stack Overflow as the primary data source for such a system. They stated that such a cross-platform system could shed a light on what professional activities are conducted by various types of users.

While there are numerous advantages of being part of communities on collaborative platforms, Wang et al. [9] outlined a few challenges that these platforms have: 1) Popular collaborative, question answering platforms get thousands of new questions daily, while there could be millions of already existing questions on the platform. The very large volume of questions makes it difficult for a contributor to decide which questions to answer; 2) Contributors tend to have various interests and expertise in different areas of their field, so the quality of the answers could be inconsistent, and 3) some questions do not receive an answer on the same day, thus some users will have to wait for long periods to get a satisfactory answer. This being said, the main motivations behind learning developer expertise from collaborative platforms are: 1) to find out whether developer expertise could be accurately extracted and learnt from collaborative platforms, 2) to investigate if users maintain similar expertise profiles across multiple collaborative platforms, and 3) to develop one or more novel techniques to extract developer expertise areas from collaborative platforms.


RQ 1: Expertise Extraction
GH_full data set
SO_full data set
Text Processing
User Profile Building
Topic Modeling
LDA Inference
Generate User and Topic Embeddings
Get Topic Membership of Users
Extract User Expertise
RQ 2 and 4: Cross-Platform Expertise
SO_past data set
Fit LDA on SO_past, perform inference on SO_recent
SO_recent data set
RQ 4: Expertise Evolution SO past-recent comparison
RQ 2: Cross-Platform SO - GH past Expertise Comparison
Fit LDA on GH_past, perform inference on SO_past
Fit LDA on GH_recent, perform inference on SO_recent
RQ 2: Cross-Platform SO - GH recent Expertise Comparison
RQ 4: Expertise Evolution GH past-recent comparison
GH_past data set
Fit LDA on GH_past, perform inference on GH_recent
GH_recent data set
Legend:
GH_full data set = GH_past data set + GH_recent data set
SO_full data set = SO_past data set + SO_recent data set
RQ 3: Cross-Platform Knowledge Transfer

Most Common Words in SO - GH recent Comparison	Most Common Words in SO - GH past Comparison
1	1
2	2
...	...
...	...
30	30

Figure 1: Our research roadmap.

1.3 Research Questions

Figure 1 presents the overall research roadmap that connects our research questions with the methodology we followed. We address the following research questions in our work:

- **RQ1: How can we extract the major expertise areas of Stack Overflow and GitHub users? How do expertise trends compare on Stack Overflow and GitHub?**

  This research question is addressing the unknown behind quantitative developer expertise: how to define, extract, measure and rank expertise of Stack Overflow and GitHub users. This research question is extremely important, as it drives the search for novel, data-driven techniques that can accurately extract expertise from software artifact related data. Once expertise has been extracted, a follow-up question will focus on comparing the newly discovered trends from Stack Overflow and GitHub.

- **RQ2: How similar are developer expertise profiles in two different collaborator platforms, Stack Overflow and GitHub?**

  This research question is intended to provide a comparative expertise analysis between the two different collaborative platforms, i.e., Stack Overflow and GitHub, to allow software engineers to understand how similar or different are the expertise profiles on the two platforms. This insight could expand our understanding of cross-platform developer expertise.

- **RQ3: What knowledge is transferable from one platform to another?**

  This research question builds further analysis on top of RQ2 by analyzing what are the similarities between the two platforms and what kind of knowledge is the most transferable from one platform to another. Getting insight into knowledge transfer would allow software developers to understand what skills could be learned by using both Stack Overflow and GitHub.

- **RQ4: How much does developer expertise evolve on Stack Overflow and GitHub?**

  This research question focuses on the change in expertise over time and it is intended to explore and better understand the expertise evolution on Stack Overflow and GitHub. This final research question is important in order to understand if people are improving their skills, or whether they gain new skills.

1.4 Contributions

The contributions made in this line of research are 7-fold:

- Development of three novel techniques to extract developer expertise topics from Stack Overflow and GitHub (*RQ1*)
- Analysis of developer expertise trends on Stack Overflow and GitHub (*RQ1*)
- Comparison of developer expertise across two collaborative platforms (*RQ2*)
- Empirical evidence about knowledge transfer between two collaborative platforms (*RQ3*)
- Analysis of developer expertise evolution trends from two collaborative platforms (*RQ4*)
- Collection of developer expertise ground truth data set
- Development of four new data sets (*GH-past and recent, SO-past and recent*) by aggregating Stack Overflow and GitHub data.

1.5 Thesis Organization

The structure for the rest of this thesis is as follows: Chapter 2 includes Background (Section 2.1) and Related Work (Section 2.2), while Chapter 3 describes our Methodology. Results are presented in Chapter 4, followed by Discussion in Chapter 5. The thesis concludes with Chapter 6 outlining Summary of Contributions (Section 6.1) and Future Work (Section 6.2).

Chapter 2

Background and Related Work

This chapter outlines the necessary background knowledge needed for this work in Section 2.1, then all related work is listed in Section 2.2.

2.1 Background

In this section, we first introduce GitHub and Stack Overflow in Section 2.1.1 as the two most popular collaborative platforms in software engineering. In Section 2.1.2, we outline the best practices of pre-processing textual data containing software artifacts. Section 2.1.3 introduces word embeddings, while Section 2.1.4 introduces topic modeling, and we offer a lengthy background on LDA models. Background concludes with a brief introduction of all evaluation metrics used throughout the thesis in Section 2.1.5.

2.1.1 GitHub and Stack Overflow

GitHub (GH) is the most popular social coding platform that allows massive worldwide collaboration for software developers on open source projects. In 2018, it was reported that GitHub had a total of more than 100 million repositories [11], but search results from March 2020 indicate only over 28.6 million repositories are public[1]. As of March 2020, GitHub search results report over 37.7 million users[2]. As early as 2014 GitHub was already called *"the largest code hosting website in the world"* by Gousios et al. [12]. GitHub also reported that 10 million new users joined in 2019. What is even more incredible that a total of 1.7 million students have learned to code on GitHub in 2019, which is 55% more than in the previous year [13]. These statistics show how important GitHub is for the software development process. Hauff and Gousios [7] mentioned that a developer's GitHub profile could become of interest to potential employers and recruiters, as they could learn more about a candidate's skills and interests.

Stack Overflow (SO) is the most popular question answering website for software developers, providing a large number of code snippets and free-form text on a wide variety of topics. In a recent public data dump from December 9th 2018 [3], Stack Overflow listed over 42 million posts from almost 10 million registered users. Similar to other software artifacts such as source code files and documentation, text and code snippets on Stack Overflow evolve over time. An example of this is when the Stack Overflow community fixes bugs in code snippets, clarifies questions and answers, and updates documentation to match new API versions. Studying Stack Overflow, its posts, users, and trends will shed some light on the evolution of expertise in such software artifacts. Furthermore, mining Stack Overflow has been a very popular research topic in Software Engineering over the past few years, thus Section 2.2.1.

2.1.2 Text Processing of Software Engineering Data

Textual data on Stack Overflow and GitHub contains a large variety of domain-specific (i.e., software engineering) terminology mixed with source code. Stack Overflow posts

[1] `https://github.com/search?q=is:public`

[2] `https://github.com/search?q=type:user&type=Users`

[3] `https://zenodo.org/record/2273117`

contain rich software artifacts such as code snippets, text blocks explaining problems and solutions using text and source code, comments which can be associated with a question or an answer, and hyperlinks to references such as API documentation or a publication explaining an answer. Liao et al. [14] in their study on GitHub mention that developers on GitHub debate "project bugs, enhancements, and tasks in issue discussions". The authors stated that a text corpus built on GitHub data is most likely to contain code snippets, error messages and warnings, technical jargon and casual natural language, which theoretically should reveal someone's expertise.

We can conclude that textual data on both GitHub and Stack Overflow captures a variety of aspects of software development, but its technical terminologies, the mix of text and source code make it difficult to extract the proper textual data to be analyzed. When performing text pre-processing on both sources of data, we need to find the right balance between more than a dozen pre-processing techniques to clean up the data just the right way, without removing software engineering domain-specific words and damaging the contextual details of software artifacts. Deciding on a precise set of techniques for the pre-processing routine can be a challenging task, thus we conducted extensive research on what text pre-processing techniques have been previously used for analyzing GitHub and Stack Overflow data.

Tian et al. [15] pre-processed textual data by performing tokenization, stop-word removal and stemming. They also split function names from cleaned up source code related keywords in the text processing phase.

Campbell et al. have also addressed the challenge of data pre-processing. In their book chapter [16] on extracting topics from software engineering data, they say that generally those who use LDA apply the following pre-processing steps: perform lexical analysis, optionally remove stop words and performing stemming, then build vocabulary, optionally filter out uncommon and frequently used words, and lastly create a bag-of-words representation of text.

Treude and Wagner [17] have performed similar but separate pre-processing routines on Stack Overflow and GitHub data. Their Stack Overflow data cleaning routine consists of removing line breaks, code blocks, all HTML tags, replacing HTML symbols and strings indicating special characters with their corresponding character, and replacing sequences of white-space with a single space. Their GitHub pre-processing routine is the same as the Stack Overflow data cleaning routine, with a few extra steps, such as removing vertical and horizontal lines, code comments, characters denoting sections headers, characters that indicate formatting or links.

Efstathiou et al. [18] converted text from Stack Overflow into lowercase, removed code snippets and HTML tags, then performed a punctuation removal, carefully keeping symbols that could appear in software artifacts, such as "+" and "#" in "C++", and "C#".

Boyd-Graber et al.'s book chapter [19] recommends HTML symbol and stopword removal, then normalizing strings by converting to lower case and applying any kind of stemming algorithm, then applying tokenization and any kind of phrase detection algorithm as the proper pre-processing routine before fitting topic models on textual data.

Liao et al. [14] analyzed issue discussions on GitHub, and in their pre-processing steps they filtered out code-specific language, which was enclosed in HTML tags. The authors then removed short posts contained less than five tokens and distinguished between data generated by developers who never committed code to a project and those who did.

2.1.3 Vector Representation of Words using Word Embeddings

In order to represent words in a quantitative way, we need to have a vector representation of words. One of the most popular deep learning models that allows the computation of vector representations of words is Google's Word2Vec [20]. In 2013, Mikolov et al. [20] introduced the Word2Vec model as a choice of two different architectures named CBOW (Continuous Bag-of-Words) and Skip-gram for learning vector representation of words using neural networks. The Word2Vec model is a feed-forward neural network with one hidden layer, that takes textual data in the form of one-hot encoding as input and learns high dimensional *word embeddings* (also called word vectors) as output. The difference between the above mentioned architectures is that CBOW learns word embeddings by predicting the current word based on its context, while Skip-gram learns by predicting the surrounding words given a current word. Other popular vector representation of words deep learning models include GloVe [21] and FastText [22].

Such deep learning models are trained to reconstruct the linguistic context of words by placing the vectors of words used in the same context close to each other. Word embeddings are widely used in the field of Natural Language Processing (NLP) as they are known for capturing semantic similarities and contextual details of words in textual data. If dimensionality reduction is performed on the word embeddings, these semantic similarities can be visualized in low dimensional *word vector spaces*. An example of such vector space can be seen in Figure 2.

[5] `https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/`

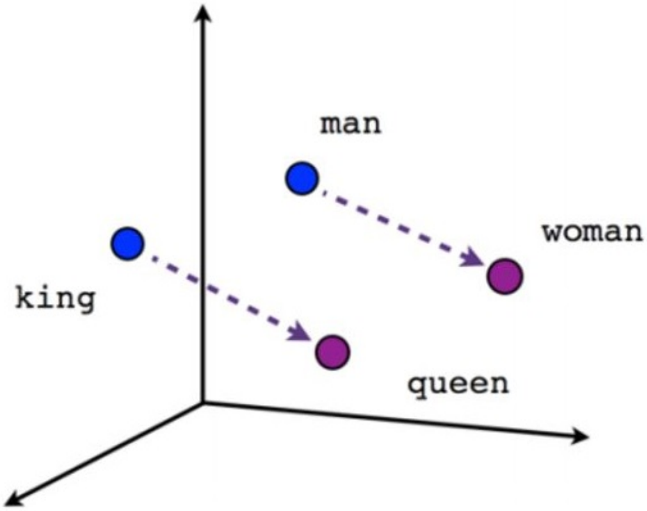


Figure 2: Photo Credit to [23]. Three dimensional vector space[5], which contains the vectors 'Man', 'Woman', 'King', 'Queen'. 'Man' and 'Woman' are related to each other the same way as 'King' and 'Queen' are related to each other, since the vectors in between them describe the male-female relationship.

2.1.4 Topic Modeling

Topic modeling is a technique used in the field of text mining to discover hidden patterns in a collection of textual documents. These hidden patterns are called *topics* and they can be modeled using distributions. A *topic* can be defined as a collection of co-occurring words that have been grouped together by a topic modeling algorithm.

Topic models are highly popular unsupervised statistical models in machine learning, as they provide a way to find structure in unstructured textual data. The most popular use case scenarios for topic models are text clustering and information retrieval tasks, where the goal is to discover the structure and extract insight from text documents. Examples of such an application could be the use of topic modeling to group similar words into topics, thus, forming text clusters. Another sample application could be using topic modeling to detect trends in online documents and build a recommender system for users to find similar online content.

The most used topic modeling algorithms are Latent Dirichlet Allocation (*LDA*), Latent Semantic Indexing (*LSI*) and Non-Negative Matrix Factorization (*NMF*). The focus of this section will be on *LDA* models, as this is the model used throughout this thesis work.

2.1.4.1 What is an LDA model?

LDA is a generative probabilistic Bayesian topic model. This means that LDA observes probability distributions in input data and generates topics based on an underlying distribution of the latent (hidden) variables. LDA being also a Bayesian model, it allows Bayesian inference methods to be performed on the model. LDA models need as input a collection of documents in the form of a *Document-Term Matrix* containing the number of occurrences of each word j in each document i, and a few hyper-parameters.

LDA models come with assumptions. The first assumption is that each input document has multiple topics. This is also referred to as mixtures of topics. Furthermore, each document can be modeled as a discrete probability distribution over some number of latent (hidden) topics. The second assumption is that a topic is a collection of words with a discrete probability distribution over all unique words contained in the input documents, which is also referred to as a vocabulary of words. Lastly, the third assumption is that the number of topics is fixed and needs to be specified by the user, as LDA models cannot infer the number of topics from the input collection of documents.

A trained LDA model outputs two matrices: *Topic Document Matrix* and *Term Topic Matrix*. The *Topic Document Matrix* contains probabilities of each document generating a specific topic. The *Term Topic Matrix* contains probabilities of each topic generating a specific word from the vocabulary. These two matrices can be further processed and transformed into a weighted list of topics representing every document. Furthermore, each topic contains a list of words, which describe the topic.

Large collections of unstructured textual data can be summarized, clustered, linked and even pre-processed using an LDA model and its rich output [16].

2.1.4.2 How does an LDA model work?

Apart from input collection of documents, LDA has four parameters:

1. α represents a document's prior topic distribution, and it is sampled randomly from a Dirichlet distribution with α hyper-parameter. This parameter is also referred to as the prior document-topic density. Larger values of α result in documents containing more topics, while smaller values of α result in documents containing fewer topics.

2. β represents a topic's prior word distribution, and it is sampled randomly from a Dirichlet distribution with β hyper-parameter. This parameter is also referred to as the prior topic-word density. Larger values of β result in topics containing a larger number of words from the corpus, while lower values of β result in topics containing fewer words from the corpus.

3. K represents the number of topics to be extracted from the input collection of documents.

4. The number of iterations represents the maximum number of iterations allowed for the LDA algorithm to convergence.

LDA assumes that it receives a collection of D documents, each of length L_i as input. Being a generative probabilistic model, LDA has a generative process, which can be described in the following steps:

1. For each topic k in $\{1, ..., K\}$ draw a word distribution $\phi_k \sim Dir(\beta)$

2. For each document d in $\{1, ..., D\}$ draw a topic distribution $\phi_d \sim Dir(\alpha)$

3. For each word i of document d draw a topic distribution $z_{d,i} \sim Multinomial(\phi_d)$ and a word distribution $w_{d,i} \sim Multinomial(\phi_{z_{d,i}})$

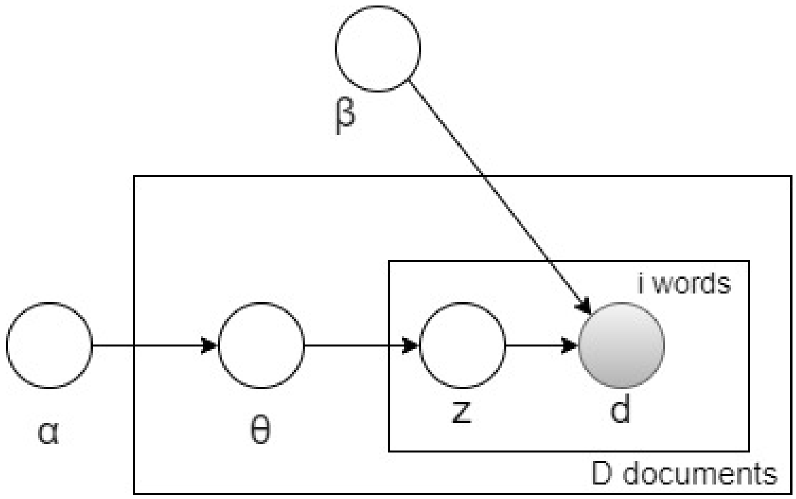


Figure 3: Graphical representation of the LDA model structure.

In Figure 3 the graphical representation of an LDA model is shown. As the figure shows, there are three hierarchies in the LDA model's representation. In the original LDA paper [24] the authors call these three hierarchies corpus-level, document-level and word-level representations. The hyper-parameters α and β are corpus-level, as they are sampled once when generating a text corpus based on the input documents. The latent variables ϕ_d are called document-level since they get sampled once for each document. Lastly, the latent variables $z_{d,i}$ and $w_{d,i}$ are word-level, as they are sampled only one time for each word in every document.

In the training process of an LDA model the multinomial parameters ϕ_d for each document d, and $\phi_{z_{d,i}}$ for each topic are inferred. This is achieved using the *Collapsed Gibbs Sampling* algorithm [25]. The core concept of applying *Gibbs Sampling* to train the LDA model is to sequentially re-sample the posterior probability of a topic j being assigned to a word i of document d from its conditional distribution and letting all the remaining variables stay fixed.

Griffith and Steyvers [25] described this training process formally as Equation: 1

$$P(z_i = j|z_{-i}, \mathbf{w}) \propto \frac{n_{-i,j}^{(w_i)} + \beta}{n_{-i,j}^{(\cdot)} + W\beta} \frac{n_{-i,j}^{(d_i)} + \alpha}{n_{-i,\cdot}^{(d_i)} + K\alpha} \tag{1}$$

where $n_{-i}^{(\cdot)}$ is frequency count without counting the current assignment of z_i, z_{-i} is z without the i-th topic. Frequency counts in this context are the number of times a specific word was assigned to topic j. K is the number of topics, and W is the number the unique words in the text corpus. The conditional probability of topic z_i being equal to topic j given all other topics z_{-i} and all the words $\mathbf{w}$ is the full conditional distribution that gets sequentially re-sampled during the training process. The first fraction represents the proportion of assignments to topic j over all documents that come from word i, while the second fraction represents the proportion of words in a document d that are currently assigned to topic j. In conclusion, the entire training process can be summarized by the following pseudo-algorithm as shown below in Algorithm 1.

Note that by the reassignment in line 13 of Algorithm 1 it is assumed that all topic assignments except for the current word i are correct, and updating the assignment of the current word using our model of how documents are generated. After a large number of Gibbs sampling iterations, a roughly constant state will be reached (LDA model converges), and the topic assignments can be considered stable. These topic assignments can be used to estimate the topic mixtures extracted from each document and the topic words associated with each topic.

2.1.4.3 Common Misuses and Pitfalls of LDA

Agrawal et al. [26] examined the most common and important mis-practises of LDA in software engineering and pointed out a way of correctly using LDA models. The first major mistake that the authors emphasize is the use of unstable LDA models. It is

Algorithm 1 LDA Training Process using Gibbs Sampling.

```
Require: $D$ documents, $K$ number topics, $IterNum$ - Maximum number of Gibbs
    Sampling iterations
for each document $d$ do
    for each word $i$ in document $d$ do
        Randomly assign word $i$ to one of the $K$ topics.
    end for
end for

# Perform $IterNum$ iterations of Gibbs sampling
for $index$ in 1, ..., $IterNum$ do
    for each document $d$ do
        for each word $i$ in document $d$ do
            for each topic $t$ in 1, ..., K topics do
                Compute full conditional probability $P$ from Equation 1
                Reassign word $i$ to topic $t$ with probability $P$
                # In the LDA model $P$ is the probability that topic $t$ generated word $i$
            end for
        end for
    end for
end for
```

crucial to only make conclusions based on a stable LDA. Model stability in this context means that LDA needs to be trained for enough Gibbs Sampling iterations that the model converges. The authors suggest that to achieve a stable model LDA needs to be tuned. Parameter tuning in this context refers to carefully choosing LDA's parameters based on an evaluation metric of the modeler's choice. Topics extracted from an untuned LDA model can produce unstable model outputs. Agrawal et al. [26] work's findings state that using arbitrary chosen or default parameters for LDA models trained on software engineering data will not necessarily result in stable models and can lead to systematic errors. This finding is corroborated by Treude and Wagner [17], who stated that general rules of thumb for LDA parameter configuration do not apply to textual corpora coming from software engineering domain. In their findings, Agrawal et al. [26] also mentions that it is not recommended to reuse an already tuned LDA model's parameters, as LDA parameter tuning is data set dependent.

When performing parameter tuning, it is suggested to always re-tune the model, when training on new data.

Campbell et al. [16] writes about LDA's common pitfalls too. Topics coming out of an LDA model are independent topics generated from word distributions. The authors warn that due to that independence of topics correlated concepts or LDA topics will not necessarily be translated into human ideas or even concepts. The authors also note that comparing topics by associating document contents to topics could be difficult assuming the independence between topics.

Boyd-Graber et al. [19] are more concerned whether the topics extracted from a topic model are interpretable, coherent, meaningful, and useful enough to a human. The authors advised evaluating the quality of topics extracted from the following aspects: topics containing general or specific words, mixed and chained topics, identical topics, stop-words in a topic, and nonsensical topics.

2.1.4.4 Model Selection for LDA models

According to Wallach et al. [27] choosing k, the number of topics present in the corpus is one of the most difficult modeling decisions in topic modeling, as there are no clear methods to follow. Deciding on the right value for k is very important, as according to Agrawal et al. [26] this parameter matters the most for classification tasks. Choosing the number of topics k could be compared to the difficult task of picking how many clusters to consider when performing cluster analysis: there are many heuristics, rules of thumb, but no consensus on which one to use. For instance, Bangash et al. [28] tried two different number of topics ($k = 20$ and $k = 50$) for their experiments, then picked whichever model Mallet's optimizer[6] returned topics that seemed close to actual topics. Panichella et al. [29] designed *LDA-GA*, a genetic algorithm capable of selecting parameters for LDA. Agrawal et al. [26] proposed a search-based software

[6] Mallet Optimizer explained: `https://dragonfly.hypotheses.org/1051`

Table 1: List of papers using LDA and their choice of parameters or parameter ranges.

Paper	k	α	β
Agrawal et al. [26]	[10, 100]	[0, 1]	[0, 1]
Treude and Wagner [17]	[3, 1000]	[0.001, 200]	[0.001, 200]
Bangash et al. [28]	{20, 50}	auto-tuned	auto-tuned
Campbell et al. [16]	{20}	0.01	0.01
Griffiths and Steyvers [25]	{50, 100, 200, 300, 400, 500, 600, 1000}	$50/k$	0.1
Hindle et al. [30]	{5, 10, 20, 40, 80, 250}	MALLET default	MALLET default
Chang et al. [35]	{50, 100, 150}	1	learnt from data
Tian et al. [15]	{100}	0.5	0.1
Arwan et al. [36]	{20}	MALLET default	MALLET default

engineering tool to better tune the LDA model's parameters. Treude and Wagner [17] defined LDA's parameters as ranges of values, then ran hyper-parameter optimization against this search space using perplexity as the evaluation metric. Hindle et al. [30] chose the model with k number of topics where the topics extracted were distinct enough. Furthermore, on top of these general rules, there are many heuristics for deciding k ([31], [32], [33], [25], [34]).

Agrawal et al. [26] states that the choice of good α and β hyper-parameters has the most amount of influence when using LDA in a clustering task. Most LDA implementations recommend and use by default a fixed normalized asymmetric prior of $1.0/k$ for the α parameter, and a fixed prior of 0.01 or 0.1 for the β parameter. Campbell et al. [16] mentions that although 0.01 is a common choice for both α and β, these hyper-parameters can be selected by the modeler based on the desired density of documents and topics. Since it was shown that the use of default parameters can lead to model instability issues, hyper-parameter tuning is strongly advised to achieve model stability [26]. More recent work by Bangash et al. [28], also Treude and Wagner [17] agrees with this claim, as they perform hyper-parameter optimization for α and β based on an arbitrary searching algorithm. Table 1 summarizes the parameter selection practices of recent research work that used the LDA model with textual data from the software engineering domain.

2.1.4.5 Evaluation of LDA models

Boyd-Graber et al. [19] write in their book chapter titled "*Care and Feeding of Topic Models: Problems, Diagnostics, and Improvements*" that perplexity measurements and log-likelihood of a held-out test data are the most common evaluation methods of topic models, but they have limitations. Other forms of topic modeling evaluation could be measuring the marginal probability of a document given a topic model, but Boyd-Graber et al. [19] also say that this is not computationally feasible, since the number of possible topic assignments for words is exponential. The authors continue by stating that based on recent research perplexity does not measure the semantic coherence of topics, and good perplexity scores do not necessarily translate into good performance at some prediction based task. The same can be said for likelihood-based metrics on a held-out data set, as good predictions on the held-out set do not guarantee topics with contextually accurate, semantic representation of concepts. Chang et al. [35] back up these claims by showing that frequently predictive likelihood or perplexity based evaluation methods are not correlated, and sometimes even anti-correlated with human judgement. Furthermore, Wallach et al. [27] found that perplexity and likelihood-based evaluation in most cases is less accurate than performing hyper-parameter optimization against some metric or task accuracy. This method is supported in the literature by Chen and Wang [37], who state that a different evaluation method should measure the LDA model's performance on one or more secondary tasks, such as some form of text classification or clustering. The authors of the original LDA paper support this method too, as Blei et al. [24] called it empirical evaluation of LDA models in various domains.

Lots of previous research was focused on designing automated measurements as close as possible to human judgment. In 2010 this was achieved by Newman et al. [38], who designed a metric based on point-wise mutual information of pairs of topic words and called it *topic coherence score*. In further research from Newman et al. [38], [39], their coherence score was compared with thousands of human evaluations to conclude that their measurements mostly agree with human judgment. A year later, Mimno et al. [40] corroborated that humans agree with word-pair based topic coherence. Topic coherence is further explained in Section 2.1.5.1, where all evaluation metrics used in this project are defined.

2.1.4.6 Validation of LDA models

Due to the various limitations and issues that some evaluation methods have, some researchers opted to manually validate the topics of LDA models by carefully reading through the collection of documents and manually checking whether the topics extracted by the model are coherent enough, and whether or not they align themselves with human judgement.

Bangash et al. [28] performed a manual exploration of topics to validate if the LDA model's suggested topics are useful. They manually read randomly sampled documents and verified that the proper topics are assigned to the sample documents. In this case, the authors also assume that their understanding is the ground truth. The down-side of such manual validation can be that it suffers from high levels of subjectivity.

Bangash et al. [28] state that naming LDA topics is important, as topic models do not come with labels for the topics extracted. This process of topic labeling is needed when doing a qualitative analysis on the list of topic words for each topic. The topic labeling methodology of Bangash et al. [28] and Hindle et al. [30] consists of assigning labels to the topics based on multiple authors' consensus. When evaluating the topics of an LDA model, it is worth noting that according to Hindle et al. [30] some topics might not have a valid and coherent interpretation, and thus, labeling such topics may not be possible. One alternative solution, when faced with this situation, is to label the topic using broad terms, or even discard the topic.

There are various other ways that researchers validate the LDA topics extracted from text documents. Agrawal et al. [26] conducted a study about topic model stability by measuring the median number of overlaps of topic words. The overlap metric used was *Jaccard Similarity*[7], which is often used to measure the similarity of topics. Chang et al. [35] proposed the *word intrusion task* to evaluate the topics extracted from topic models. In this task, a user is given six randomly ordered words. The user's job is to choose the word that does not belong. When the set of words without the word that does not belong makes sense together, the choice should be easy, while in the opposite scenario the authors observed that users tend to pick a word at random, which is a signal that the topic has low coherence. The authors also noted in their findings that there was not a significant correlation between measures of held-out set's likelihood and the measures coming from their word intrusion task.

2.1.4.7 The Use of LDA in the Software Engineering Community

In their book chapter on "*Extracting Topics from Software Engineering Data*", Campbell et al. [16] stated that LDA models emerged as a popular technique within the software engineering community. One reason for their popularity is that LDA models are useful for feature reduction tasks in software engineering. Another reason is that LDA models can be used to create additional features from the input documents,

[7] More about Jaccard Similarity in section 2.1.5.2

which then can be included in the pre-processing steps of other machine learning models.

Campbell et al. [16] claimed that the most important use of LDA in the software engineering domain is linking software artifacts. The authors mentioned that Baldi et al. [41] extracted LDA models' topics, then labelled and compared them to aspects in software development, concluding that some topics were mapping to actual aspects. Campbell et al. [16] wrote that other uses of LDA models in software engineering include performing cluster analysis on issue reports and summarizing the contents of large data sets. Applying LDA to text summarization tasks requires manual or automated labelling of most topics generated by the LDA model. Hindle et al. [30] noted that other uses of LDA in the software engineering field are in "traceability tools, project dashboards, and knowledge management systems".

2.1.5 Evaluation Metrics Used

In this subsection, we introduce four evaluation metrics that we use for model selection and evaluation.

2.1.5.1 Topic Coherence

Human judgement and interpretability of topics are attempted to be quantified by *Topic Coherence* metrics. Boyd-Graber et al. [19] refers to *Topic Coherence* metrics as measures of association between word pairs from a set of top n topic words of a topic. The main concept is that a topic might be evaluated as coherent if word pairs from that topic are associated with each other. Röder et al. [42] worked on a comprehensive study on topic coherence measures to create state-of-the-art hybrid coherence metrics by combining different components of already existing coherence metrics. According to Röder et al., there are two types of aspects behind recent research on topic coherence: scientific philosophy and topic modeling. Röder et al.

proposed a coherence framework that consists of four steps: 1) segmentation of word pairs, 2) estimation of word probabilities, 3) computation of “confirmation measures”, which tests how strong the coherence between any two word pairs is, then 4) aggregating “confirmation measures” to form a single coherence score. Based on Röder et al.’s research four promising topic coherence metrics emerge as the metrics that are most correlated with human judgements and interpretability:

- UCI is a metric based on Point-wise Mutual Information (PMI), and it estimates word probabilities based on word co-occurrences. UCI is also known for having the largest correlation with human annotations.
- $UMASS$ is an asymmetric “confirmation” metric between top word pairs, and it takes into consideration the ordering of topic words in a topic.
- $NPMI$ is a metric based on normalized Point-wise Mutual Information (PMI), and it works by generating context vectors for each topic word in a topic.
- C_V is the best performing coherence metric, being a hybrid metric between cosine similarity measures, the $NPMI$ metric and a boolean sliding window approach.

These four coherence metrics were used to guide model selection in Section 3.5.

2.1.5.2 Jaccard Similarity and Distance

Jaccard similarity measures the similarity and diversity of sets, and is it defined as “the size of the intersection divided by the size of the union of the sets” [43]. *Jaccard similarity* is used in Section 3.7.2 in the task of *Cross-platform Expertise*, and also in Section 3.7.1.4 for the task of *Expertise Prediction*, to measure the similarity between a model’s set of prediction words and a human annotation’s set of words.

Jaccard distance being the conjugate of *Jaccard similarity*, it is defined as the measure for the dissimilarity between sets, and it can be calculated by "subtracting *Jaccard similarity* from 1" [43]. This metric is used in Section 3.7.4 for the task of *Expertise Evolution*.

2.1.5.3 BLEU Score

BLEU score (Bilingual Evaluation Understudy score) is a metric for evaluating a generated to a reference sentence, translation or summary. *BLEU scores* are often used in machine translation tasks. The inventors of *BLEU score* explain it as a comparison of "n-grams of the candidate with the n-grams of the reference translation [or summary] and count the number of matches. These matches are position-independent." [44]. This metric is used in Section 3.7.1.4 for the task of *Expertise Prediction*, to evaluate how similar is a model generated, candidate expertise summary to a reference summary coming from a human's annotation.

2.1.5.4 Cosine Similarity

Cosine similarity measures "the cosine of the angle between two non-zero vectors of an inner product space" [45]. *Cosine similarity* is often used in data mining and information retrieval tasks as a means of comparing high dimensional feature vectors. Two feature vectors aligned in the same orientation have a cosine similarity score of 1, meanwhile, two feature vectors aligned perpendicularly produce a similarity score of 0. Lastly, two feature vectors oriented in exactly opposite directions produce a similarity score of -1. In most data mining and information retrieval applications, *Cosine similarity* is used only in the positive space, being bounded between 0 and 1. *Cosine similarity* is used in Section 3.7.1.4 for the task of *Expertise Prediction* to compute a multi-word comparison between a model's prediction words and a human annotation's set of words.

2.2 Related Work

In this section, we describe the related work which we split into two main areas. *Mining Stack Overflow and GitHub* is presented in Section 2.2.1, while Section 2.2.2 is dedicated to the previous work on *developer expertise learning and recommendation.*

2.2.1 Mining Stack Overflow and GitHub

Vasilescu et al. [46] was one of the first researchers to explore the interaction between Stack Overflow and GitHub activities, and the software development process. The authors have linked Stack Overflow and GitHub user accounts to explore development activities on both platforms. This study also looked at commit patterns and question/answer activities of active developers and analyzed on which platform are developers more active. Their findings are significant, as they conclude that active GitHub users ask fewer questions and provide more answers than other users. Furthermore, the study shows that active Stack Overflow users split their work in more non-uniform ways than users who do not ask questions. Lastly, the study also concludes that the rate of activity on Stack Overflow is correlated with the rate of code changes on GitHub.

Building on the work of Vasilescu et al. [46], Badashian et al. [47] investigated influence, involvement, and contribution across the same two platforms, by correlating activities on GitHub and Stack Overflow users. The authors have used the same data set as Vasilescu et al. [46]. The study suggests that the early users of GitHub are also early users of Stack Overflow, which shows a sense of belonging to a community. The findings of the study also indicate that before 2010 users were more active on Stack Overflow but after 2010 GitHub activity increased significantly and users were more active on GitHub than Stack Overflow. Lastly, in terms of the type of activities performed across the two platforms, there are two distinguishable groups: typical users and highly-active users. The most frequent activities of typical users were committing, posting and answering questions, while highly-active users posted more answers, committed more often and barely asked questions.

Lee and Lo [48] took Badashian et al. [47]'s work one step further and analyzed developer interests across the same two platforms, GitHub and Stack Overflow. The authors were looking into whether developers share interests across their GitHub and Stack Overflow accounts, and do developers share interests with other users who participated in similar activities as them. Their findings conclude that there are common interests between GitHub repositories and Stack Overflow posts of a user, and the similarity of interest in on average, 39%. Furthermore, many users share common interests with other users who participated in similar activities as them.

Vasilescu et al. [49] built on their work from the previous year ([46]) and investigated how mailing lists and Stack Exchange provide knowledge sharing in open source software communities. Their research questions focused on investigating the differences between active members of both mailing lists and Stack Exchange and members active on a single platform, also what are there differences between the two online communities (mailing lists and Stack Exchange)? The study's findings show that mailing list participation decreases drastically since 2010. Around the same time, more questions started to be asked about R on Stack Overflow and Cross Validated. Vasilescu et al. concluded that it is very likely that part of the mailing list population has migrated over to Stack Exchange at some point during 2010, where they can answer and get answers to questions much faster than on the mailing list.

Barua et al. [50]'s study is connected to Vasilescu et al. [49]'s work by studying the process of sharing and gaining knowledge through being part of and interacting with like-minded developers in open source software communities. Barua et al. [50]'s goal was to analyze the textual data of Stack Overflow posts with the use of LDA topic models and discover topic trends in developer discussions. Barua et al. collected two years' worth of Stack Overflow data between July 2008 and 2010. Some of the findings of their analysis show that in those two years mobile application development was increasing in popularity at an even faster rate than web development. According to the authors both Android and iPhone development were much more relevant than development for Blackberry's. In terms of programming language popularity, PHP was becoming quite popular between 2008 and 2010, and Java was still considered as one of the main programming languages. Furthermore, other insights coming out of their analysis show that Git overtook SVN in version control popularity, while at that time MySQL was the most popular database.

Considering the popularity of Github and Stack Overflow in the software engineering research community ([46], [47], [48]) Gousios et al. [51] has published the *GHTorrent project*, which is a public mirror data set of all public projects available on GitHub. This new data set contains 16 entities about all aspects related to the GitHub context, including projects, users, commits, followers, watchers, issues and pull requests. The *GHTorrent* data set became a crucial resource for almost all GitHub related research ever since its released, as it enabled researchers to easily retrieve all public projects hosted on GitHub.

Kalliamvakou et al. [52]'s work builds on top of the research hype of mining GitHub's events data, trying to understand how developers use GitHub to collaborate on open source projects. The authors focus on studying the quality and characteristics of repositories on GitHub. The findings of their study concluded that GitHub is indeed a rich data source, there are a handful of insights mined from repositories, such as that over 70% of projects are personal, and only a small percentage of them use pull requests. According to the authors, surprisingly, a large portion of repositories on GitHub are not used for software development purposes, but rather for free source code storage and other reasons. Another interesting finding is that over half of the projects are personal and inactive, while close to half of all pull requests do not show up as merged in, but they actually are.

While others have focused on trends and knowledge sharing on Stack Overflow, Arwan et al. [36] have looked at source code retrieval on Stack Overflow. The authors have collected data from the public Stack Exchange data dumps, then applied LDA topic modeling to retrieve source code relevant to a queried concept. The authors have concluded that LDA is capable of retrieving source code with the best accuracy of 86% precision and 70% recall. It was also found that the proportion of relevant documents to the query concept highly impacts the precision and recall values achieved.

Chowdhury and Hindle [53] show one of the many real-life applications of mining Stack Overflow data by designing a chat filtering algorithm that can hide off-topic discussion in Stack Overflow and other online forum discussions. The authors ended up considering only two machine learning algorithms, Support Vector Machine and Naive Bayes classifiers to discover off-topic discussions with an F-score of 87.9%.

In another application of data mining in software engineering, Yang et al. [54] studied how code snippets are copied into new projects. The authors were curious of code snippets are copied from Stack Overflow and pasted into source files that end up on GitHub, and if this is the case, them how much change is made to the original code snippet. The authors analyzed close to 1 million unique Python projects hosted on GitHub and detected 1.9 million Python code snippets from Stack Overflow. Their analysis shows the rare (less than 1%) existence of exact duplication of code, but rather token-level duplication is the more common type of code similarity. Furthermore, after carefully reviewing the findings of the analysis, the authors noticed that the majority of code duplication is usually 2 lines of code, and in most cases, no comments are provided. Lastly, the authors did find evidence of small ‘migration’ of code from Stack Overflow to GitHub, and in serious cases, this is being mentioned in the form of comments.

After seeing the popularity of mining Stack Overflow and GitHub data together

([46], [48], [47], [54]), Baltes et al. [55] followed the footsteps of Gousios et al. [51] and created the *SOTorrent* open data set based on the official Stack Overflow data dump. *SOTorrent* stored version histories of Stack Overflow data on two separate levels: entire posts and individual code and text blocks. What contributed to the fast-growing popularity of *SOTorrent* is the ability to link Stack Overflow posts to GitHub repositories by detecting hyperlink references from GitHub files to Stack Overflow posts. After releasing *SOTorrent* data set to the public the authors have performed the first analysis on the entirety of Stack Overflow data. Their analysis shows that out of all posts on Stack Overflow, 36.1% of them have been edited after their creation. Furthermore, Baltes et al. argued that all edited posts are very rich in comments, having a large number of comments compared to posts with no edits. Baltes et al. inferred that these comments lead to the edit of the post. Finally, the authors' vision was that *SOTorrent* will be used to explore and understand the evolution of Stack Overflow posts and their relation to other online software communities.

In 2019, the above mentioned "vision" turned into reality, when the *SOTorrent* data set became the annual mining challenge [8] at the Mining Software Repositories (MSR) conference. Baltes et al. [56] encouraged researchers in the MSR community to explore the maintenance of code snippets on Stack Overflow, or design ways to better detect clones of code snippets. Other suggested topics to discover were detecting source code containing bugs, predicting bug-fixing edits, understanding the evolution and migration of Stack Overflow code snippets into GitHub repositories, or predicting the popularity of code snippets. This mining challenge designed by Baltes et al. [56] opened up new, innovative research avenues and made even more popular the mining of software artifacts, such as Stack Overflow in the software engineering community.

Manes and Baysal [57] used both SOTorrent [56] and GHTorrent [51] data sets in their work to investigate how often GitHub developers copy code snippets from Stack

[8] `https://2019.msrconf.org/track/msr-2019-Mining-Challenge`

Overflow into their projects, also what concepts are referenced more often in their code. The authors have mined the connectivity between code snippets on Stack Overflow posts and software projects on GitHub. Manes and Baysal [57] discovered that GitHub developers reference programming language related discussions that match the language of their code, and secondly they found that 79% of posts with code snippets evolve.

Treude and Wagner [17] used the *SOTorrent* data set from Baltes et al. [56]'s work and studied the characteristics of GitHub and Stack Overflow text corpora to predict good configurations for LDA models built on such corpora. Their work was purely data-driven by sampling 40 text corpora from GitHub and another 40 from Stack Overflow to discover the impact of software artifact related corpus characteristics on parameter selection of topic models. Their findings are significant, as they conclude that general guidelines for parameter selection of LDA models do not apply to GitHub and Stack Overflow corpora, as such data contains different characteristics to ordinary textual data. Lastly, Treude and Wagner conclude that to achieve a good model fit, LDA parameters can be predicted even for unseen data by studying the characteristics of corpora.

Treude and Wagner [17]'s work on LDA parameter selection is highly relevant when it is linked with the proper application, such as Bangash et al. [28]'s work on applying LDA models to Stack Overflow posts about machine learning. Their study focuses on questions such as what machine learning topics are discussed in posts, and what characteristics do machine learning posts have on Stack Overflow. The results of the study are surprising as even though machine learning is a really popular field of computer science, Bangash et al. suggest that many developers do not have an adequate introductory knowledge of machine learning, and the feedback from online communities in not helping them close the gaps in their knowledge. Lastly, through topic modeling, it was discovered that the most frequent machine learning topics discussed on Stack Overflow are algorithms, classification, and training data-set related.

Chatterjee et al. [58] performed a similar study to Bangash et al. [28] by exploring the content of Slack chats as a potential source of software artifact related information to be mined, such as Stack Overflow. Their study focuses on the characteristic similarities of Slack chats and Stack Overflow posts, also could automatic mining of chat data be useful towards the improvement of software engineering tools. Throughout the study, it was observed that most of the information mined from Stack Overflow posts are also available on Slack channels, but there is an exception: API mentions, which is more available on Stack Overflow posts rather than Slack chats. Other findings of their study include the insight that the majority of Slack conversations are about software design and in some cases there are a large number of accepted answers from Stack Overflow posts present in Slack chats.

Prana et al. [59] worked on better understanding and categorizing content in GitHub repository README files. The authors have manually annotated over 4,000 README file sections from almost 400 randomly sampled GitHub projects to create a classifier that can distinguish a README file's sections automatically. The main goal of this research work was to allow repository owners to provide better documentation and to make browsing through the information on description files easier for other GitHub users. The findings of the study show that many description files do not have information about the purpose and status of the project. The classification model built during this study achieved an accuracy of almost 75%, and this classifier was also used to label sections in unseen README files from other GitHub repositories.

Other researchers, like Liao et al. [14] chose to take advantage of the large variety of diverse data being available on GitHub and use discussion data to study the "sociolinguistics perspectives" of tagging in open source projects. The study's emphasis was on the usage of language in projects. The authors were wondering whether it is possible to detect very productive or relevant developers just by the language that they use in issue discussions. The authors used multiple language models to convert textual data into vector representation of words, as they found that learning the semantic context of discussions is important. Their study found the tagged developers in discussions could be predicted with a 'reasonable' level of accuracy.

2.2.2 Developer Expertise Learning and Recommendation

Tian et al. [15] formulated the task of finding expert developers in open source software communities as building a model to predict the best answerer for a new question posted on Stack Overflow. The authors proposed the idea of modeling a Stack Overflow user's topical expertise and interest using LDA models. Tian et al. made some assumptions along the way: firstly, the user considered to be the best answerer for a question should have high interest and expertise on the specific question being answered. Secondly, Stack Overflow questions belong to or come from a mixture of topics (to agree with LDA's most important assumption) and lastly, user interest and expertise in Stack Overflow questions are affected only by user's interest and expertise on the related topics. Based on these assumptions the authors build a user profile consisting of previous questions answered or asked, for each user. The authors measure the accuracy of their approach using the *Success at N* evaluation metric, where N varies. In the end, the topic modeling-based approach is compared with a frequency-based (TF-IDF) baseline approach. The authors concluded that their proposed approach outperforms the baseline approach, but there is still room to improve the effectiveness of the technique.

Badashian et al. [60] worked on the task of bug report assignment, which consisted of assigning a software bug report to the developer who is most fit to address the bug. In this task, a list of developers with relevant skills and knowledge need to be compiled, then ranking the developers based on their expertise is needed. For this task, Badashian et al. created a novel approach, determining a developer's expertise based on their contributions to Stack Overflow. By the use of this approach, the authors concluded that Stack Overflow is a rich source of developer expertise related data. In the evaluation process, the authors report that the proposed novel approach reached state-of-the-art accuracy.

Montandon et al. [61] surveyed developers about their expertise in three software libraries and frameworks, then merged the self-reported expertise areas with features extracted from each developer's GitHub account. Their study focused on studying what features distinguish the expertise of developers from the collected data, then how accurately could a machine learning algorithm classify different expert classes found in the data. In the results of the study, the authors suggested that standard machine learning algorithms could not accurately predict expertise based on GitHub data only, as not all survey subjects had a strong public activity on GitHub. Better results were achieved using cluster analysis, thus the authors have proposed the detection of experts based on how similar the expertise of a new developer is to the already pre-determined cluster of experts from the surveyed data.

Similar to Montandon et al. [61]'s work, Sindhgatta [62] applied information retrieval and document clustering techniques to detect the domain expertise of developers from source code. The proposed technique generated documents from text documents from source code by filtering out all the programming language related keywords, then clustering is used to group documents and extract relevant concepts. The author has performed multiple case studies to evaluate the above technique and conclude that their technique can extract the relevant concepts from source code.

Baltes and Diehl [4] are credited with proposing the first comprehensive theory of software development expertise based on a data-driven mixed-methods survey. The authors have described their work as a grounded theory of expertise, which is a qualitative and quantitative combination of knowledge and experience. They argue that among other things, skills and work context influence the creation of expertise, which can be detected through "well-structured, readable, and maintainable source code". Furthermore, throughout the process of grounded theory, the authors augment their conceptual theory by introducing a "task-specific view on expertise". This represents the addition of the concept of "deliberate practice", which includes the process of practice, asking for feedback and performing self-reflection on the work. The authors introduced this crucial aspect of their conceptual theory to focus on performance as a consequence of gaining more expertise.

In a very recent study, Li et al. [63] surveyed almost two thousand senior software engineers asking them what attributes make a great software engineer. The authors compiled all survey responses and found that the five highest-ranked attributes to be a great software engineer are: 1) producing good source code, 2) maximizing the value of work, 3) practicing knowledgeable decision-making, 4) not making someone else's jobs more difficult, and 5) continuous learning. The lowest-ranked attribute was favour trading.

Wang et al. [9] have created a survey paper type overview of the latest research techniques proposed for the problem of expert recommendation in question answering communities, such as Stack Overflow. The authors categorized the currently existing methods into 8 categories: simple, language model-based, topic model-based, network-based, classification-based, probabilistic, collaborative filtering- and hybrid methods. Data sets vary between *Yahoo! Answers, Stack Overflow, Tripadvisor, Java Developer Forum*, other *Stack Exchange* sites and many others. Evaluation metrics between the above listed categories of methods do not agree, as Wang et al. found over 10 different evaluation metrics used for the task of expertise recommendation. Some of these evaluation metrics include precision, recall, F1-score, Precision at top n, matching set count, Pearson correlation coefficient, area under ROC curve, accuracy by rank and many others. For future work, Wang et al. recommended using more external data to develop more realistic user models, emphasized the importance of model robustness and opened up a new research avenue focusing on the recommendation of experts as a collaborative group.

Tian et al. [8] have addressed the problem of user recommendation in GitHub and Stack Overflow. The authors have proposed a novel methodology to extract talented developers from both GitHub and Stack Overflow and built a complete system that automatically ranks developers in various areas of computer science. Tian et al. have also created a user interface to visualize the user recommendations, ranked based on their skills.

Teyton et al. [5] proposed an approach to finding relevant experts of common libraries among GitHub developers. The authors' definition of a library expert is a developer who committed changes to the underlining source code of the library. Teyton et al. [5]'s work involved designing a 'search engine' for library experts by creating a novel query language to detect expert developers. While such a tool can be useful to open source software communities, the authors note that their proposed system is underused, and lacks appropriate validation.

Liao et al. [64]'s work is similar to Teyton et al. [5] and Tian et al. [8], as they suggested ranking influential developers on GitHub. The authors define the influence of a developer as the number of followers it gains over some period of time. The authors design a novel network-based algorithm named '*DevRank*', which ranks developers based on influence and commit behaviour. Liao et al. compared their approach to link analysis and traditional network-based algorithms like *PageRank* and *HITS*, and achieved better precision scores.

Hammad et al. [65]'s work is similar to Teyton et al. [5] by exploring how to identify the appropriate software designer for the task of changing the inner architecture of the software. The authors also propose a methodology to detect the type of design knowledge a developer possesses and the type of design knowledge needed for a task.

The system proposed by Hammad et al. automatically recommends a list of candidate software designers suited to work a design change task, and creates a ranking of the candidates based on their knowledge. Lastly, the authors conclude that their system is useful for handling requests in large software projects consisting of many developers.

Similar to Hammad et al. [65]'s work, Kagdi et al. [66] proposed an approach to suggest a ranked list of experts to implement software changes. The authors credit themselves with the first use of a “concept location” technique combined with software repository mining techniques to recommend expert developers. Their approach was evaluated using change requests from three open-source software. The authors report accuracy values between 47% and 96% for correctly recommending developers to fix specific bugs, while for recommending developers to implement specific features the accuracy values are between 43% and 60%.

Alonso et al. [67] conducted a study that used the rule-based classification of source code trees as a way to detect expertise of committers and other contributors to open source projects. The authors explain that only a small number of people are developers in a project (they have commit privileges), but there are plenty of other people who can contribute to the source code, via bug fixes and patches. One of the author's secondary goals (besides detecting expertise) is to be able to distinguish the developers from contributors. Alonso et al. also developed a visualization of the results of the proposed technique, and concluded that it is possible to automatically detect and extract the expertise of developers from version control log files and source code trees.

Surian et al. [68] built a graph-based model to represent developers collaborating on projects. Their goal was to recommend a set of developers that are good matches to work on an input project based on their previous experiences and skills that they have. In Surian et al. [68]'s work an edge connected to a developer and a project if the developer worked on that project. Another edge connected a project and a property if the project has that specific property. The authors have computed similarity measures based on random walks of the graph with restarts and extracted the top n recommended developers. This approach resulted in an 83.33% accuracy on a ground truth data set.

Similar to Surian et al. [68], Hauff and Gousios [7] worked on matching GitHub developer profiles to job advertisements by automatically suggesting job opportunities to developers based on their GitHub activities. Their approach involved concept extraction from job descriptions and GitHub profiles, introducing a weighting system for the concepts extracted to create a hierarchy, then matching job descriptions to developers. The job - developer matches are obtained by computing the similarity between weighted concept vectors, which are essentially feature vectors. Hauff and Gousios have created a great tool, but perhaps it could be improved with the addition of a formal validation and filtering stage for the candidate matches.

Nguyen et al. [69] have conducted a study to infer the expertise of developers based on the characteristics of their work when assigned to fix defects in software. The authors categorize each corrected defect by topic and rank developers by inferring their expertise in the defect's each topic. Nguyen et al. concluded that the amount of time required to fix a defect is determined by the expertise of the developer in the defect's topic. To validate their work the authors have interviewed many project managers and asked them to compare their ranking of developer expertise with the results of the study. The authors have concluded that automated developer expertise ranking could be valuable for software development.

Zhang et al. [10] explored how to recommend compatible projects to developers based on their user activity and behaviour data on GitHub. The authors focus on specific research questions, such as what type of user behaviour data is needed in recommend suitable projects, and is there a relationship between recommended projects found using user behaviour? The findings show that *'fork', 'watch', 'pull-request' and* *'member'* are the user activities and behaviours relevant for recommending projects to developers. Furthermore, Zhang et al. discovered that the majority of recommended projects are either dependent on one another, or they are used together, or they are just similar projects.

Yu et al. [70] investigated whether previous approaches used in code review assignment can be adapted and improved to recommend reviewers for pull requests. The authors have tried three methods based on traditional machine learning adapted from bug and code-review assignment, then proposed a novel method for pull requests recommendation by mining social relationships between the submitter of the pull request and potential reviewers. The results show that traditional machine learning adapted methods outperform the baseline, while the authors' novel approach achieves similar F-scores as the traditional machine learning approaches. Finally, Yu et al. also experimented with a hybrid approach, which resulted in a significant increase in precision and recall, and the overall accuracy is more balanced than using one or the other approaches independently.

Greene and Fischer [6] have created a tool called *'CVExplorer'*, which extracts, explores and visualizes technical skills of developers from their GitHub account. Their approach uses mines and aggregates the technical skills of developers from multiple data sources and generates an 'interactive tag cloud' to allow potential recruiters to further exploration. The authors claimed that they evaluated *'CVExplorer'* by recommending candidates for open positions at two companies, and showed that such a tool could be very useful to non-technical recruiters.

Similar to Yu et al. [70]'s work, Rahman et al. [71] have proposed a novel code reviewer recommendation technique that takes into account the cross-project experience of a developer in specific technologies on top of looking into their previous projects on GitHub. Their approach goes even further and attempts to detect the developer's expertise as a candidate code reviewer by making associations between the developer's previous pull requests and their experience and skill levels in specific technologies. The authors have done extensive experiments using ten commercial projects and ten external libraries dependent on commercial projects. The results of their experiments show that their technique's accuracy is between 85% and 92% in code reviewer recommendation, which beat the state-of-the-art technique at that time.

Looking at previous research on cross-platform expert recommendation, Constantinou and Kapitsaki [72] proposed an approach for the extraction of developer expertise in different programming languages. The authors have developed a method that tracks the commit activity of developers on GitHub, taking into account the quantity and consistency of their contributions. The authors validated their method against the same developer's user activity on Stack Overflow via a linked data set of active users on GitHub and Stack Overflow from Vasilescu et al. [46]'s work. The authors concluded their work by stating that their proposed approach can identify developers with similar expertise and recommend developers for specific tasks.

Venkataramani et al. [73] suggested an approach to rank the expertise of developers by mining their activity in open source projects. The authors have built a recommendation system for two tasks: developer recommendation and question recommendation for Stack Overflow using data mined from GitHub. This type of cross-platform system is very similar to Constantinou and Kapitsaki [72]'s work.

Silvestri et al. [74] linked together user accounts across Stack Overflow, GitHub and Twitter to run a comparative study of user interaction on the above three platforms. The authors' objective was to investigate correlations between user interactions across the three different platforms. Their study revealed a couple of insights: firstly, users on Twitter are more connected than users on Stack Overflow and GitHub. Secondly, GitHub and Twitter have a more skewed distribution of user popularity, and more correlated user activities between them, than Stack Overflow.

Huang et al. [75] worked on a novel approach for modeling and scoring the programming skills of developers through mining data from Stack Overflow and GitHub. The authors have analyzed a developer's Stack Overflow posts and evaluated their GitHub projects to assign expertise scores for each programming language that a developer is active in. After conducting extensive experiments Huang et al. reported that their approach is practical and achieved a maximum precision score of 80% on Stack Overflow and GitHub data.

Zhang et al. [76] proposed a recommendation system that combines users' activities on Stack Overflow and GitHub to recommend developer candidates to open source projects. The author's experiments show that their system is good at proposing candidates for the cold start problem, while some of their hybrid approaches perform well at recommending proper developers for open source projects.

There have been numerous previous research efforts addressing the detection of developer expertise, especially using data mining of social or collaborative platforms. However, the similarity between these approaches is that most of them have major flaws or limitations: they either address only very specific expertise areas (e.g., Montandon et al. [61] only looked at expertise in three JavaScript libraries), or consider only one type of data (e.g., commit activity of developers on GitHub in Constantinou and Kapitsaki [72]'s work; source code in Sindhgatta [62]'s work), or used generalized expertise formulas (e.g., Tian et al. [8] calculated expertise scores using self-created expertise formulas). We have considered the theory behind developer expertise [4], numerous suggestions ([9], [15], [5], [68], [6], [64], [60]) and limitations of other papers ([61], [72], [62], [8]) when developing and proposing three topic modeling based robust, purely data-driven and novel techniques for cross-platform expertise learning. Our proposed approach's philosophy matches Tian et al. [15]'s by agreeing that user activity is a rich source of data about the software development process, and by building user activity profiles one can discover the hidden patterns of a user's expertise.

Furthermore, we both recognized the LDA model's ability to model topics in user activity profiles.

Chapter 3

Methodology

This chapter describes the methodology we followed to study cross-platform developer expertise. Section 3.1 describes two key data sources such as Stack Overflow and GitHub and offers details on the data collection process. Section 3.2 reports how data was stored throughout the entirety of the project. In section 3.3, we explain the data wrangling and aggregation process, while in section 3.4, we present all details behind the data cleaning process. Section 3.5 represents the start of the modeling part of our methodology, as training of topic models is described. In section 3.6, we present an expertise study and how we collect human annotations for developer expertise. Section 3.7, being the largest section of methodology, describes what algorithms are designed to extract expertise insights from data. Last, section 3.8 contains a detailed overview of all the software tools used.

3.1 Data Collection

When mining GitHub[1] and Stack Overflow[2], a natural starting point is to use GHTorrent [51] and SOTorrent [55] as data sources.

[1] https://github.com/

[2] https://stackoverflow.com/

3.1.1 Stack Overflow Data

SOTorrent[3] is an open data set based on periodic, official Stack Overflow data dumps. SOTorrent was built for mining and analyzing the evolution of Stack Overflow posts. A new version of this data set based on the latest Stack Overflow data dump is released every 3 months. Each version of the SOTorrent data set can be downloaded from Zenodo[4]. The latest versions of the SOTorrent data set are also available online, and can be queried through Google BigQuery[5].

When downloading the SOTorrent data set, the version from *December 9th 2018* was used, as it was the latest version available at the start of the project. The SOTorrent data set has its files organized into tables. Not all tables were downloaded, as certain tables related to post evolution (e.g., `PostHistory` and `PostVersion` tables) were unrelated to the expertise learning task. The tables downloaded included `Users`, `Posts`, `PostLinks`, `PostType`, `PostReferenceGH`, `Comments`, `CommentUrl`, `GHMatches`, `Tags`, `Badges` and `Votes`, totaling 29 GB of compressed raw data. After downloading and decompressing all relevant data files, the data set contained just shy of 100 GB of data.

3.1.2 GitHub Data

GHTorrent[6] is also an open data set, which mirrors the data offered through the GitHub REST API[7]. GHTorrent collects data from all public projects available on GitHub and releases a new version of MySQL data dumps every month, while it also offers daily data dumps through MongoDB. Each version of MySQL data can be downloaded from GHTorrent[8]. GHTorrent data can also be queried online, as it is

[3] `https://empirical-software.engineering/projects/sotorrent/`
[4] `https://zenodo.org/record/2273117`
[5] `https://bigquery.cloud.google.com/dataset/sotorrent-org:2018_12_09`
[6] `http://ghtorrent.org/`
[7] `https://developer.GitHub.com/`
[8] `http://ghtorrent.org/downloads.html`

available via a DBLite web interface[9].

When downloading the GHTorrent data set, we used the version from *March 1st 2019*, as it was the latest version available at the time of download. The data set has its files organized into large tables stored in compressed CSV files, totalling 96,557 MB of compressed data. All files were downloaded and uncompressed which resulted in 21 raw data tables totalling over 400 GB of data.

3.2 Data Storage

3.2.1 Database Setup and Data Import

A MySQL database was the obvious choice of database, as both data sources come with SQL scripts performing table creations, data imports and general database manipulations[10] [11]. Managing disk space during the data import phase represented a challenge, as the raw, uncompressed data files totalled up to over 500 GB. Importing of the data into the database was not achieved all at once, but rather through several iterations of uncompressing only a few data files. After a file was uncompressed and the table was successfully imported into the database, we deleted its raw CSV or XML file to free up disk space.

During the data import phase, we discovered that using *MyISAM* over *InnoDB* as the database engine is more favourable since there was a significant execution time difference between the two database engines when importing large amounts of data into the MySQL database. *MyISAM* database engine does not support foreign keys constraints, while it supports table-level locking. On the other hand *InnoDB* supports foreign key constraints and row-level locking. *MyISAM* is preferred when performing tasks that require a fast import and querying speed, while *InnoDB* being

[9] `http://ghtorrent.org/dblite/`

[10] `https://GitHub.com/sotorrent/db-scripts/tree/master/sotorrent`

[11] `https://GitHub.com/gousiosg/GitHub-mirror/tree/master/sql`

the default database engine for MySQL is more optimal for regular operations. For this particular reason, we imported all of the large raw data files into the database using the *MyISAM* database engine. After all necessary database manipulations have been performed, the database engine was changed back to the default engine, *InnoDB*, to allow foreign key constraints to be enforced.

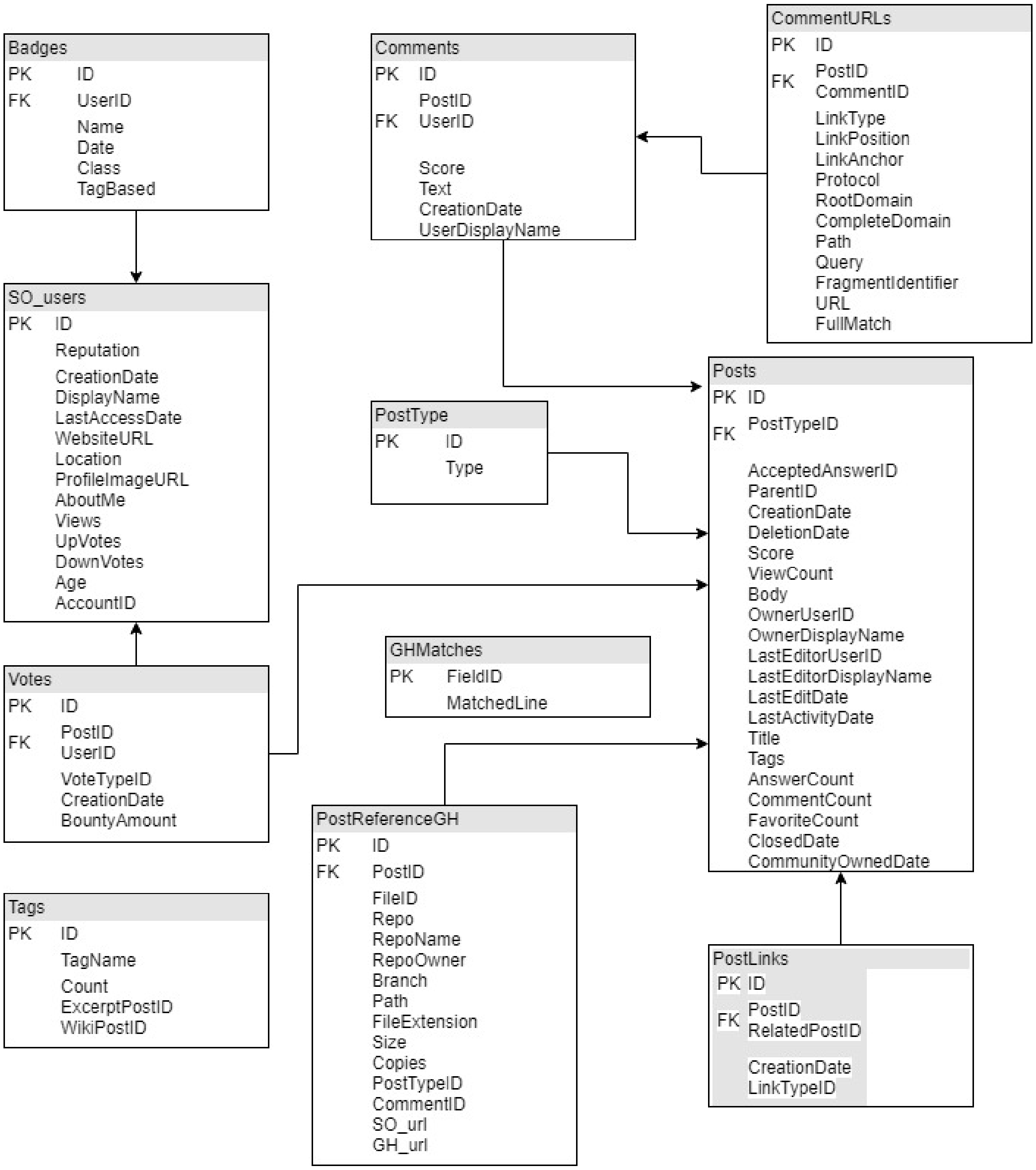


Figure 4: Database schema for Stack Overflow data.

Figure 4 and 5 show the database schema, including table attributes of the SOTorrent and respectively the GHTorrent data set. It is worth noting that the above mentioned database schemas were not designed by us, instead, the figures have been re-created by us based on Baltes et al. [55]'s and Gousios [51]'s work. For the SOTorrent database design in Figure 4 only the tables relevant to our work are shown.

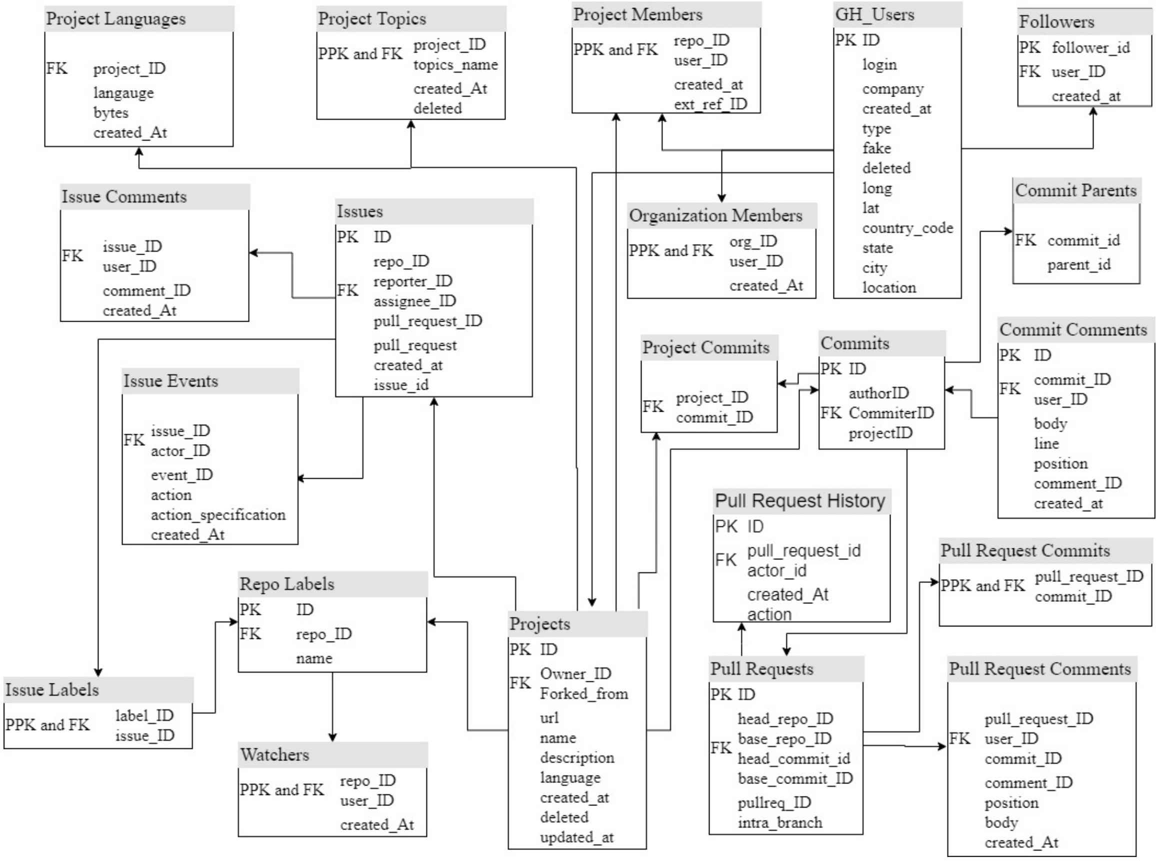


Figure 5: Database schema for GitHub data.

3.2.2 Linking together GHTorrent and SOTorrent

To perform cross-platform analysis of GitHub and Stack Overflow, we need to link the GHTorrent and SOTorrent data sets. This task requires identifying the same user's accounts on both platforms. Looking at relevant literature on this problem, we can identify the task of *identity merging*, which consists of identifying the same person in two or more different environments. Vasilescu et al. [46] researched this problem rigorously and after careful consideration of limiting the number of false positives they used email addresses to determine user identity. In 2013, at the time of publishing their work, email addresses were present in the GitHub data set, while in the Stack Overflow data set email addresses were not publicly released, but their MD5 hashes were available. Vasilescu et al. [46] merged a GitHub and Stack Overflow user if the MD5 hash of the email in GitHub was the same as the MD5 hash of Stack Overflow user's email. This resulted in 93,771 GitHub users being linked to Stack Overflow. Vasilescu et al. [46] further investigated and concluded that out of 93,771 linked users only 46,967 users were active at the time. The linked data set alongside a replication package has been made publicly available by the authors[12]. This data set has been used multiples times to analyze the interaction of GitHub and Stack Overflow data [47] and [48], thus it became the backbone of the data sampling process behind linking together GitHub and Stack Overflow users.

There was one major problem with using Vasilescu et al. [46]'s data set. The replication package contained a data set linking GitHub email addresses to Stack Overflow user IDs, without fully mapping GitHub user IDs to Stack Overflow user IDs, therefore, we needed to map GitHub email addresses to user IDs. This mapping was possible when Vasilescu et al. [46] published their work, but after March 2016 GHTorrent was not allowed to store email addresses in their open data sets, as it violated the EU General Data Protection Regulation (GDPR). Our solution to this problem was to download the `User` table data from GHTorrent's last available version that still contains email addresses of GitHub users in the data set. This older version of GHTorrent is dated *February 1st 2016*, and it was downloaded, then imported into the MySQL database. We performed a manual check on matching user IDs and login names between the `User` table's March 2019 and February 2016 versions.

[12] `https://www.win.tue.nl/mdse/stackoverflow/`

This manual check made sure that the user IDs from both versions of the table are referring to the same login name. If this was not the case, the user was dropped from the data set, due to the inconsistency of the two versions of the table. This manual check assured consistency in the data linkage between Stack Overflow and GitHub users, but resulted in the removal of over 10,000 users from the original 93,771 users linked. The final data set contains 83,550 user accounts being linked between the two platforms, offering a unique GitHub user ID to Stack Overflow user ID mapping. It can be speculated that removed users either deleted or changed the login name of their GitHub account between 2016 and 2019, thus causing the inconsistency in the two versions of the `User` table.

3.2.3 Discarding Unlinked Data

As mentioned in Section 3.1, the total size of the GHTorrent data set was over 400 GB of data, while SOTorrent contained close to 100 GB of data. To make data querying and processing more efficient, we needed to perform a data reduction on the full data set. We filtered out unlinked data by iterating through each existing table in the database and keeping only the observations that are connected or related to users with a user ID present in the list of 83,550 unique user IDs linked. Discarding unlinked data reduced the size of the data set and allowed only linked data to be analyzed. This process reduced the size of the MySQL database from over 500 GB of data to 122 GB.

3.3 Data Aggregation

The end goal of the analysis on Stack Overflow and GitHub data was to create topic models capable of deciphering the hidden patterns of underlying topics that represent a user's expertise and activity on a platform. Tian et al. [15] modeled user topical interest and expertise by building LDA models on user activity (user profile) data. They have shown that a Stack Overflow user's expertise can be extracted using a topic model applied over a set of textual documents consisting of the user's activity on the platform collected into a user profile.

In order to perform cross-platform analysis of GitHub and Stack Overflow users' expertise, we modeled their expertise through LDA models. LDA topic models are fitted on a set of documents. In order to fit topic models on GitHub and Stack Overflow user data, one needs to define what a document consists of, and how to extract a user's profile from a platform such as GitHub or Stack Overflow. This process is explained in the following three subsections (i.e., Section 3.3.1, Section 3.3.2, and Section 3.3.3).

3.3.1 Extracting Stack Overflow User Profiles

When deciding on how to create documents for the LDA topic model, we define a document as a single user profile. Each user who is present in the data set of 83,550 linked users described in Section 3.2.2 has one document (user profile) describing their activity on Stack Overflow. This document contains all of their activity on the platform. Extracting a user's activity (user profile) from all the public data stored about a user started by inspecting each table of the SOTorrent database schema 4 for attributes that contain relevant and meaningful textual information about a user. The relevant textual attributes included into the user profiles are *badge names* obtained by a user, the *about me* description from the user profile, *questions* posted by a user, *answers* given by a user, *titles* and *tags* of posts that a user participated in, and *comments* made by a user to any post.

A detailed description of how each attribute from the user profiles was processed can be found in Table 3.3.1. To restore the semantic context of the user's answer (*Post Answer*), the question being asked is concatenated before the answer, thus

Table 2: Stack Overflow user profile extraction.

Attribute Name	**Attribute(s)**	**Description**
Badges	Badges.Name	Concatenation of list of badges obtained by the user
About Me	Users.AboutMe	Stack Overflow user profile's about me description
Post Answer	Posts.Body, AcceptedAnswerId	The user's each individual answer concatenated with the question it is related to
Post Question	Posts.Body	The user's each individual question concatenated with the accepted answer it is related to
Title and Tags for Questions	Posts.Tags, Posts.Title	Concatenation of post tags and title for each question that the user asked
Title and Tags for Answers	Posts.Title, Posts.Tags	Concatenation of post tags and title for each answer that the user provided
Comments	Comments.Text, Posts.Body, Posts.Title	Concatenation of the user's each individual comment and the post (question or answer) it is related to

creating a question-answer pair. For the same reason, the *Post Question* attribute is getting the accepted answer concatenated in, thus adding more contextual clarity to the user's question. This process is being applied to the *Comments* attribute as well, creating question-comment or answer-comment pairs, depending on what the comment is related to. At the end of this process, all attributes in the user profile are merged into one large document representing a user's full activity on the SO platform.

3.3.2 Extracting GitHub User Profiles

Each user who is present in the data set of 83,550 linked users described in Section 3.2.2 has one document (user profile) describing their activity on GitHub. We extracted a user profile by inspecting each table of the GHTorrent database schema 5 for attributes that contain relevant textual information about a user. The textual attributes selected to be part of the GH user profile data include the *names*, *labels*, *languages* used, *description of the repository* that a user owns, as well as their *commit* and *pull request comments* posted on GitHub.

Table 3: GitHub user profile extraction.

Attribute Name	**Attribute(s)**	**Description**
Project Name, Description and Metadata	Projects.[name, description, language], Repo-Labels.name	Description of each user's project together with the repository's name, description, languages used, repository labels it contains.
Commit-Comments	Commit-Comments.body	List of user's commit comments.
Code-review Comments	Pull-Request-Comments.body	List of user's code review (pull request) comments.

A detailed description of how each attribute from the user profiles was processed can be found in Table 3. Attributes such as the name, description and labels of a repository, alongside the programming languages used are valuable information describing the content of a repository on GitHub. Discussion between users in the form of comments to commits and pull request comments are also essential to establish contextual details about the project, thus these attributes have been included in the user profile. All the above listed attributes get merged into one large document representing a user's full activity on the platform.

3.3.3 Creating Time Based User Profiles

After defining how to extract user profiles from both Stack Overflow and GitHub, it is worth factoring in a *timeline* into the user profile. The simplest approach would be to model the recent and past activity of users by splitting their activity into before and after a specific date. Being able to model past and recent user profile data separately creates opportunities for comparative analyses to be done on the evolution of expertise of GitHub and Stack Overflow users. We decided to define recent activity as the activity performed by a user in the last 3 years. Since our study took place in 2019, we split user activity profile into recent and past, i.e., any activity dated before *January 1st 2016* is considered as *past activity*, while activity that took place after *January 1st 2016* is defined as *recent activity*.

Based on our definition of recent activity, three types of data sets have been created: 1) data containing only recent activity, 2) only past activity data, and 3) both past and recent activity data. The past activity data set includes data from the day the platform started recording data up to *January 1st 2016*, while the recent data set contains data only after *January 1st 2016*, up to the date of data collection. The "full" data set contains all the data for a platform, and has no time segmentation. This can be also said that the "full" data set is the concatenation of the past and recent data sets. In the following sections this data segmentation will be referred to as *past, recent and full*, alongside the platform abbreviation SO and GH for Stack

Overflow and GitHub, respectively, i.e., *GH-past, GH-recent, GH-full, SO-past, SO-recent and SO-full.*

3.4 Data Cleaning

Based on the common practise text processing routines used for software engineering domain-specific textual data presented in *Background* (see Section 2.1.2), we developed a *user-level* and a *corpus-level* pre-processing routines.

3.4.1 User-Level Text Processing

The user-level pre-processing routine is executed on each user's aggregated textual data during the user profile extraction processing described in Section 3.3.1 and Section 3.3.2. The user-level routine involved the removal of HTML links, symbols and tags, proceeded by stop-word, mentions (using @) and multiple white space removal. Finally, a tokenization technique is applied, then individual tokens consisting of only numbers or punctuation with no words present in the token are removed. This data processing routine is executed on the already aggregated (concatenated) data obtained using the description in Section 3.3.1 and Section 3.3.2.

3.4.2 Corpus-Level Text Processing

The corpus-level pre-processing routine is executed on each text corpus (GH-past, GH-recent, GH-full, SO-past, SO-recent and SO-full, as described in Section 3.3.3) before the model fitting, but after the data aggregation process. The corpus-level pre-processing includes tokenization, standardization of tokens, frequent phrase detection, then removing tokens that include only numbers and punctuation (if any). A subset of symbols and punctuation are also removed, keeping symbols such as “+”, “_”, “.” and

"#", which often appear in programming contexts. Finally, rare and very common tokens are removed by filtering out tokens that occur in less than 10 documents or more than 50% of the documents. In this context, standardization of tokens means re-writing common tokens spelled multiple ways as a unique token. Examples of such common tokens include "node.js", "node-js", "node_js" for "nodejs"; "angular.js", "angular-js", "angular_js" for "angularjs"; or "js" for "javascript', and many more. A list of the most frequent phrases detected in the GH-full and SO-full data sets can be found in Appendix A.1. These common phrases used in the software engineering domain have a different meaning as a phrase (bigram), thus they have been re-written as a single word (unigram), using a "_" character connecting the phrases. As a final note on data cleaning, both user- and corpus-level pre-processing routines are designed based on common best practises of previous data cleaning routines (e.g., [15], [16], [17], [18], [19], and [14]) applied to software engineering related textual data.

3.5 Topic Modeling

The inspiration to use topic modeling to learn cross-platform developer expertise came from Tian et al.'s work [15]. The authors used topic models to predict the best answerer for a new question on Stack Overflow. Their approach learnt user topical expertise and interest levels by profiling each user's previous activity and reputation on Stack Overflow. When working with large collections of textual data in software engineering applications the most popular topic modeling technique to use is Latent Dirichlet Allocation (LDA) [16].

LDA models have been previously used to learn semantic similarity between user profiles, posts ([15]) and even source codes ([36]) on Stack Overflow. The proposed novel approach is creating a cross-platform developer expertise extraction task consisting of topical expertise learnt from topic analysis on user profiles extracted through mining public software repository platforms such as Stack Overflow and GitHub. Campbell et al.'s explanation on the use of LDA models shows that extracting such expertise is possible by providing a summary or ranking of the weighted list of topic words for each topic that the user is present in or belongs to [16].

3.5.1 Model Setup

As described in Section 3.3.3, there are three different versions of Stack Overflow and GitHub corpora: past, recent and full corpus. Separate topic models are fitted on GitHub and Stack Overflow corpora. Separate models are created for each one of the past, recent and full versions of the corpus, thus a total of six topic models are fitted, three on processed Stack Overflow data (named *SO-past, SO-recent and SO-full*) and three on processed GitHub data (named *GH-past, GH-recent and GH-full*). The dictionary of terms used to fit the topic models is the same, and consistent across all three versions of a corpus. Each corpus consists of a collection of documents, which are user profiles. There are 83,550 linked users in the data set, each user having only one document (user profile) describing their activity, thus there are always 83,550 documents in each version of a corpus. It is worth noting that some users may be inactive on one or both of the platforms, thus making their user profile be an empty document due to the lack of activity on the platform. In the original intersected version of the GitHub and Stack Overflow data set, Vasilescu et al. [46] defined an active user as someone who authored at least one commit, asked or answered a question between a specific period of time. In our project the definition of an active user is very similar to the one provided by Vasilescu et al. An active Stack Overflow user is defined as a user who posted a question or answer within a specified time range (past or recent time segmentation). An active GitHub user is defined as a user who performed a commit or a pull request within a specified time range (past or recent time segmentation).

3.5.2 Parameter Selection and Model Optimization

Treude and Wagner [17] studied good parameter configurations of LDA models for GitHub and Stack Overflow data and concluded that text corpora in the software engineering domain have special properties and characteristics. Thus, to achieve a good model fit, LDA models require different parameter configurations than the popular rules of thumb.

When choosing α and β, hyper-parameter optimization is performed by combining the best practices of Bangash et al. [28] and Treude and Wagner [17]' work on finding the right hyper-parameters. We used Hoffman et al.'s implementation of *Online learning for LDA models* [77] to learn an asymmetric prior from the data for both α and β hyper-parameters.

The models fitted in this project use 400 iterations through the corpus when inferring the topic distribution of a corpus (instead of default 50 iterations). This should assure the model's stability, as model convergence is checked before finishing the fitting process. For the random state of the model training initialization, we used a fixed seed to promote reproducibility.

When deciding on k, we followed the combination of methodologies offered by Bangash et al. [28] and Treude and Wagner [17]. We defined a parameter search space of [3, 100], then performed a hyper-parameter optimization against this search space, with the evaluation metric selected in Section 3.5.3. We define the upper bound of 100 topics based on the assumption that more than 100 topics would be difficult to label, interpret and keep track of. The parameters α and β are being learnt during model fitting, thus the hyper-parameter optimization can be performed only on the number of topics k parameter, which was selected based on whichever model maximized the evaluation metric. Optionally, the β parameter could be included in the hyper-parameter optimization process by defining a commonly used range of values, [0.001, 1] as a search space.

3.5.3 Evaluation of Topic Models

Evaluating LDA models is a highly debated topic within the topic modeling community. There are a few general rules to follow, but which ones to follow and to what extend is highly subjective. In previous studies, there are examples for uses of topic coherence, task-based prediction accuracy, likelihood-based and perplexity based evaluation metrics. For instance, Treude and Wagner [17] used perplexity for model evaluation, but they mention that perplexity is not the only metric which can be used to evaluate topic models, and in their future work they suggest using conciseness or coherence. Boyd-Graber et al. [19] also suggest the use of topic coherence as an evaluation metric. Röder et al. [42]'s topic coherence measures seem to be the current state-of-the-art in topic coherence advancements, thus in this work, we use their most promising coherence measures (*C_v, C_umass, C_npmi and C_uci*) as evaluation metrics. The final model was selected based on a majority agreement of the highest coherence scores between all four measures considered.

3.6 Expertise Ground Truth Study

Bangash et al. [28] validated their work by manually reading a random sample of their input documents and checking the LDA topics extracted. This process can be considered highly subjective, thus we looked for a more objective, and quantitative process. After careful consideration, we decided to collect human annotations via a developer expertise study to obtain a ground truth. Therefore, we designed a developer expertise study for collecting our ground truth data.

3.6.1 Study Setup

Using the data set created in Section 3.2.2 and based on the definition of an active user in Section 3.5.1, we selected a random sample of 100 active users on both GitHub and Stack Overflow. The list of profile page URLs of 100 random Stack Overflow and GitHub users can be found in the Appendix A.2. A visual of sample study questions can be seen in Figure 17, Appendix A.2. The list of 100 user profiles was split into lists of 10 user profiles, as the labeling of 100 user profiles would be too time consuming for the study participants. Each list of 10 user profiles became part of one expertise study, each containing 10 non-overlapping Stack Overflow and GitHub user profile URLs. It is worth noting that the studies were intentionally designed to not contain overlapping GitHub and Stack Overflow user profiles. We wanted to avoid participant bias and ensure that a study participant's annotations of a Stack Overflow profile would not be influenced by the same user's GitHub profile.

The study contained a list of 10 Stack Overflow and 10 GitHub user profile URLs and had the following instructions: "Enter *20* comma-separated words describing each user's expertise. Your answers need to come from evaluating a user's full activity (i.e., every publicly available data that you can see and click-through) on GitHub or Stack Overflow". Additionally, we provided the following example answer for a fictional user: *"pytorch, CNN, RNN, auto-encoders, Keras, git, TensorFlow, python, java, C#, web_dev, machine_learning, random_forest, SVM, NLP, Java_streams, distributed_computing, parallel_computing, R, statistics, visualization"*. The study participants were Computer Science graduate students at Carleton University. Participants received no remuneration except for small bonus marks by the instructor of the course.

Each expertise study was completed by 2 participants. Overall, we recruited 20 participants who provided us with 2 separate ground truth human annotations for each one of the 10 studies. Participants had several weeks to complete the expertise ground truth study. The study asked for the participant's name to be able to clarify their annotations if needed. Once we have collected the expertise ground truth, we removed the names of participants to protect participants' identity and confidentiality.

3.6.2 Processing of Human Annotations

The ground truth human annotations obtained via the study were pre-processed using the same text processing routines used to clean the entire data set (see Section 3.4). We then faced several challenges related to misspelled words, and expressions or phrases being not in the dictionary of the data set. To avoid inconsistencies between the corpora and human annotations, we performed manual checks on all annotations to correct misspelled words, remove duplicate annotations, remove annotations that made no sense, and re-write phrases out of vocabulary as individual words. In this process we acted as a third annotator, whose job is to verify the validity of the human annotations.

One last concern related to human annotations was that not every participant provided exactly the required amount of 20 keywords per user. We intended to represent the expertise of a developer with a minimum amount of words extracted from a topic. Hindle et al. [30] recommend using a list of 20 words when describing an LDA topic, thus if a user's expertise belongs to only one topic, there should be a minimum of 20 words describing the user's expertise. Since not all participants entered exactly 20 labels for a user's expertise, the length of the human annotations was inconsistent, which represented a problem when trying to measure annotator inter-agreement scores. Cohen's Kappa [78] is a statistic that could be used to measure inter-rater agreement between annotations. Cohen's Kappa requires the annotations to be the same length, thus we could not calculate this for our ground truth data. Instead, we used Krippendorff's alpha coefficient [79] for calculating an inter-rater agreement statistic. The last debate related to human annotations was about the aggregation of human annotations. When comparing an algorithm's prediction of expertise against the ground truth we need to be able to easily calculate one or more metrics to determine the accuracy of the algorithm. Having two annotations is inconvenient, as two separate accuracy values will be obtained, then the averaging of the two values could be calculated in multiple ways (weighted average, micro or macro average). To avoid such a situation, we decided to aggregate annotations via the set union, and obtain one annotation only. We applied the set union operation instead of the set intersect operation to create a rich sense of expertise by having a much larger sized set of annotations.

3.7 Data Analysis and Algorithm Design

Sections 3.1 through 3.4 were focused on how to acquire and process the data. Section 3.5 explained the modeling of data, and Section 3.6 was concerned with establishing the ground truth knowledge. All of these sections serve as a build-up towards the culmination of methodology, Section 3.7, outlining four separate tasks (*Expertise Extraction, Cross-platform Expertise, Cross-platform Transferable Knowledge, and Expertise Evolution*) which represent four separate analyses performed to answer research questions described in Sections 3.7.1 through 3.7.4. Each of these sections will describe the path from either modeling to prediction or modeling to insight. For the tasks in Sections 3.7.2 through 3.7.4 the past and recent segmentation of data explained in Section 3.3.3 are used for both Stack Overflow and GitHub data. Only the expertise extraction task uses the full version (see Section 3.3.3) of the data, with no segmentation.

3.7.1 Expertise Extraction

Our first research question (i.e., RQ1 from Section 1.3) is concerned with extracting major expertise areas of Stack Overflow and GitHub users and comparing expertise trends on Stack Overflow and GitHub. The goal of this task is to design one or more techniques to extract the expertise of a user, then check the accuracy of such techniques against the ground truth knowledge from Section 3.6. Expertise trends comparison was performed by comparing the topics of the LDA models that produced the most accurate expertise words extracted for both GitHub and Stack Overflow users. For the expertise extraction task, three novel techniques have been designed, and they are described in detail below.

3.7.1.1 Technique 1: Baseline LDA Based Expertise Extraction

Our first technique takes a fitted LDA model and leverages the insights gained through learning the topic distribution of documents (user profiles), then outputs the extracted expertise words for each user based on thresholds applied to the likelihood of a user belonging to an LDA topic. The extracted expertise words take the form of topic words representing topics that a user most likely belongs to. The formal algorithm for *Baseline LDA Based Expertise Extraction* can be found in Algorithm 2.

This algorithm takes two parameters: a fitted, already optimized *LDA* topic model, and a probability threshold, T, which is a value strictly between 0 and 1. The first major step in the algorithm is to create a user-topic mapping for each user using the topic distribution present in the topic-document matrix. In lines 11–13 of Algorithm 2, a topic becomes associated with a user if the probability of that topic being present in the user's profile is larger than or equal to the probability threshold T. It is important to note that the probability threshold value T is chosen empirically, based on running numerous simulations with different threshold values

Algorithm 2 Baseline LDA Based Expertise Extraction.

```
Require: $LDA$: Topic Model, $threshold$: Probability Threshold
Get Topic-Document Matrix $M$ from fitted $LDA$ topic model
User-Topic Mapping = { }
Expertise-Predictions = { }

// Create User-Topic Mapping of each user
for all users $user\_ID$ in the data set do
    Get user profile document $d$ of user $user\_ID$
    Get topic distribution $TD$ of document $d$ from matrix $M$
    listOfTopics = [ ]
    for all topics $t$ with non-zero probability in $TD$ do
        if probability of topic $t \geq threshold$ then
            Add topic $t$ to listOfTopics
        end if
    end for
    User-Topic Mapping[ $user\_ID$ ] = listOfTopics

    // Extract expertise for each user
    listOfWords = [ ]
    for all topic $t$ in User-Topic Mapping[ $user\_ID$ ] do
        $TW$ = Get top-20 topic words that describe topic $t$
        Add topic words $TW$ to listOfWords
    end for
    Apply a word ranking algorithm to listOfWords
    Expertise-Predictions[ $user\_ID$ ] = sorted listOfWords
end for
return Expertise-Predictions
```

between 0 and 1, and choosing a probability threshold T which maximizes the desired evaluation metric (in this case cosine similarity score) of the task. After various simulations the threshold value T can be chosen by the modeler or user, and it can be optimized for each data set, just as a hyper-parameter. The topic distribution of each user profile document allows the extraction of topics that most likely describe a user's expertise. Once the user-topic mapping is created, the top-20 topic words from each topic associated with the user are collected into a list of candidate expertise words. Next, a ranking of the candidate expertise words is created by applying a word ranking metric commonly used in topic modeling called *relevance*. Sievert et al. [80] proposed *relevance* as a method for ranking terms within topics, which is currently highly used in open source libraries such as *pyLDAvis*[13]. Sievert et al. defined the *relevance* of a topic-word w to a topic k given a regularization parameter λ bound by [0,1] as Equation 2:

$$Relevance(w, k|\lambda) = \lambda \log(\phi_{kw}) + (1-\lambda)\log(\frac{\phi_{kw}}{p_w}) \tag{2}$$

where ϕ_{kw} is the probability of term $w \in \{1, ..., V\}$ for topic $k \in \{1,...,K\}$, where V is the number of tokens in the vocabulary, and K is the number of topics that the LDA model was fitted with. Also, p_w represents the marginal probability of term w in the corpus. Sievert et al. [80] mentioned that the ϕ_{kw} function in this LDA setting could be estimated by Variational Bayes methods or Collapsed Gibbs Sampling, while the p_w marginal distribution could be calculated empirically from the distribution of the text corpus. The λ regularization term can to be chosen by the modeler or user, and it could be optimized for each data set, just as a hyper-parameter. For instance, in Sievert et al. [80]'s work the optimal value of λ was *0.6*.

Once the candidate expertise words have been ranked, the top-n ranked words

[13] `https://github.com/bmabey/pyLDAvis`

are returned as the expertise words describing a user. n can be chosen to be a fixed length for the set of expertise words, or the technique can be used to predict as many words as human annotated labels that exist on a user. A step-by-step visualization of the entire Algorithm 2 is shown in Figure 6.

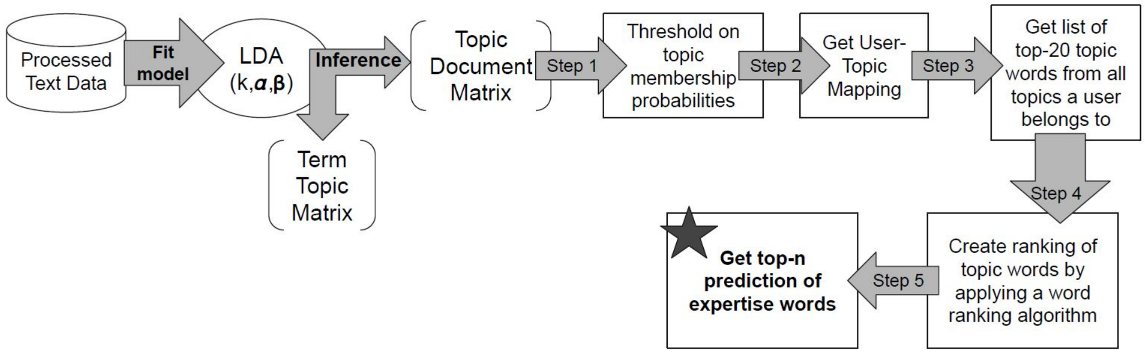

Processed Text Data
Fit model
LDA (k,α,β)
Inference
Term Topic Matrix
Topic Document Matrix
Step 1
Threshold on topic membership probabilities
Step 2
Get User-Topic Mapping
Step 3
Get list of top-20 topic words from all topics a user belongs to
Step 4
Create ranking of topic words by applying a word ranking algorithm
Step 5
Get top-n prediction of expertise words

Figure 6: Baseline LDA based expertise extraction.

3.7.1.2 Technique 2: Improved LDA Based Expertise Extraction

Our second technique introduces the idea of generating user and topic embeddings in the same vector space, so their similarity could be computed. The creation of the user-topic mapping is accomplished by comparing cosine similarities between the user and topic embeddings, thus associating a user with the most similar topics.

All unique words in a user profile are contributing to generate user embeddings using aggregated latent scores of each word present in the LDA model's term-topic matrix. The formal algorithm for *Improved LDA Based User Embedding Generation* is reported in Algorithm 3.

The *Improved LDA Based User Embedding Generation* algorithm takes as input the collection of user profiles, and a term-topic matrix learned during inference of a fitted LDA topic model. In line 4 of Algorithm 3, a set of words is created, which contains all the words appearing in a user's profile. In lines 7–10, this set of words is iterated over, and for each word a vector of latent scores is obtained from the input

Algorithm 3 Improved LDA Based User Embedding Generation.

```
Require: $M$: Term-Topic Matrix, $D$: User Profile Documents
1: listOfEmbeddings = [ ]
2: for all users $user\_ID$ in the data set do
3:     Get user profile document $d$ for user $user\_ID$ from $D$
4:     Get unique set words $WordSet$ in document $d$
5:     listOfVectors = [ ]
6:
7:     for each word $w$ in $WordSet$ do
8:         Get $1 \times k$ vector $v$ of latent scores for word $w$ from matrix $M$
9:         Add vector $v$ to listOfVectors
10:    end for
11:    Convert listOfVectors to matrix $LV$ with dimensions $n \times k$; $n = length(WordSet)$
12:    Perform column-wise max-pooling or average-pooling to reduce matrix $LV$ to
       a user embedding vector $emb$, dimension $1 \times k$
13:    Add user embedding vector $emb$ to listOfEmbeddings
14: end for
15: Convert listOfEmbeddings to matrix $E$ with dimensions $u \times k$; where $u$ = number
    of users in the data set
16: return Matrix $E$
```

term-topic matrix. In line 12, a column-wise aggregation is performed by applying either max-pooling or average-pooling down-sampling technique. There are 2 variations of Algorithm 3. In order to explore which down-sampling technique works best, we have been performed experiments using both max and average-pooling, thus obtaining a convenient $1 \times k$ user embedding vector for each user.

Applying the *Max-pooling down-sampling technique* in this context is rather unusual since Max-pooling originated from the field of deep learning and most often is used to reduce the spatial dimensions on a convolution neural network. The objective is to down-sample the input representation by reducing its dimensionality, which allows assumptions to be made about the features present in the image data [81]. In deep learning, the gains of down-sampling are significant: it helps combat over-fitting by making the representation more abstract, and it reduces the computational cost by having fewer parameters to be learnt. A variation of the *Max-pooling down-sampling* *technique* can also be used on word embeddings as a way to perform feature selection. Applying a column-wise average or maximum function is a common word embedding aggregation technique [82] used to create feature vectors from word embeddings.

Topic embeddings are generated using aggregated latent scores of top-20 topic words of each topic present in a fitted LDA model. The formal algorithm for *Improved LDA Based Topic Embedding Generation* can be found in Algorithm 4.

Algorithm 4 Improved LDA Based Topic Embedding Generation.

```
Require: $LDA$: Topic Model, $M$: Term-Topic Matrix
 1: listOfEmbeddings = [ ]
 2: for each topic $T_i$ in $LDA$ fitted model do
 3:     Get top-20 topic words $TW$ describing topic $T_i$
 4:     listOfVectors = [ ]
 5:
 6:     for each word $w$ in $TW$ do
 7:         Get $1 \times k$ vector $v$ of latent scores for word $w$ from matrix $M$
 8:         Add vector $v$ to listOfVectors
 9:     end for
10:     Convert listOfVectors to matrix $LV$ with dimensions $n \times k$; where $n = 20$, $k =$
        number of topics in $LDA$ model
11:     Perform column-wise max-pooling or average-pooling to reduce matrix $LV$ to
        a topic embedding vector $emb$, dimension $1 \times k$
12:     Add topic embedding vector $emb$ to listOfEmbeddings
13: end for
14: Convert listOfEmbeddings to matrix $E$ with dimensions $k \times k$
15: return Matrix $E$
```

The *Improved LDA Based Topic Embedding Generation* algorithm takes as input a fitted LDA model and the term-topic matrix learned during inference. In lines 2 and 3 of Algorithm 4, for each topic in the LDA model a set of top-20 topic words best representing that topic is obtained from the LDA model. In lines 6–19, this set of topic words is iterated over, and for each topic word a vector of latent scores is obtained from the input term-topic matrix. In line 11, a column-wise aggregation is performed by applying either max-pooling or average-pooling down-sampling technique. Same as for the *Improved LDA Based User Embedding Generation*, there are 2 variations of Algorithm 4 too, by applying both max and average-pooling, and experimenting which down-sampling technique works best. After applying this process for each topic, a convenient $1 \times k$ topic embedding vector is obtained.

The prediction of expertise is accomplished through a collection of topic words representing topics that a user most likely belongs to based on thresholds applied to cosine similarities calculated between a user embedding and each topic embedding. The formal algorithm for the overall *Expertise Extraction using LDA Based User and Topic Embeddings* can be seen in Algorithm 5.

The *Improved LDA Based Expertise Extraction* algorithm takes as input the collection of user profiles, an already fitted *LDA* topic model, and a cosine similarity threshold, which is value strictly between 0 and 1. After getting the term-topic matrix from the *LDA* model, Algorithm 3 and 4 are executed to generate user and topic embeddings. Line 7 of Algorithm 5 represents the iteration over each user embedding. In lines 11 and 12, the cosine similarities between a user embedding and each topic embedding are calculated and stored. In line 15, the non-zero cosine similarities previously calculated are associated with similarities between a user and a topic. In lines 16–20, a topic becomes associated with a user if the cosine similarity between their respective embeddings is larger than or equal to the input cosine similarity threshold value. In lines 26–29, the top-20 topic words from each topic associated with a user are collected into a list of extracted expertise words. Next, a ranking of the candidate expertise words is created by applying the same word ranking metric used in Algorithm 2, *relevance*. Similar to the first novel technique, once the candidate expertise words have been ranked, the top-n ranked words are returned as the expertise words describing a user. A step-by-step visualization of the entire Algorithm 5 can be seen in Figure 7.

Algorithm 5 Improved LDA Based Expertise Extraction.

```
Require: $D$: User Profile Documents, $LDA$: Topic Model, $Experimentold$: Cosine
    Similarity Threshold
Get Term-Topic Matrix $M$ learned during inference of fitted $LDA$ topic model
User-Topic Mapping = { }
Expertise-Predictions = { }
Run LDA based User Embedding Generation to get $UserEmb$ matrix
Run LDA based Topic Embedding Generation to get $TopicEmb$ matrix

for each user embedding $U$ in $UserEmb$ do
    listOfTopics = [ ]
    user_to_topic_sim = [ ]
    for each topic embedding $T$ in $TopicEmb$ do
        Computer cosine similarity $SIM$ between user embedding $U$ and topic embedding $T$
        Add $SIM$ to user_to_topic_sim
    end for

    Get list of topics $LT$ associated with non-zero cosine similarities in user_to_topic_sim
    for topic $t$ in $LT$ do
        if user_to_topic_sim[ $t$ ] $\geq threshold$ then
            Add topic $t$ to listOfTopics
        end if
    end for
    Get $User\_ID$ associated with user embedding $U$
    User-Topic Mapping[ $User\_ID$ ] = listOfTopics

    // Extract expertise for each user
    listOfWords = [ ]
    for all topic $t$ in User-Topic Mapping[ $user\_ID$ ] do
        $TW$ = Get top-20 topic words that describe topic $t$
        Add topic words $TW$ to listOfWords
    end for
    Apply a word ranking algorithm to listOfWords
    Expertise-Predictions[ $user\_ID$ ] = sorted listOfWords
end for
return Expertise-Predictions
```

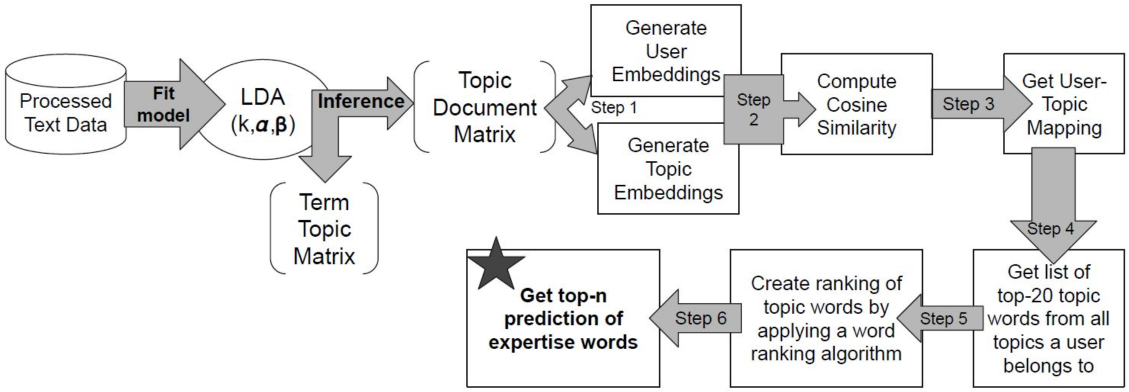

Processed Text Data
Fit model
LDA (k,α,β)
Inference
Term Topic Matrix
Topic Document Matrix
Step 1
Generate User Embeddings
Generate Topic Embeddings
Step 2
Compute Cosine Similarity
Step 3
Get User-Topic Mapping
Step 4
Get list of top-20 topic words from all topics a user belongs to
Step 5
Create ranking of topic words by applying a word ranking algorithm
Step 6
Get top-n prediction of expertise words

Figure 7: Improved LDA based expertise extraction

3.7.1.3 Technique 3: Pre-trained Word2Vec Based Expertise Extraction

The third and last novel technique is a one step change on the second technique, *Improved LDA Based Expertise Extraction*. The only difference is that the LDA based latent score representation of words is replaced by a continuous vector representation of words coming from the pre-trained Word2Vec model. This model is publicly available, as it is Efstathiou et al. [18]'s work on *Word Embeddings for the Software Engineering Domain*. The Word2Vec model [20] was trained over 15GB of textual data from Stack Overflow posts, thus it captures software engineering specific contextual semantics. Making this change to technique 2 should be beneficial, as, in theory, the pre-trained Word2Vec model should provide a better representation of words, from which better user and topic embeddings could be generated. The formal algorithms for the *Pre-trained Word2Vec Based User and Topic Embedding Generation* can been seen in Algorithms 6 and 7. Changes between the *LDA Based* and *Pre-trained Word2Vec Based* User and Topic Embedding Generation algorithms are highlighted with the color blue.

The *Pre-trained Word2Vec Based User Embedding Generation* algorithm takes as input all user profile documents and a pre-trained Word2Vec model. In line 8 of Algorithm 6, a user profile's each word's Word2Vec vector representation is queried.

Algorithm 6 Pre-trained Word2Vec Based User Embedding Generation.

```
Require: $D$: User Profile Documents,$word2vec$: Pre-trained Model
listOfEmbeddings = [ ]
for all users $user\_ID$ in the data set do
    Get user profile document $d$ for user $user\_ID$ from $D$
    Get unique set words $WordSet$ in document $d$
    listOfVectors = [ ]

    for each word $w$ in $WordSet$ do
        Look up vector representation $v$ of word $w$ in pre-trained $Word2Vec$ model
        Add vector $v$ to listOfVectors
    end for
    Convert listOfVectors to matrix $LV$ with dimensions $n \times d$; $n = length(WordSet)$,$d$ = dimensionality of $Word2Vec$ model
    Perform column-wise max-pooling or average-pooling to reduce matrix $LV$ to a user embedding vector $emb$, dimension $1 \times d$
    Add user embedding vector $emb$ to listOfEmbeddings
end for
Convert listOfEmbeddings to matrix $E$ with dimensions $u \times d$; where $u$ = number of users in the data set
return  Matrix $E$
```

Contrary to common practice, out of vocabulary words are not assigned a vector of 0's, but rather skipped, as it would falsify the average-pooling down-sampling results, which are obtained in line 12. The algorithm produces $1 \times d$ dimensional user embedding vectors, where d is 200, as Efstathiou et al. [18] trained their model with vector dimensions of 200 features. In the end, a matrix with dimensions $u \times d$ is returned, where u is the number of users in the data set.

Algorithm 7 Pre-trained Word2Vec Based Topic Embedding Generation.

```
Require: $LDA$: Topic Model,$Word2Vec$: Pre-trained Model
 1: listOfEmbeddings = [ ]
 2: for each topic $T_i$ in $LDA$ fitted model do
 3:     Get top-20 topic words $TW$ describing topic $T_i$
 4:     listOfVectors = [ ]
 5:
 6:     for each word $w$ in $TW$ do
 7:         Look up vector representation $v$ of word $w$ in pre-trained $Word2Vec$
            model
 8:         Add vector $v$ to listOfVectors
 9:     end for
10:     Convert listOfVectors to matrix $LV$ with dimensions $n \times d$; where $n = 20$,$d =$
        dimensionality of $Word2Vec$ model
11:     Perform column-wise max-pooling or average-pooling to reduce matrix $LV$ to
        a topic embedding vector $emb$, dimension $1 \times d$
12:     Add topic embedding vector $emb$ to listOfEmbeddings
13: end for
14: Convert listOfEmbeddings to matrix $E$ with dimensions $k \times d$, $k$ = number of
    topics in $LDA$ model
15: return Matrix $E$
```

The *Pre-trained Word2Vec Based Topic Embedding Generation* algorithm takes as input an LDA topic model, and a pre-trained word2vec model. In line 7 of Algorithm 7, each topic word's Word2Vec vector representation is queried. The algorithm produces $1 \times d$ dimensional topic embedding vectors, where d is 200. In the end, a matrix with dimensions $k \times d$ is returned, where k is the number of topics in the input LDA model.

The prediction of expertise for the third technique is the same process as in Algorithm 5. The formal algorithm for the overall *Expertise Extraction Using Pre-trained Word2Vec Based User and Topic Embeddings* can be seen in Algorithm 8.

The *Pre-trained Word2Vec Based Expertise Extraction* algorithm takes as input the collection of user profiles, an already optimized, fitted LDA topic model, and a cosine similarity threshold, T, which is a value strictly between 0 and 1. The only major change compared to the second technique is that in lines 4 and 5 Algorithms 6 and 7 are executed to generate pre-trained Word2Vec based user and topic embeddings. The rest of the technique follows the same process as the second technique. A step-by-step visualization of the entire Algorithm 8 can be seen in Figure 8.

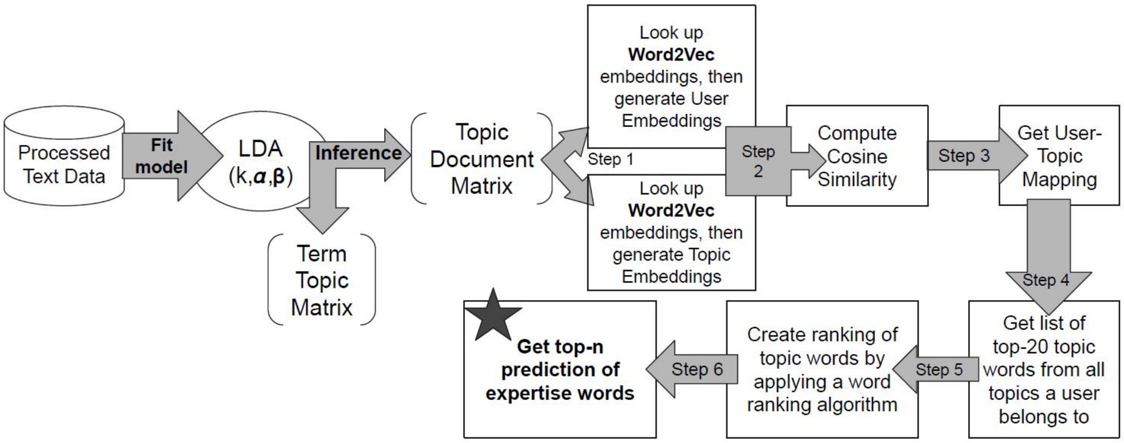

Processed Text Data
Fit model
LDA (k,α,β)
Inference
Term Topic Matrix
Topic Document Matrix
Step 1
Look up Word2Vec embeddings, then generate User Embeddings
Look up Word2Vec embeddings, then generate Topic Embeddings
Step 2
Compute Cosine Similarity
Step 3
Get User-Topic Mapping
Step 4
Get list of top-20 topic words from all topics a user belongs to
Step 5
Create ranking of topic words by applying a word ranking algorithm
Step 6
Get top-n prediction of expertise words

Figure 8: Pre-trained Word2Vec based expertise extraction.

The above three techniques were carefully designed through multiple rounds of simulation and various changes to the implementation were performed. Algorithm 2, 5 and 8 presented above are the final versions that were proven to work the best. The most noteworthy change was considering topic probability or cosine similarity thresholding techniques instead of clustering users into similar groups. Algorithms such as *k-means* and *agglomerative* clustering were tested to cluster user embeddings and topic embeddings in the same vector space. The theoretical idea was that user

Algorithm 8 Pre-trained Word2Vec Based Expertise Extraction.

```
Require: $D$: User Profile Documents, $LDA$: Topic Model, $threshold$: Cosine Similarity Threshold,$Word2Vec$: Pre-trained Model
Get Term-Topic Matrix $M$ learned during inference of fitted $LDA$ topic model
User-Topic Mapping = { }
Expertise-Predictions = { }
Run Word2Vec based User Embedding Generation to get $UserEmb$ matrix
Run Word2Vec based Topic Embedding Generation to get $TopicEmb$ matrix

for each user embedding $U$ in $UserEmb$ do
    listOfTopics = [ ]
    user_to_topic_sim = [ ]
    for each topic embedding $T$ in $TopicEmb$ do
        Computer cosine similarity $SIM$ between user embedding $U$ and topic embedding $T$
        Add $SIM$ to user_to_topic_sim
    end for

    Get list of topics $LT$ associated with non-zero cosine similarities in user_to_topic_sim
    for topic $t$ in $LT$ do
        if user_to_topic_sim[ $t$ ] $\geq threshold$ then
            Add topic $t$ to listOfTopics
        end if
    end for
    Get $User\_ID$ associated with user embedding $U$
    User-Topic Mapping[ $User\_ID$ ] = listOfTopics

    // Extract expertise for each user
    listOfWords = [ ]
    for all topic $t$ in User-Topic Mapping[ $user\_ID$ ] do
        $TW$ = Get top-20 topic words that describe topic $t$
        Add topic words $TW$ to listOfWords
    end for
    Apply a word ranking algorithm to listOfWords
    Expertise-Predictions[ $user\_ID$ ] = sorted listOfWords
end for
return Expertise-Predictions
```

embeddings which get grouped into the same cluster as a topic embedding represent a user-topic association, thus a user would belong to the topic(s) that their user embedding got clustered into. This idea, unfortunately, did not work out, as the curse of dimensionality did not allow for meaningful clusters to be formed. Dimensionality reduction could have been applied through techniques such as PCA [83] or t-SNE [84], but this was undesirable since the interpretation of cluster memberships was a priority, and interpreting principal components instead of any similarity scores was inconvenient. Furthermore, user and topic embeddings are already a form of down-sampling. Applying even further reduction of the data would create questions about the validity of the techniques by too high loss of information due to dimensionality reduction. Instead, thresholding on topic probability and cosine similarity seemed to be more ideal for the creation of user-topic association or mapping.

3.7.1.4 Evaluation of Techniques

To evaluate the performance of the above techniques, three metrics have been considered: *BLEU score, Jaccard similarity, and Cosine similarity.* The formal definitions and more details about these metrics can be found in the Background chapter, i.e., Section 2.1.5. When designing these techniques, the task seemed closer to text summarization, rather than a prediction task, as the human annotations summarize a developer's expertise area, thus the BLEU score is a justified metric. In this context, the candidate summary is the model's set of prediction words, and the reference summary is the human annotation. When using *BLEU scores*, the uni-gram matches only are taken into consideration. The similarity between the model's set of prediction words and the human annotation's set of words can be formulated into a word overlap formula, such as the one *Jaccard similarity* has, thus this metric is also justified. In the evaluation of the above three techniques, *cosine similarity* is used to compute a multi-word comparison between the model's prediction words and the human annotation's set of words, by using a pre-trained Word2Vec model [18] as a word vector representation look-up.

Subjectively, the most suited metric to the task of *Expertise Extraction* should be *cosine similarity*, as it measures the semantic similarity between the two sets of words (annotation and model hypothesis), as opposed to counting how many words out the two sets of words are the same. *Cosine similarity* as a metric would make the evaluation more robust, as opposed to making it a binary frequency count. All three metrics are reported to give a better sense of accuracy, but the *cosine similarity* score is our number one choice as the evaluation metric.

In order to compare each technique to a baseline, a *random model* was also created. This random model takes the text corpus' dictionary and calculates the frequency of each unique word in the corpus, then it normalizes them to create a frequency-based discrete probability distribution. The random model samples n words from the dictionary based on the probability distribution described above. The n sampled words become the extracted expertise words, thus defining a random baseline for the techniques. The goal is to outperform the random baseline as much as possible.

Finally, as mentioned at the beginning of the subsection, comparing expertise trends on Stack Overflow and GitHub is still needed. After the best performing technique was found for both Stack Overflow and GitHub data sets, the best performing LDA model is extracted from the technique. The topics of these two LDA models fitted on Stack Overflow and GitHub are manually labelled based on the topic labeling methodology of Bangash et al. [28] and Hindle et al. [30]. The observations, trends and patterns noticed when comparing the topics of the two best performing models become the insight about expertise trends on Stack Overflow and GitHub.

As mentioned at the beginning of Section 3.7, the remainder of the sections are focused on better understanding the modeling of expertise. The goal of these sections was to perform analysis on LDA topics about expertise data and gain some insights into how cross-platform expertise is being extracted from Stack Overflow and GitHub.

3.7.2 Cross-platform Expertise

This section is focused on comparing the expertise gained by the same users on Stack Overflow and GitHub. The goal is to explore the similarity between expertise profiles on Stack Overflow and GitHub, by performing analysis on four separate text corpora (GH-past, GH-recent, SO-past and SO-recent) which are explained in Section 3.3.3. The set-up of the analysis is structured in a way to directly compare the expertise of the same user on GitHub and Stack Overflow, even at the same time. Two separate comparisons are considered: the text corpus of GH-past is compared with SO-past, and GH-recent is compared with SO-recent. The comparison takes shape by extracting key expertise terms of the same user from both Stack Overflow and GitHub user profiles and comparing how much overlap there is in expertise terms of the same user on two different platforms. The full details of the algorithm used to extract expertise terms to be compared can be found in Algorithm 9.

The *Extraction of Expertise Terms* algorithm takes in three parameters: the first and second text corpus, and the LDA model fitted on the first text corpus. In line 1 of Algorithm 9, the first topic document matrix ($TD1$) containing the topic distribution of each document in $Corpus\_1$ is obtained from the fitted LDA_Model. In line 2, the second topic document matrix ($TD2$) is obtained by performing inference on $Corpus\_2$'s documents using the fitted LDA_Model. This is a crucial step of the algorithm, as two topic distributions coming from two separate models can not be compared in the same analysis. Using Bayesian inference on the second corpus allows both corpora to have a common topic distribution coming from a single fitted LDA_Model. In line 7, an iteration of all users within the data set is performed. In lines 8–13, a list of all relevant expertise terms for $Corpus\_1$'s topic distribution is constructed. In line 9, $TD1[\ user\_ID\ ]$ represents the topic distribution of $user\_ID$'s

Algorithm 9 Extraction of Expertise Terms.

```
Require: $Corpus\_1$: First Text Corpus, $Corpus\_2$: Second Text Corpus,
    $LDA\_Model$: Topic Model Fitted on the First Corpus
Get common topic distribution $TD1$ for $Corpus\_1$ from the fitted $LDA\_Model$
Get common topic distribution $TD2$ for $Corpus\_2$ by performing inference using
    the fitted topic model, $LDA\_Model$
Expertise-Terms_1 = { }
Expertise-Terms_2 = { }

// For each user get expertise terms from $TD1$ and $TD2$
for all users $user\_ID$ in the data set do
    listOf_topicWords = [ ]
    for all topics $t$ in $TD1$[ $user\_ID$ ] with probability $\geq 0.1$ do
        $TW$ = Get top-20 Topic Words representing topic $t$
        Add Topic Words $TW$ to listOf_topicWords
    end for
    Expertise-Terms_1 [ $user\_ID$ ] = listOf_topicWords

    listOf_topicWords = [ ]
    for all topics $t$ in $TD2$[ $user\_ID$ ] with probability $\geq 0.1$ do
        $TW$ = Get top-20 Topic Words representing topic $t$
        Add Topic Words $TW$ to listOf_topicWords
    end for
    Expertise-Terms_2 [ $user\_ID$ ] = listOf_topicWords

end for
return Expertise-Terms_1, Expertise-Terms_2
```

user profile found in $Corpus\_1$. In the same line, the iteration over all topics t in $TD1[\ user\_ID\ ]$ with probability greater than 0.1 represents the extraction of relevant topics found in $user\_ID$'s topic distribution. In this context, relevant topics were defined as topics that describe the expertise of a user with at least a probability value of 0.1 (or 10 %). In line 10–11, the top-20 topic words of $TD1[\ user\_ID\ ]$'s relevant topics are extracted and stored in a list of topic words, which become the expertise terms extracted from $Corpus\_1$'s topic distribution. In lines 15-20, the same process is applied to $Corpus\_2$'s documents. In line 16, $TD2[\ user\_ID\ ]$ represents the topic distribution of $user\_ID$'s user profile found in $Corpus\_2$. In line 17–18, the top-20 topic words of $TD2[\ user\_ID\ ]$'s relevant topics are extracted and stored in a list of topic words, which become the expertise terms extracted from $Corpus\_2$'s topic distribution. At the end of the algorithm, the expertise terms extracted from $Corpus\_1$ and $Corpus\_2$ are returned.

The *Extraction of Expertise Terms* algorithm is executed twice. In the first run, the parameters were the GH-past and SO-past text corpora, and the GH-past LDA model. For the second run, the GH-recent and SO-recent text corpora, and the GH-recent LDA model were passed as parameters. During both runs, the LDA model is chosen based on whichever corpus has more data to train a larger LDA model. For both past and recent data segmentation, the GitHub corpus is multiple times larger than the Stack Overflow one. Performing analysis on GH-past and SO-past text corpora, then on the GH-recent and SO-recent text corpora allows the direct expertise comparison of the same user on GitHub and Stack Overflow, at the same time. Once the expertise terms have been extracted, an iteration over all users is performed, and the *Jaccard similarity* of the two sets of expertise terms is calculated. The *Jaccard similarity* results are aggregated and interpreted using histograms, density plots and bar plots to draw insights about the extent that developers build similar cross-platform expertise.

3.7.3 Cross-platform Transferable Knowledge

This section is focused on discovering what knowledge is transferable from one platform to another. To answer this question the same analysis set-up as in Section 3.7.2 is considered, as in the task of *Cross-platform Expertise* the expertise of the same user is directly compared on GitHub and Stack Overflow. Performing the same experiment as in the previous task, but rather analyzing the *Top-n* most frequent common expertise terms between the two platforms would shed some light on transferable knowledge.

For the *Cross-platform Transferable Knowledge* task the *Extraction of Expertise Terms* algorithm is executed twice, with the {GH-past, SO-past}, and {GH-recent, SO-recent} text corpora pairs. The frequencies of common expertise terms in the text corpora pairs are calculated and the *Top-n* common expertise terms, alongside their frequency counts are returned. *n* could take any value, but a larger value such as 30, 50 or 100 would be considered appropriate. Any trends or patterns between the *Top-n* common expertise terms are noted, then a grouping of expertise terms into categories of transferable knowledge is performed. The patterns, trends, and groups of expertise terms become the insight gained into *Cross-platform Transferable Knowledge.*

3.7.4 Expertise Evolution

This section is focused on comparing the evolution of expertise of the same users on Stack Overflow and GitHub. The goal is to explore how developer expertise changes over time. Furthermore, we explore how the evolution of expertise different on Stack Overflow and GitHub. To answer these questions, analysis is performed on four separate text corpora (GH-past, GH-recent, SO-past and SO-recent) which are explained in Section 3.3.3. The set-up of the analysis is structured in a way to directly compare the expertise of the same user in the past and recent times, individually on

GitHub, then on Stack Overflow. Two separate comparisons are considered: the text corpus of GH-past is compared with GH-recent, and SO-past is compared with SO-recent. The comparison is very similar to the one performed in Section 3.7.2, for *Cross-platform Expertise*: expertise terms of the same user are extracted from both recent and past text corpora from both Stack Overflow and GitHub data sets. Instead of comparing how much overlap there is in expertise terms of the same user (like Section 3.7.2), the opposite of overlap, difference or change in expertise terms of the same user is considered for the task of *Expertise Evolution.* Since the task is concerned with expertise change over time, it seemed more appropriate to choose a metric that is related to the dissimilarity of sets, which is *Jaccard distance.*

Algorithm 9, the *Extraction of Expertise Terms* was executed twice for the task of *Expertise Evolution* too. In the first run, the parameters were the GH-past and GH-recent text corpora, and the GH-past LDA model. For the second run, the SO-past and SO-recent text corpora, and the SO-past LDA model were passed as parameters. During both runs, the LDA model is chosen based on whichever corpus has more data to train a larger LDA model. For both GitHub and Stack Overflow data sets the past corpus segmentation is larger than the recent one. Performing analysis on GH-past and GH-recent text corpora, then on the SO-past and SO-recent text corpora allows the analysis of expertise evolution of the same user on GitHub, then on Stack Overflow independently. Once the expertise terms have been extracted, an iteration over all users is performed, and the *Jaccard distance* of the two sets of expertise terms is calculated. The results are aggregated and interpreted using histograms, density plots and bar plots to draw insights about the extent that developers' expertise changes over time.

3.8 Software Tools Used

Every algorithm described in Section 3.7 was implemented in Python 3.6[14]. The entirety of the data cleaning and processing stages, alongside the topic modeling was also done in Python. All implementations of common models and data science techniques come from open-source Python libraries such as *gensim*[15], *pyLDAvis*[16], *numpy*[17], *pandas*[18], *nltk*[19], *scikit-learn*[20], *scikit-optimize*[21] and *sqlalchemy*[22]. Occasionally Jupyter Notebooks[23] were used on Google's Colaboratory [85] platform[24]. For data storage a MySQL database[25] was chosen, and SQL was used to handle the data. All data plots were created using the R [86] statistical language's *ggplot2* [87] library[26].

[14] https://www.python.org/downloads/release/python-360/

[15] for Topic Modeling - https://radimrehurek.com/gensim/index.html

[16] Topic Modeling Visualization https://pyldavis.readthedocs.io/en/latest/readme.html

[17] for Numpy Arrays - https://numpy.org/

[18] for Data Structures - https://pandas.pydata.org/

[19] for Textual Data - https://www.nltk.org/

[20] for Machine Learning - https://scikit-learn.org/stable/index.html

[21] for Optimization - https://scikit-optimize.github.io/

[22] for Database Connection - https://www.sqlalchemy.org/

[23] https://jupyter.org

[24] Google Colab https://colab.research.google.com

[25] https://www.mysql.com/

[26] https://ggplot2.tidyverse.org/

Chapter 4

Results

In this chapter, we present the results of experiments conducted and provide answers to all research questions from Section 1.3. Section 4.1 takes up the majority of this chapter and addresses results achieved on the task of *Expertise Extraction*. Analysis results of *Cross-Platform Expertise* are presented in Section 4.2, while *Cross-Platform Transferable Knowledge* results are reported in Section 4.3. Section 4.4 presents our results and conclusions for *Expertise Evolution*.

4.1 Expertise Extraction

For the Expertise Extraction task *three* novel techniques have been proposed: Baseline LDA Based Expertise Extraction *(Technique 1)*, Improved LDA Based Expertise Extraction *(Technique 2)*, and Pre-trained Word2Vec Based Expertise Extraction *(Technique 3)*. Technique 2 and Technique 3 have two different variations each: the first one is using *Average-pooling*, while the second one is using the *Max-pooling* technique for down-sampling. Including these two variations, a total of five models are considered:

- Model 1: Baseline LDA Based Expertise Extraction, labelled as *B_LDA*,

- Model 2: Improved LDA Based Expertise Extraction obtained by Average-pooling, labelled in tables as *LDA_AVG*,

- Model 3: Improved LDA Based Expertise Extraction obtained by Max-pooling, labelled in tables as *LDA_MAX*,

- Model 4: Pre-trained Word2Vec Based Expertise Extraction obtained by Average-pooling, labelled in tables as *W2V_AVG*,

- Model 5: Pre-trained Word2Vec Based Expertise Extraction obtained by Max-pooling, labelled in tables as *W2V_AVG*.

The above listed five models are evaluated against a baseline (naive) model (see Section 3.7.1.4), which is a frequency-based random model. When evaluating the extraction of expertise areas, two similar experiments were used. In the first experiment, each model's performance is evaluated against the ground truth knowledge from Section 3.6 with the specification that a model has to output exactly as many expertise terms as the ground truth annotation contains. This case is referred to as Experiment 1. In the second experiment, each model is evaluated against the ground truth annotation, but there are no restrictions applied to the number of expertise terms to be returned by each model. In this scenario, the models can be more robust, as they can return all the expertise terms found by the underlying algorithm, without needing to rank and return only the top-n expertise terms. Throughout this chapter, this more robust, less restrictive experiment setup will be referred to as Experiment 2. Furthermore, each one of the above mentioned experiments has to variations: expertise extraction from 1) GitHub data (labelled Experiment A) and 2) Stack Overflow data (labelled Experiment B).

Table 4: Results of Experiment 1A.

Top 3 Results	Model Metrics	**B_LDA**	**LDA_AVG**	**LDA_MAX**	**W2V_AVG**	**W2V_MAX**	**Baseline**
1	Cosine Sim.	0.6690	0.7187	0.7357	**0.7998**	0.7317	0.5962
	Jaccard Sim.	0.0658	0.0751	0.1040	0.0765	0.1049	0.0286
	BLEU Score	0.1197	0.1340	0.1767	0.1368	0.1782	0.0540
2	Cosine Sim.	0.6689	0.7183	0.7351	0.7959	0.7316	0.5962
	Jaccard Sim.	0.0658	0.0750	0.1037	0.0818	0.1049	0.0286
	BLEU Score	0.1197	0.1338	0.1762	0.1452	0.1782	0.0540
3	Cosine Sim.	0.6683	0.7183	0.7351	0.7959	0.7316	0.5962
	Jaccard Sim.	0.0652	0.0750	0.1037	0.0818	0.1049	0.0286
	BLEU Score	0.1186	0.1338	0.1762	0.1452	0.1782	0.0540

4.1.1 Experiment 1

Table 4 shows the top-3 highest evaluation metric scores for each one of the five models from *Experiment 1A* - Expertise Extraction from GitHub Data. The top-3 ranking is done based on the highest cosine similarity scores. The five models are compared against a frequency-based random model (referred to as *Baseline*), whose evaluation metrics comes from averaging the results of 10,000 random samples.

The results from Table 4 indicate that each model's top-3 highest evaluation scores outperform the baseline model's each one of the three metrics. *W2V_AVG*, which is named *Expertise Extraction using Pre-trained Word2Vec based User and Topic Embeddings obtained by Average-pooling* achieves the highest cosine similarity score (*0.7998*). The baseline cosine similarity score of the random model is *0.5962*. Jaccard similarity and BLEU score are also reported, but cosine similarity is considered the primary evaluation metric when comparing different techniques' performance, due to the reasons explained in Section 3.7.1.4. *B_LDA* is performing the worst, having its best cosine similarity score of *0.6690*, and that is expected, since it is a topic probability distribution based technique, and does not use user and topic embeddings.

Table 5: Example of cosine similarity scores between term-pairs.

Term 1	Term 2	Cosine Similarity	Term 1	Term 2	Cosine Similarity
php	python	0.2529	html	javascript	0.6024
java	python	0.3929	ajax	jquery	0.6315
analysis	visualization	0.4524	sklearn	tensorflow	0.6858
nodejs	reactjs	0.4909	bagging	random-forest	0.7251
java	jdk	0.5517	mysql	postgresql	0.7997
xml	json	0.5866	keras	tensorflow	0.8391

LDA_MAX and *W2V_MAX* are performing very similarly, both plateauing around a cosine similarity score of *0.73*. *LDA_AVG* performs a little worse than *LDA_MAX* and *W2V_MAX* by having the maximum cosine similarity score of *0.7187*.

It is difficult to understand how accurate these techniques are only based on cosine similarity scores obtained from calculating the semantic similarity between the bag of words of human annotations and model extraction. To illustrate different levels of semantic similarity of cosine similarity scores Table 5 contains examples of term-pairs and their respective cosine similarity scores. These cosine similarity scores were obtained using the same pre-trained Word2Vec model [18] used in Section 3.7.1.3.

Based on the sample cosine similarity scores from Table 5 one can say that *W2V_AVG*'s best model extracts expertise terms with similar semantic similarity level to the contextual similarity between the 'mysql' and 'postgresql' term-pair. Likewise, *LDA_AVG*, *LDA_MAX* and *W2V_MAX* extract expertise terms with similar semantic similarity level to 'random-forest' and 'bagging' term-pair, while *B_LDA*'s best cosine similarity score classifies its semantic similarity level just below 'tensorflow' and 'sklearn' term-pair's. One could argue that the above mentioned semantic similarity levels correspond to contextually similar software engineering related word associations, thus illustrating that Table 4's results lead to high levels of semantic similarity between the bags of words of human annotations and expertise terms extracted by the models.

Unfortunately, as far as we are aware, the same kind of representation of similarity levels can not be easily created for *Jaccard Similarities* and *BLEU Scores*. Jaccard similarity is a measure of set similarity, and it highly depends not only on the magnitude of the set intersection, but the set union as well. BLEU score values range between 0 and 1, and the inventors of BLEU score report that in their study a professional human translator was able to achieve a score of *0.3468* calculated against four reference annotations and *0.2571* against two reference annotations [44].

Table 19 found in Appendix A.3.1 contains all major parameters used to fit the models outlined in Table 4. The models were fitted using a hyper-parameter optimization process described in Section 3.5.2 and their parameters come from the parameter search space defined in Section 3.5.2. Table 19 suggests a general pattern of the GitHub data containing a relatively small number of topics: 10 in *LDA_AVG*, 11 being the most popular in *B_LDA*, *LDA_MAX* & *W2V_MAX*, and 16 in the superior *W2V_AVG*. The parameter β highly varies between 0.001, 0.005, 0.01 and 0.05, but in most cases $\beta = 0.005$.

Table 6 offers *good, very good and bad* examples of actual expertise terms extracted by our models during Experiment *1A*, compared with the ground truth human annotations. The expertise terms that are common across human annotation and model extraction are shown in **bold**. Alongside the expertise terms, the number of terms matched and the cosine similarity between the two bags of words are also reported in Table 6. This type of expertise terms comparison demonstrates that Experiment *1A*'s results lead to high levels of semantic similarity between the bags of words of human annotations and expertise terms extracted by the models.

Table 7 shows the top-3 highest evaluation metric scores for each one of the five models from *Experiment 1B* - Expertise Extraction from Stack Overflow Data. Same as before, the top-3 models are ranked based on the highest cosine similarity scores,

Table 6: Sample annotation and model extraction comparison.

Type of Example	Human Annotation	Model Extraction	Match	Cosine Sim.
Bad	[ui, linux, images, books, perl, c, data_process, proxy, blogs, python, go, shell, c#, javascript, web, maintaining, creating, java, css, html, dockerfile, image_process, software, libraries, docker, ruby, tex, c++, groovy, writing, **development**, makefile, agile]	[php, wordpress, laravel, bundle, symfony, drupal, wp, framework, plugin, magento, composer, doctrine, symfony2, api, extension, redmine, simple, phpunit, library, cms, module, zend, p, form, admin, standards, integration, skeleton, silex, documentation, guzzle, custom, **development**]	1/33	0.5361
Good	[tls, programming, oss_fuzz, c, bcyrpt, go, security, ssl, **rust**, **java**, homebrew, urllib3, **python**, software_engineering, objective_c, **ruby**, azure, powershell, **shell**, infra, propery_base_testing, cryptography, networking, makefile, html, open_ssl, cpython, algorithms, **clojure**, vorbis]	[**python**, **ruby**, php, scala, emacs, **clojure**, javascript, haskell, emacs_lisp, c, **rust**, **java**, perl, spring, c++, rails, docker, mode, **shell**, r, ansible, awesome, elm, react, django, grails, elixir]	6/30	0.7749
Very Good	[objective_c, type, go, css, typescript, **dockerfile**, **java**, **scala**, **python**, **ruby**, **javascript**, **shell**, **php**, **haskell**, script, swift, **c++**, smarty, html, **django**, **clojure**, vim_script]	[**python**, **ruby**, **php**, **scala**, **clojure**, **javascript**, **haskell**, c, rust, **java**, perl, spring, **c++**, rails, **docker**, **shell**, r, ansible, awesome, elm, react, **django**]	12/22	0.9138

Table 7: Results of Experiment 1B.

Top 3 Results	Model Metrics	**B_LDA**	**LDA_AVG**	**LDA_MAX**	**W2V_AVG**	**W2V_MAX**	**Baseline**
1	Cosine Sim.	0.5044	0.5582	**0.5837**	0.5607	0.5820	0.3721
	Jaccard Sim.	0.0160	0.0406	0.0435	0.0320	0.0556	0.0104
	BLEU Score	0.0313	0.0770	0.0823	0.0612	0.1041	0.0199
2	Cosine Sim.	0.4997	0.5560	0.5717	0.5574	0.5746	0.3721
	Jaccard Sim.	0.0295	0.0747	0.0814	0.0404	0.0784	0.0104
	BLEU Score	0.0560	0.1366	0.1471	0.0768	0.1424	0.0199
3	Cosine Sim.	0.4755	0.5409	0.5676	0.5537	0.5736	0.3721
	Jaccard Sim.	0.0127	0.0754	0.0821	0.0718	0.0305	0.0104
	BLEU Score	0.0247	0.1366	0.1478	0.1312	0.0584	0.0199

and the models are compared against a *Baseline*.

The results from Table 7 show that each model's top-3 highest evaluation scores outperform the baseline model on each one of the three metrics. *LDA_MAX*, which is named Expertise Extraction using LDA based User and Topic Embeddings obtained by Max-pooling achieves the highest cosine similarity score (*0.5837*), thus it can be considered the superior technique on the Stack Overflow data set. The baseline cosine similarity score of the random model is *0.3721*, which is much lower than all other model's cosine similarity scores. *B_LDA* is performing the worst, having its best cosine similarity score of *0.5044*. *LDA_AVG* and *W2V_AVG* are performing very similarly, both plateauing close to a cosine similarity score of *0.56*. *W2V_MAX* is a very close second by having a maximum cosine similarity score of *0.5820*, and barely being outperformed by the superior *LDA_MAX*.

The results of Table 7 show a trend of *LDA_MAX* and *W2V_MAX* outperforming *LDA_AVG* and *W2V_AVG*. This means that performing down-sampling using Max-pooling, instead of Average-pooling seems to work better on the Stack Overflow data set, which was not the case in Experiment 1A conducted on the GitHub data set.

Sample cosine similarity scores from Table 5 can help associate semantic similarity levels to the results presented in Table 7. Based on Table 5 one can say that *LDA_MAX* and *W2V_MAX*'s best models extract expertise terms with similar semantic similarity level to the contextual similarity between the 'xml' and 'json' term-pair. Likewise, *LDA_AVG* and *W2V_AVG* extract expertise terms with similar semantic similarity level to 'java' and 'jdk' term-pair, while *B_LDA*'s best cosine similarity score classifies its semantic similarity level just above 'nodejs' and 'reactjs' term-pair's. The above mentioned term-pairs correspond to contextually similar software engineering related word associations with similar semantic similarity levels to the model performances in Table 7. Thus, one could argue that Table 7's results lead to high levels of semantic similarity between the bags of words of human annotations and expertise terms extracted by the models.

Table 20 found in Appendix A.3.1 contains all major parameters used to fit the models outlined in Table 7. Table 20 suggests a general pattern of the Stack Overflow data containing a relatively large number of topics: {40,29,45} in *B_LDA*, and *LDA_AVG* and *LDA_MAX* have 33 & 27 number of topics in common, while *W2V_AVG* and *W2V_MAX* have 48 & 31 number of topics in common. The most common number of topics out of the best performing models is 27. The parameter β highly varies between 0.01, 0.1, 0.5 and 1, but in most cases $\beta = 1$.

4.1.2 Experiment 2

The results of Experiment *2A* and *2B* can be seen in Table 8 and Table 9. Note that Experiment *2A* and *2B* have no restrictions applied to the number of expertise terms returned by each model, thus the model will return as many expertise terms as it can find. This stipulation will significantly disadvantage the performance of Jaccard similarity and BLEU score (since the ratios change), but theoretically, it could increase the cosine similarity scores.

Table 8: Results of Experiment 2A.

Top 3 Results	Metrics \ Model	**B_LDA**	**LDA_AVG**	**LDA_MAX**	**W2V_AVG**	**W2V_MAX**	**Baseline**
1	Cosine Sim.	0.6828	0.7377	0.7602	**0.7761**	0.7680	0.5962
	Jaccard Sim.	0.0246	0.0218	0.0251	0.0209	0.0144	0.0286
	BLEU Score	0.0277	0.0245	0.0279	0.0212	0.0127	0.0540
2	Cosine Sim.	0.6827	0.7364	0.7598	0.7750	0.7672	0.5962
	Jaccard Sim.	0.0249	0.0256	0.0254	0.0204	0.0138	0.0286
	BLEU Score	0.0281	0.0292	0.0288	0.0208	0.0122	0.0540
3	Cosine Sim.	0.6821	0.7362	0.7595	0.7748	0.7671	0.5962
	Jaccard Sim.	0.0241	0.0244	0.0259	0.0218	0.0131	0.0286
	BLEU Score	0.0271	0.0278	0.0295	0.0222	0.0116	0.0540

Experiment *2A*'s results from Table 8 show that each model's top-3 highest cosine similarity scores outperform the baseline model's score. Unfortunately, that is not the case with Jaccard similarity and BLEU score, as these two metrics have decreased, even below the baseline model's performance. Oddly, this makes sense, as both of these metrics are ratios, which get affected by having no restrictions applied to the number of expertise terms returned by each model. For example, Jaccard similarity is calculated by dividing the cardinality of the set intersection with the cardinality of the set union. Having applied no restrictions to the number of expertise terms returned by the model, the set union increases drastically, and as a consequence Jaccard similarity scores have decreased. Same as for Experiment 1, when comparing different techniques' performance cosine similarity is considered the primary evaluation metric, due to the reasons explained in Section 3.7.1.4.

Interpreting the results from Table 8 one can see that *W2V_AVG*, i.e., Expertise Extraction using Pre-trained Word2Vec based User and Topic Embeddings obtained by Average-pooling, achieves the highest cosine similarity score (*0.7761*), thus it can be considered the superior technique on the GitHub data set, as this was the case in Experiment *1A* as well. The baseline cosine similarity score of the random model is *0.5962*. *B_LDA* is the worst performing model, but it improved its best cosine similarity score from *0.6690* on Experiment *1A* to *0.6828* on Experiment *2A*. This model having a low performance is no surprise, since *B_LDA* is a topic probability distribution based model which does not use user and topic embeddings. *LDA_MAX* and *W2V_MAX* are performing very similarly, both plateauing around a cosine similarity score of *0.76*, which is higher than the score of *0.73* achieved by both models in Experiment *1A*. *LDA_AVG* performs a little worse than *LDA_MAX* and *W2V_MAX* by having a maximum cosine similarity score of *0.7377*, which is still an improvement from *0.7187* in Experiment *1A*.

Based on the examples of semantic similarity levels from Table 5 one can say that *W2V_AVG*'s best model extracts expertise terms very close to the semantic similarity level between 'mysql' and 'postgresql' term-pair. In addition, *LDA_AVG* extract expertise terms with similar semantic similarity level to 'random-forest' and 'bagging'. Furthermore, *B_LDA*'s best cosine similarity score from Experiment *2A* classifies its semantic similarity level similar to the term-pair of 'tensorflow' and 'sklearn'. These semantic similarity levels correspond to contextually similar software engineering related word associations, which strongly suggest that Table 8's results lead to high levels of semantic similarity between the bags of words of human annotations and expertise terms extracted by the models.

Table 21 found in Appendix A.3.1 contains all major parameters used in the hyper-parameter optimization when fitting the models outlined in Table 8. The best models suggest a general pattern of the GitHub data containing a relatively small number of topics: 11 in *B_LDA*, 16 being the most popular in *LDA_MAX*, *W2V_AVG* & *W2V_MAX*, while a surprising 21 in *LDA_AVG*. The parameter β highly varies between 0.001, 0.005, 0.01 and 0.05, but in most cases it is either $\beta = 0.001$ or $\beta = 0.005$.

Experiment *2B*'s results from Table 9 show that each model's top-3 highest cosine

Table 9: Results of Experiment 2B.

Top 3 Results	Model Metrics	**B_LDA**	**LDA_AVG**	**LDA_MAX**	**W2V_AVG**	**W2V_MAX**	**Baseline**
1	Cosine Sim.	0.5263	0.5361	0.5837	**0.5902**	0.5117	0.3721
	Jaccard Sim.	0.0099	0.0230	0.0435	0.0295	0.0102	0.0104
	BLEU Score	0.0117	0.0231	0.0822	0.0363	0.0087	0.0199
2	Cosine Sim.	0.5080	0.5304	0.5717	0.5821	0.5087	0.3721
	Jaccard Sim.	0.0223	0.0175	0.0816	0.0196	0.0120	0.0104
	BLEU Score	0.0267	0.0196	0.1471	0.0222	0.0101	0.0199
3	Cosine Sim.	0.5036	0.5266	0.5697	0.5698	0.5086	0.3721
	Jaccard Sim.	0.0111	0.0149	0.0303	0.0180	0.0106	0.0104
	BLEU Score	0.0136	0.0169	0.0579	0.0180	0.0090	0.0199

similarity scores outperform the baseline model's score. Unfortunately, this can not be said about Jaccard Similarity and BLEU Score as these two metrics have decreased even below the baseline model's performance. *W2V_AVG*, which is named *Expertise Extraction using Pre-trained Word2Vec based User and Topic Embeddings obtained by Average-pooling* achieves the highest cosine similarity score (*0.5902*), slightly outperforming the best score (*0.5837*) from Experiment *1B*. The baseline cosine similarity score of the random model is *0.3721*, thus *W2V_AVG* outperforms the baseline by over 0.20, which is a large difference. Interestingly, for the first time *W2V_MAX* has the worst performance, obtaining a maximum cosine similarity score of *0.5117*, which is much lower than its previous performance of *0.5820* in Experiment *1B*. *LDA_AVG* and *B_LDA* are performing very similarly, both plateauing between cosine similarity scores of *0.52* and *0.54*. *LDA_MAX* is a very close second by having a maximum cosine similarity score of *0.5837*, and barely being outperformed by the superior *W2V_AVG*.

Sample cosine similarity scores from Table 5 help associate semantic similarity levels to the results presented in Table 9. Based on Table 5 one can say that *W2V_AVG* and *LDA_MAX*'s best models extract expertise terms with similar semantic similarity level to the contextual similarity between the 'xml' and 'json' term-pair. Likewise, *LDA_AVG* extracts expertise terms close to the similar semantic similarity level of 'java' and 'jdk'. Furthermore, *W2V_MAX*'s best cosine similarity score from Experiment *2B* classifies its semantic similarity level similar to the term-pair of 'nodejs' and 'reactjs'. The above mentioned term-pairs correspond to contextually similar software engineering related word associations, thus one could argue that Table 9's results lead to acceptable semantic similarity levels between the bags of words of human annotations and expertise terms extracted by the models.

Table 22 found in Appendix A.3.1 contains all major parameters used to fit the models outlined in Table 9. Table 22 suggests a general pattern of the Stack Overflow data containing a relatively large and versatile number of topics: 40 topics in *B_LDA*'s best model, 13 topics in *LDA_AVG*'s, while 33 topics in *LDA_MAX*'s best model. *W2V_AVG*'s model was fitted with 14 topics, and lastly *W2V_MAX* had 30 number of topics. The parameter β highly varies between 0.001, 0.005, 0.01, 0.1, 0.5 and 1, but in most cases $\beta = 0.1$.

Overall, the results of Table 9 show both improvements and declines in performance compared to Experiment *1B* conducted on the same Stack Overflow data set. Interestingly, if we compare Experiments *1A* and *2A*, we could notice mostly improvements in performance, but the same can not be said about Experiments *1B* and *2B*, which have mixed performance differentials.

4.1.3 GitHub and Stack Overflow Topic Patterns

The best performing LDA models from Experiment *1A* and *1B* have been extracted, and its topics have been manually labelled based on the methodology outlined in Section 2.1.4.6. For Experiment *1A* an LDA model with 16 topics using the *W2V_AVG* algorithm had the highest cosine similarity score, while an LDA model with 33 topics paired with the *LDA_MAX* algorithm performed the best for *Experiment 1B*. Table 10

Table 10: Labels given to topics of the best LDA model trained on GitHub.

Topic	Given Name	Topic	Given Name
1	Discarded/ Omitted	9	Mobile App Development
2	Web Development	10	Research
3	Cloud Computing	11	Unix
4	Front-end Web Development	12	Rust
5	C/C++	13	PHP
6	Web Layout	14	Big Data
7	Ruby	15	JVM
8	Data Science	16	Emacs

reveals the labelled topics of the LDA model trained on GitHub data for Experiment *1A*, while Table 11 exposes the hidden patterns in Stack Overflow expertise terms using 33 labelled topics for Experiment *1B*.

Table 10 paints a clear picture of what are the major trends of topics in GitHub's expertise terms. Topic #1 contained quite general keywords as topic words, thus it was discarded. Topics #2 to #4, and #6 are heavy in web development related technologies, while topics #5, #7, #12 and # 13 are very specific to a programming language like C/C++, Ruby, Rust or PHP. Furthermore, other topics include very specific fields within or related to software development, like *data science, mobile application development, research* and *big data*. Other loosely connected topics present in the data are *Unix, JVM* and *Emacs*. The impressiveness of these results is shown in topics mapping to a specific programming language or field of Computer Science, which makes this LDA model useful and interpretable for practitioners.

Table 11 sheds a light on the major trends of topics in Stack Overflow expertise terms. The obvious difference between the topics of Stack Overflow and GitHub is that the former contains many more topics. Out of the 33 topics, 31 of them map to recognizable entities or artifacts in software development, leaving only two

Table 11: Labels given to topics of the best LDA model trained on Stack Overflow.

Topic	Given Name	Topic	Given Name	Topic	Given Name
1	File Management	12	C/C++	23	UI
2	Data Visualization	13	Version Control	24	Android
3	JavaScript Programming	14	Data Management	25	Web Graphics
4	Front-end Web development	15	Database	26	VIM
5	Python	16	Java Build Tools	27	Distance Calculation
6	Algorithms	17	Unix	28	Discarded Topic
7	HTTP	18	Ruby	29	PHP
8	OOP	19	Angular	30	Data Searching
9	Server-client	20	Web Layout	31	SSL + Game Development
10	Java	21	iOS Development	32	Encryption
11	Data Types	22	Parallel Programming	33	Discarded/ Omitted

topics being discarded due to containing too general terms. Topics #3, #4, #7, #9, #20 and #25 lead to web development related software artifacts, while topics #5, #10, #12, #18 and #29 map to programming languages such as Python, Java, C/C++, Ruby and PHP. Much more specific areas of Computer Science, such as *data management, version control, algorithms, data types, data visualization, databases, iOS development, parallel programming, Android development*, and *encryption* could also be found as topics within the LDA model's list of topics. Other topics loosely connected to software development include *distance calculation, file management*, and *data searching.*

Table 12 contains examples of ambiguous and unambiguous collections of topic words that were labeled in Tables 10 and 11. The GitHub data set's unambiguous topic example comes from Topic #7, labeled *Ruby* by both topic annotators. The ambiguous example is Topic #10 labeled *'Real-time app development'* by the first annotator, and labeled *'Research'* by the second annotator. A third opinion ruled the label to be *'Research'*. The overall percent agreement between the two annotators was 62.50% for the topics generated by the best performing LDA model for the GitHub data set. The Stack Overflow data set's unambiguous topic example comes from Topic

Table 12: Examples of ambiguous and unambiguous topic words generated by the best performing LDA models for the GitHub and Stack Overflow data sets.

Data Set	Unambiguous Topic-Words	Ambiguous Topic-Words
GitHub	[ruby, rails, elixir, api, gem, heroku, app, simple, client, library, buildpack, test, javascript, css, phoenix, wrapper, activerecord, application, rspec]	[data, erlang, test, repo, html, code, tex, julia, git, makefile, package, analysis, R, repository, source, tools, issue, environment, github]
Stack Overflow	[class, object, method, use, instance, methods, objects, classes, constructor, base, inheritance, create, defined, declared, type, model, interface, code]	[matrix, ssl, certificate, pos, openssl, opengl, 3d, camera, unity, texture, scene, depth, polygon, webgl, buffer, nominal, projection, framebuffer]

#8, labeled *'OOP'* by both annotators. The ambiguous example is Topic #31 labeled *'SSL + Game Development'* by the first annotator, and labeled *'Animation/Game Development'* by the second annotator. A third opinion ruled the label to be *'SSL + Game Development'*. The overall percent agreement between the two annotators was 48.48% for the topics generated by the best performing LDA model for the Stack Overflow data set. One can notice that the topics generated by the LDA model for the Stack Overflow data set tend to be more specific, as even the agreement between the two annotators is lower than for the topics generated for the GitHub data set.

There are numerous similarities and differences between the discovered topics outlined in Tables 10 and 11. Both GitHub and Stack Overflow expertise areas include a few popular programming language related topics, and both seem to be dominated by web development related skills, technologies, platforms, frameworks or software artifacts. The difference between the two collaborative platform's expertise areas is that GitHub expertise areas tend to be few, and more general, while Stack Overflow expertise areas tend to be more specific and numerous.

Answer to RQ1: From the three novel techniques evaluated we found that *W2V_AVG* performs best for *Experiment 1A*, *LDA_MAX* slightly outperforms *W2V_MAX* to be the best for *Experiment 1B*, while *W2V_AVG* performs best for both *Experiment 2A* and *Experiment 2B*.

4.2 Cross-platform Expertise

The *Cross-platform Expertise Analysis* is concerned with comparing the expertise gained by users in two separate collaborative platforms, Stack Overflow and GitHub. This analysis should answer the research question about the similarity of developer expertise profiles in Stack Overflow and GitHub. In this analysis the similarity of four separate text corpora (GH-past, GH-recent, SO-past, SO-recent; see Section 3.3.3) are compared. For the experiment set-up behind this analysis see Section 3.7.2.

Figure 9 and 10 show the results of comparing the most recent activity of users on GitHub and Stack Overflow. For the rest of the analysis this will be referred to as the *GH-recent – SO-recent* comparison. We chose Jaccard similarity as the similarity metric for this analysis, as we are interested in measuring the similarity between two sets of keywords for each user, coming from the *GH-recent* and *SO-recent* text corpora. Figure 9 illustrates the distribution of the Jaccard similarity scores obtained from the *GH-recent – SO-recent* comparison. Plot A shows a density plot, while Plot B is a histogram of the Jaccard similarity scores. The distribution from Figure 9 shows that the majority of the population has very low Jaccard similarity scores, which means that there is a very low similarity between most user's *GH-recent* and *SO-recent* data. The histogram paints the clearest picture by showing that the highest relative percentage of Jaccard similarity scores are {0, 0.05, 0.10 and 1}.

Figure 10 drives home this point by creating five non-overlapping sub-intervals of Jaccard similarity scores from the [0,1] range, classifying each sub-interval of values to a scale of *No, Slight, Moderate, Large and Very Large Overlap.* When comparing

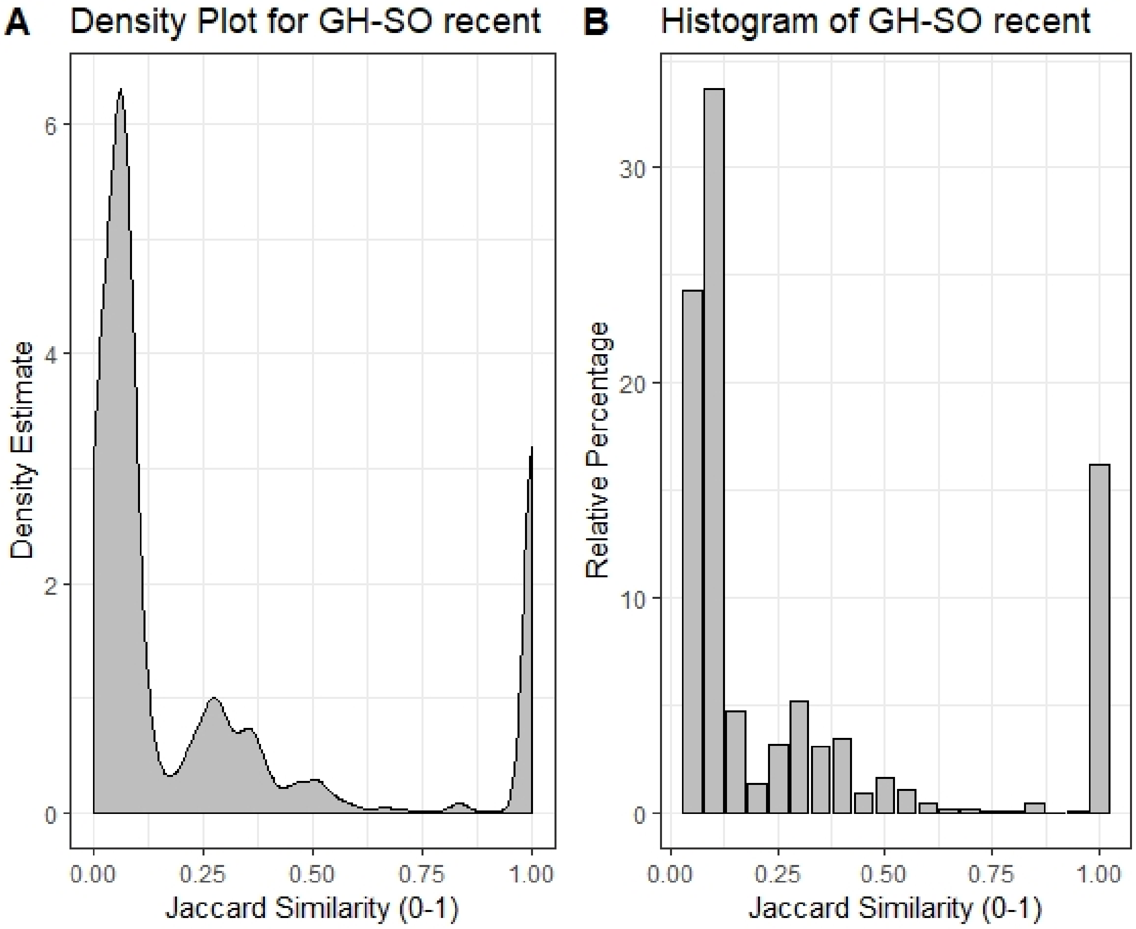


Figure 9: Density plot(A) and histogram(B) of GH-recent and SO-recent comparison.

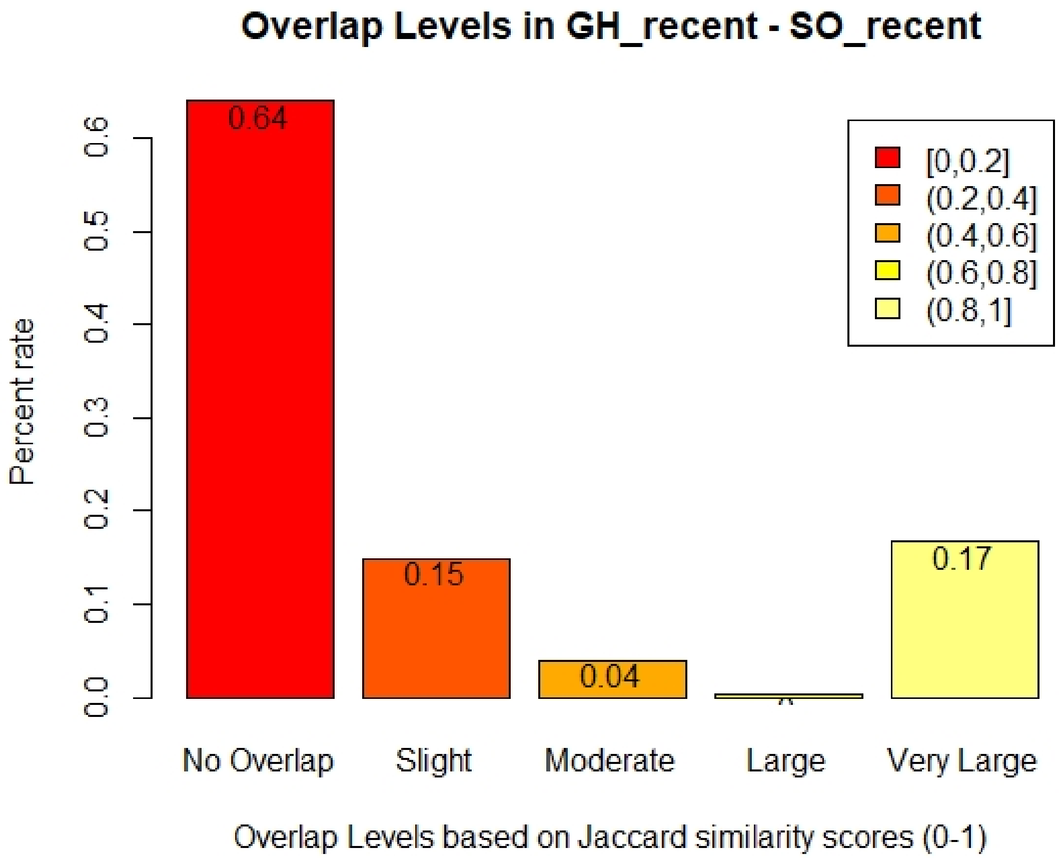


Figure 10: Overlap levels between GH-recent and SO-recent data sets.

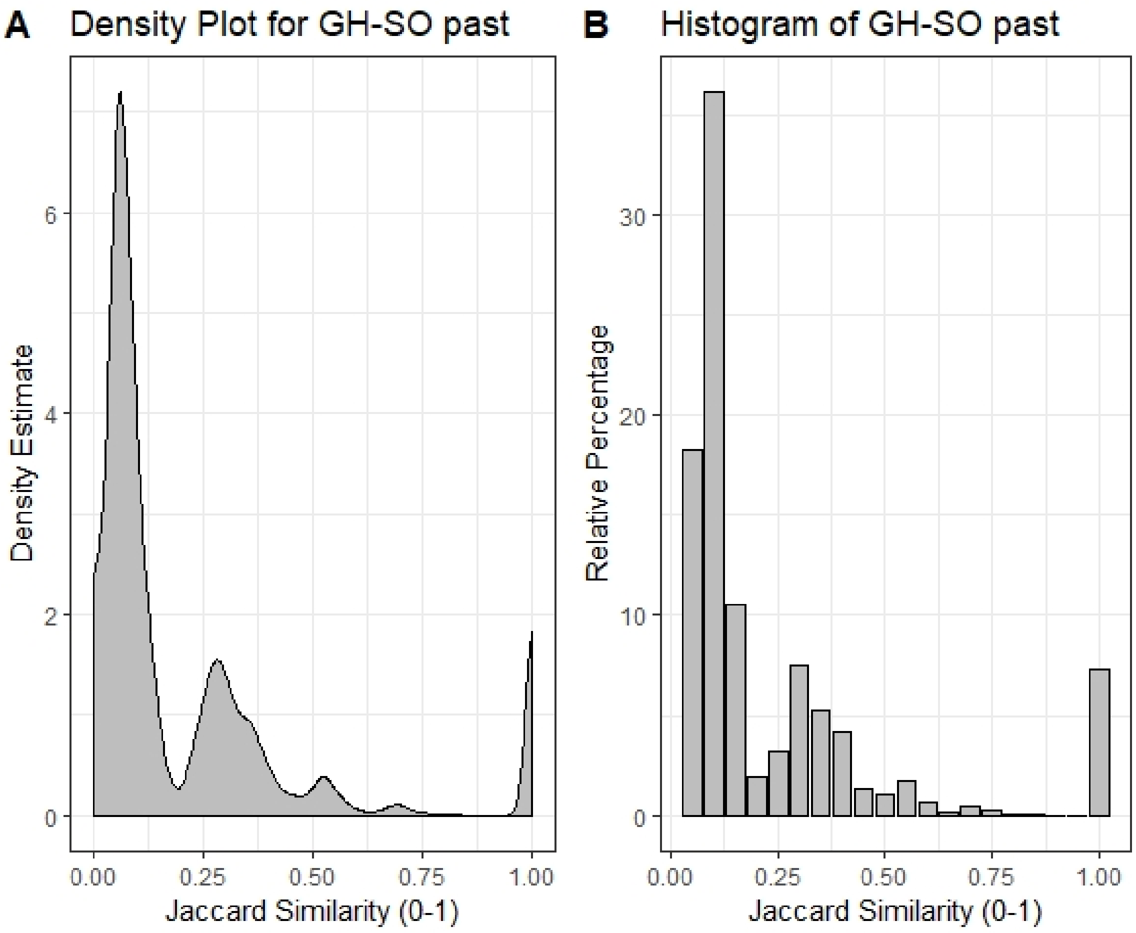


Figure 11: Density plot(A) and histogram(B) of GH-past and SO-past comparison.

the GH-recent – SO-recent text corpora Figure 10 shows that 64% of the population has no overlap, 15% has slight overlap, 4% has moderate overlap, and 17% has a very large overlap.

Figure 11 and 12 show the results of comparing the past activity of users on GitHub and Stack Overflow. This will be referred to as the *GH-past – SO-past* comparison. Same as above, Jaccard similarity is the similarity metric chosen to measure the similarity between two sets of keywords for each user, coming from the *GH-past* and *SO-past* text corpora. Figure 11 illustrates the distribution of the Jaccard similarity scores obtained from the *GH-past – SO-past* comparison. Plot A shows a density

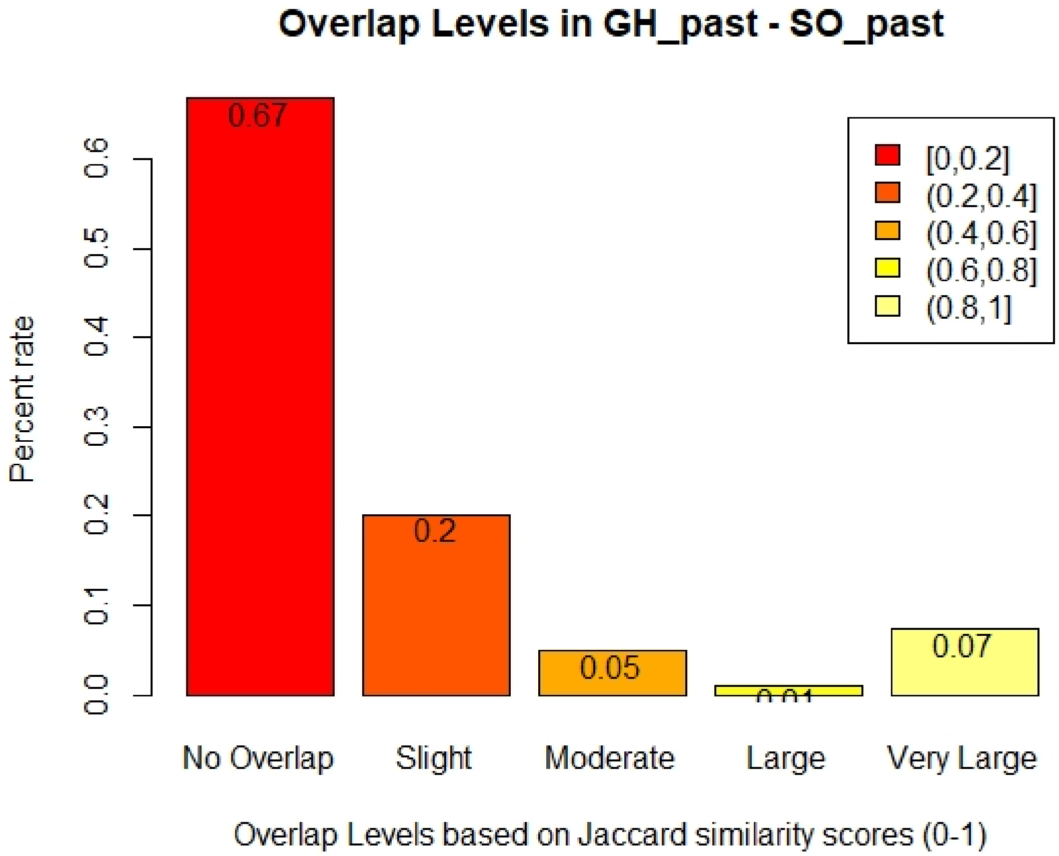


Figure 12: Overlap levels between GH-past and SO-past data sets.

plot, while Plot B is a histogram of the Jaccard similarity scores. The distribution from Figure 11 is similar to the one from Figure 9 and it shows that the majority of the population has very low Jaccard similarity scores. The density plot shows that most of the population has Jaccard similarity scores of {0.05, 0.10, 0.15}, while there are quite a few extreme scores of 0 and 1. This distribution suggests that there is a very low similarity between most user's expertise terms found in the *GH-past* and *SO-past* corpora.

Figure 12 confirms the findings above by considering the same five overlap levels as in Figure 10. When comparing the expertise terms in the GH-past - SO-past text corpora, Figure 12 shows that 67% of the population has no overlap, 20% has slight overlap, 5% has moderate overlap, 1% has large overlap, and 7% has a very large overlap.

Answer to RQ2: The comparison of GH-recent and SO-recent text corpora showed that 64% of the population have no overlap, while the GH-past and SO-past comparison concluded that 67% of the population have no overlap. These results suggest that developers build different expertise profiles on GitHub and Stack Overflow.

4.3 Cross-platform Transferable Knowledge

This section is concerned with discovering what knowledge is transferable from one platform to another. This question is answered by qualitatively analyzing the *Top*-30 most frequent common expertise terms for each user between the two platforms. For the experiment setup behind this analysis, we refer you to Section 3.7.3.

Table 13 displays the frequencies of common expertise terms in the pair of {GH-past, SO-past} text corpora, while Table 14 presents the same kind of common expertise terms in the pair of {GH-recent, SO-recent} text corpora.

In Table 13 one can notice a general pattern of mostly web development related terms showing up in the *Top*-30 most frequent common expertise terms used in GH-past and SO-past text corpora. Examples of such terms include *javascript, web, https, html, client, http, page, website*, and *api*. Another noticeable trend is the source code and version control related keywords, such as *library, code, project, tool, file, github, script, source, language, implementation, test, package, repository*, and *method*. The first pattern could suggest that most common conversations in the GH-past and SO-past text corpora involve discussions about web development. The second pattern is intuitive, as both collaborative platforms contain source code and

Table 13: Most common words in GH-past and SO-past text corpora.

Ranking	Keyword	Frequency	Ranking	Keyword	Frequency
1	library	43,123	16	base	16,798
2	code	37,621	17	implementation	16,670
3	simple	32,762	18	client	15,833
4	type	30,948	19	test	15,723
5	javascript	30,044	20	http	15,674
6	project	26,255	21	page	15,423
7	web	25,083	22	game	13,890
8	tool	24,967	23	website	13,255
9	https	24,738	24	package	12,982
10	file	24,737	25	repository	11,690
11	html	22,333	26	add	11,421
12	github	21,317	27	method	11,421
13	script	20,318	28	line	11,252
14	source	20,266	29	api	11,144
15	language	19,855	30	datum	10,980

Table 14: Most common words in GH-recent and SO-recent text corpora.

Ranking	Keyword	Frequency	Ranking	Keyword	Frequency
1	test	56,302	16	heroku	15,764
2	simple	51,226	17	buildpack	15,764
3	library	44,781	18	cli	15,764
4	app	43,750	19	rb	15,764
5	api	35,881	20	activerecord	15,764
6	base	26,075	21	rspec	15,764
7	client	25,381	22	active	15,764
8	code	22,052	23	github	15,297
9	file	21,538	24	web	12,890
10	application	19,620	25	add	12,849
11	https	16,764	26	change	12,849
12	ruby	15,764	27	remove	12,849
13	rail	15,764	28	check	12,849
14	ember	15,764	29	make	11,231
15	gem	15,764	30	comment	11,231

project related software artifacts, thus a possible knowledge transfer could occur by the cross-platform interaction of developers.

In Table 14 one can see the same general pattern of web development related terms dominated the *Top*-30 most frequent common expertise terms used in GH-recent and SO-recent text corpora, however, there is one noticeable discrepancy. In the *recent* text corpora (containing data between 2016 and 2019) the list of common expertise terms have shifted towards *Ruby and Heroku* related terms. Examples of this pattern include keywords like *ruby, rail, gem, heroku, buildpack, rb, activerecord*, and *rspec.* The web development pattern is also present in Table 14 by terms such as *app, api, client, application, https, ember*, and *web.* Finally, there is also a smaller cluster of source code and version control related keywords, such as *test, library, code, file,* *github*, and *comment*. These patterns suggest that most common conversations in GH-recent and SO-recent text corpora involve discussions about web development, source code, version control, and there might have been a popular trend in *Ruby on Rails* discussions that the LDA model's topics detected more often than other programming language related discussions.

The co-occuring keywords in both Table 13 and Table 14 are *test, simple, library, api, base, client, code, file, https, github, web*, and *add*. These keywords could be subjectively labelled as a combination of *source code, GitHub and web development* related terms.

Answer to RQ3: Common expertise terms used in both GitHub and Stack Overflow text corpora suggest that *source code, version control and web development* related skills are most transferable knowledge.

4.4 Expertise Evolution

The *Expertise Evolution Analysis* is concerned with comparing the evolution of expertise of the same users on Stack Overflow and GitHub. This analysis should answer the research question about how developer expertise changes over time on the two collaborative platforms. In this analysis, the change over time of GitHub and Stack Overflow text corpora are being analyzed. Two separate comparisons are considered: GH-past with GH-recent, and SO-past with SO-recent. For further details on the experiment setup behind this analysis see Section 3.7.4.

Figure 13 and Figure 14 show the results of comparing the past and recent activity of users on GitHub. For the rest of the analysis, this will be referred to as the *GH past-recent* comparison. We chose *Jaccard Distance* as the distance metric for this analysis, as we are interested in measuring the difference, thus distance (or dissimilarity) between two sets of keywords for each user, coming from the *GH-past*

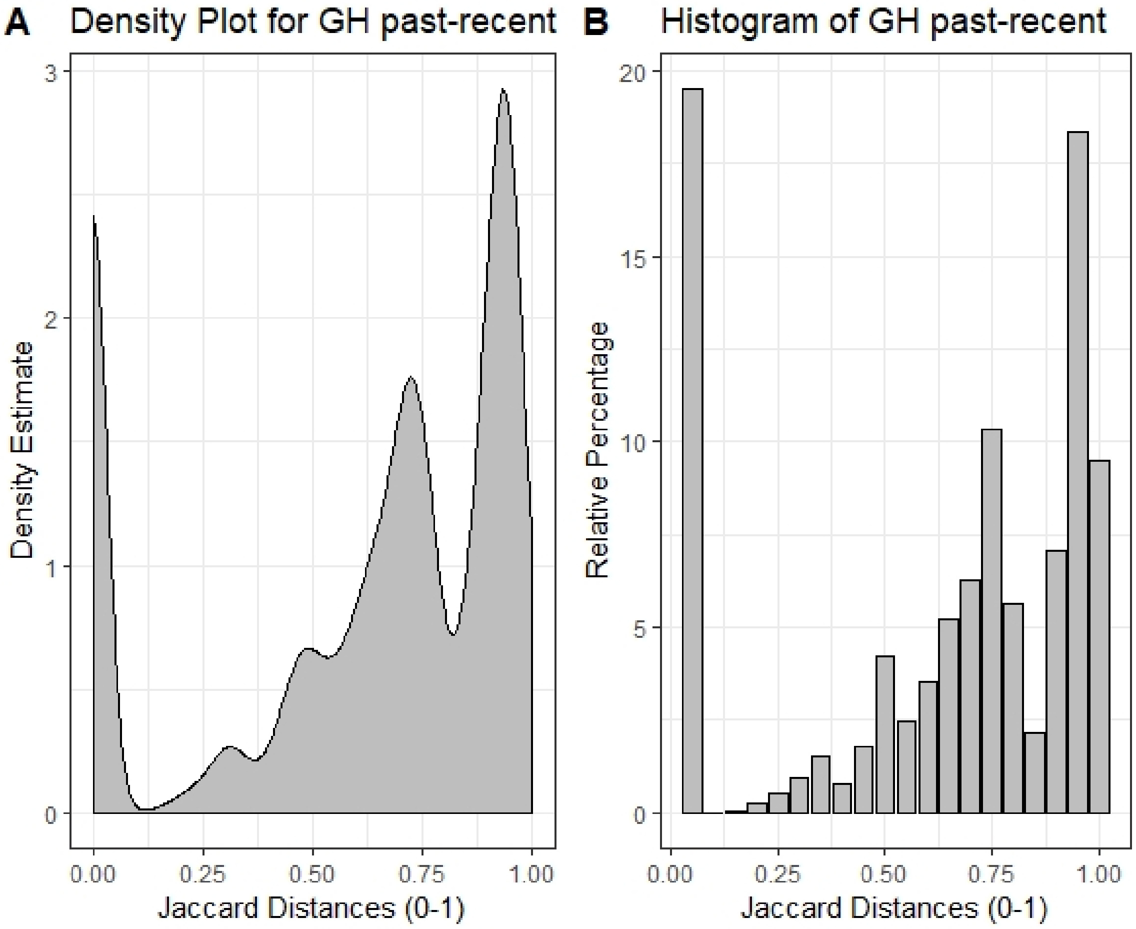


Figure 13: Density plot(A) and histogram(B) of GitHub past and recent comparison.

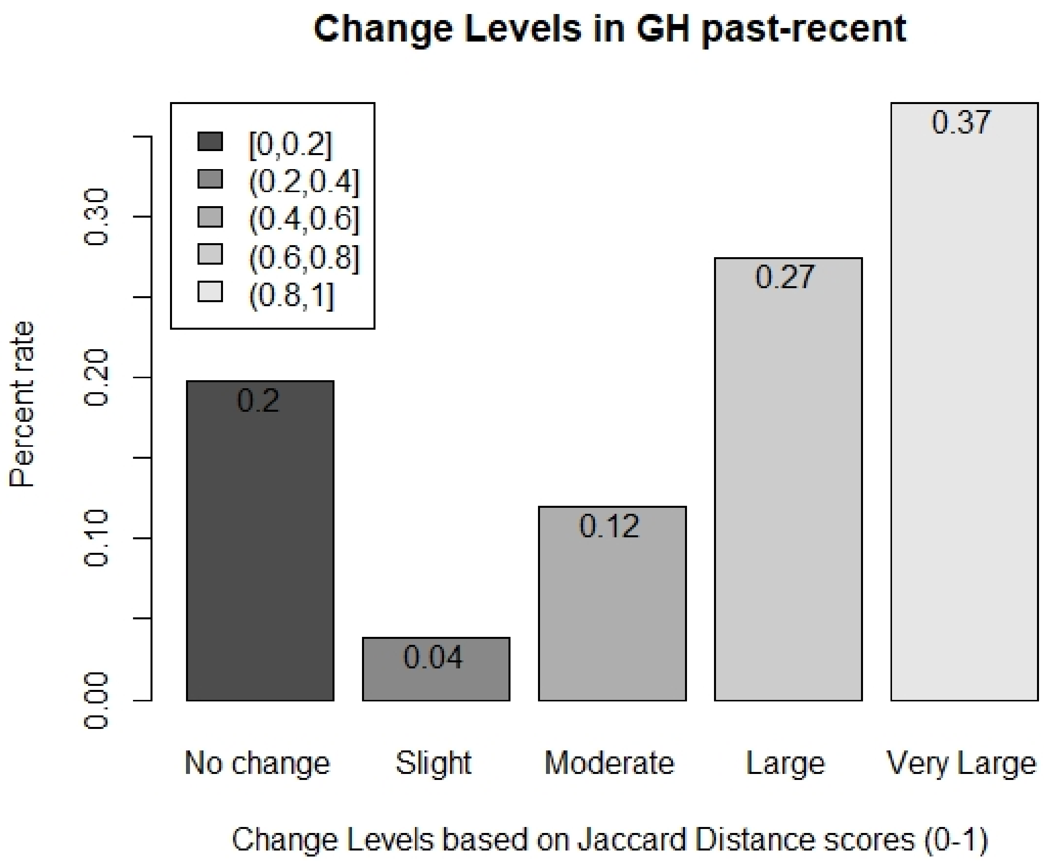


Figure 14: Change levels between GitHub past and recent data sets.

and *GH-recent* text corpora.

Figure 13 illustrates the distribution of the *Jaccard Distance* scores obtained from the *GH past-recent* comparison. Plot A shows a density plot, while Plot B is a histogram of the *Jaccard Distance* scores. The distribution from Figure 13 shows that the majority of the population has very large *Jaccard Distance* scores, which means that there is significant difference between most users' *GH-past* and *GH-recent* data. However, we can see on the histogram that there is a spike at *Jaccard Distance* score of 0, meaning that there are some users who have the same expertise data in both *GH-past* and *GH-recent* text corpora.

The distribution seen in Figure 13 suggests a diverse, mixed population of users between high and low *Jaccard Distance* scores. Figure 14 should paint a more clear picture by creating five non-overlapping sub-intervals of *Jaccard Distance* scores from the [0,1] range, classifying each sub-interval of values to a scale of *No, Slight, Moderate, Large and Very Large Change.* The *GH past-recent* comparison in Figure 14 shows that 20% of the population has no change, 4% has a slight change, 12% has moderate change, 27% has a large change, while 37% has a very large change over time. This result concludes that most of the developer population has largely changed their expertise over time, however, 20% of users maintain the same expertise profile on GitHub over time.

Figure 15 and Figure 16 show the results of comparing the past and recent activity of users on Stack Overflow. For the rest of the analysis, this will be referred to as the *SO past-recent* comparison. In this analysis, we are measuring the difference between two sets of keywords for each user coming from the *SO-past* and *SO-recent* text corpora, thus we use *Jaccard Distance.* Figure 15 illustrates the distribution of the *Jaccard Distance* scores obtained from the *SO past-recent* comparison. Plot A is a density plot, while Plot B is a histogram of the *Jaccard Distance* scores.

The distribution in Figure 15 shows many spikes at various *Jaccard Distance* values, which shows a very specific behaviour. From the histogram it can be seen that the largest spike it at a value of 0, meaning that some users have the same expertise data in both *SO-past* and *SO-recent* text corpora. The second trend shows that some user activity comparison having the same *Jaccard Distance* values of 0.40, 0.50 and 0.60, meaning that their more recent expertise differs from the past one.

The distribution seen in Figure 15 suggests a diverse population of both high and low *Jaccard Distance* scores. Figure 16 offers more details about the population by creating five non-overlapping sub-intervals of *Jaccard Distance* scores, classifying each sub-interval of values to a scale of *No, Slight, Moderate, Large and Very Large Change.*

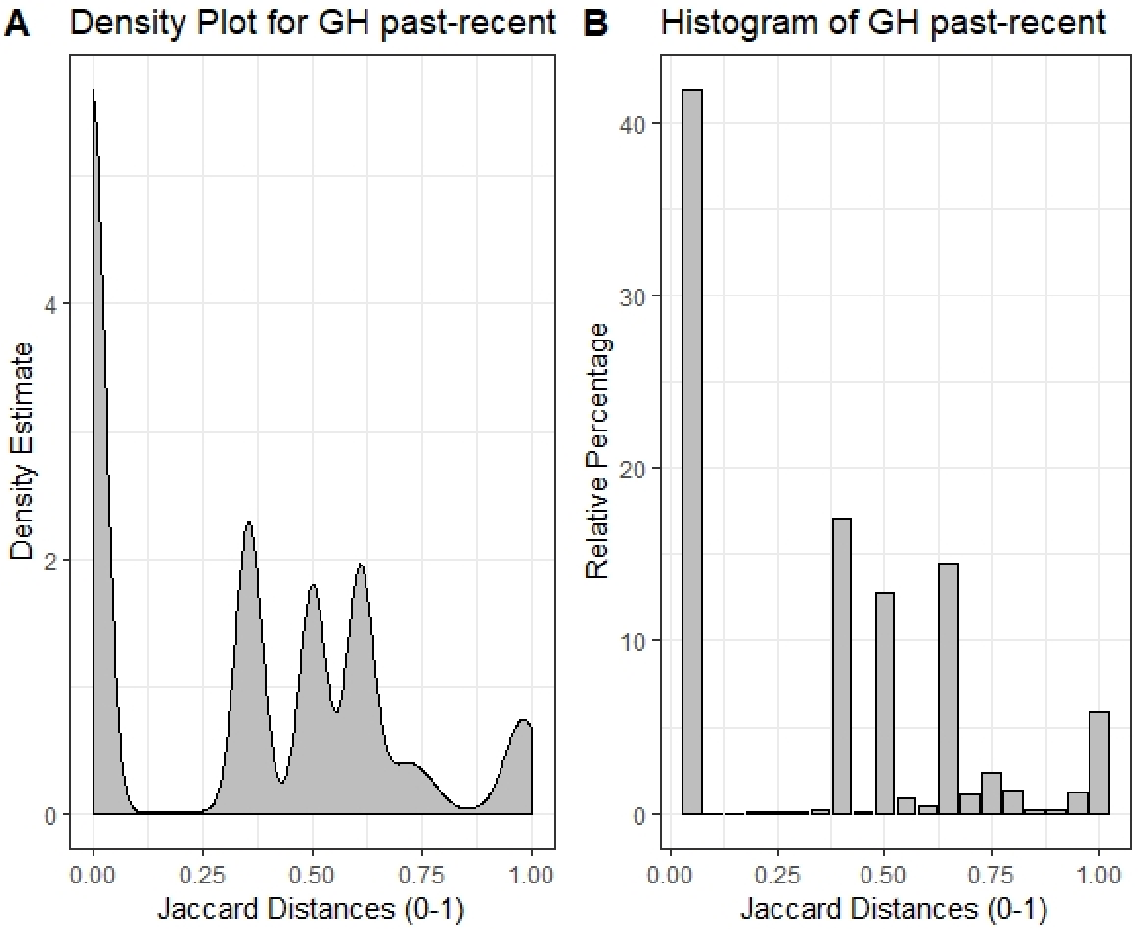


Figure 15: Density plot(A) and histogram(B) of Stack Overflow past and recent comparison.

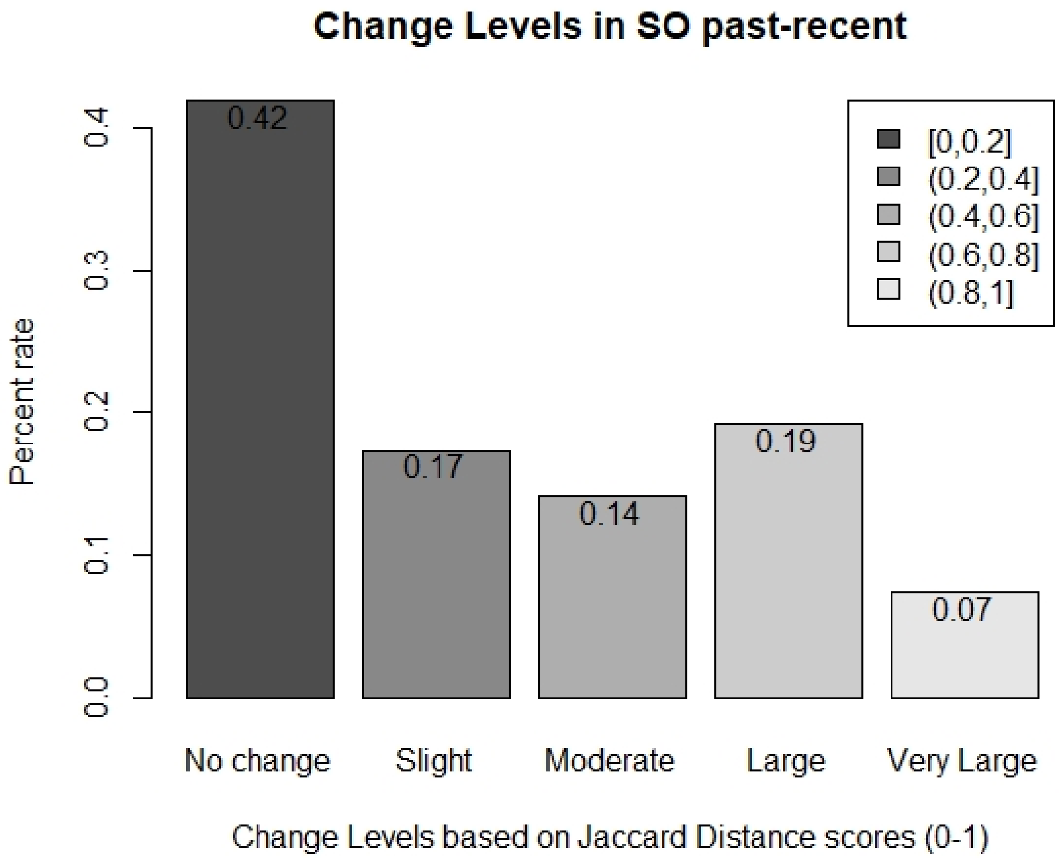


Figure 16: Change levels between Stack Overflow past and recent data sets.

The *SO past-recent* comparison in Figure 16 reveals that 42% of the population has no change, 17% has slight change, 14% has moderate change, 19% has a large change, while only 7% has a very large change over time, thus concluding that most analyzed developer population on Stack Overflow has *no, slight or moderate changes* in their expertise over time.

Answer to RQ4: The comparison of GH past-recent text corpora showed that 64% of the analyzed population of developers has either large or very large changes in their expertise, concluding that most of the analyzed GitHub population has largely changed their expertise over time. For the comparison of SO past-recent text corpora, this is not the case, as 42% of the population has no change, while 31% of the population has either slight or moderate changes in their expertise, concluding that most of the analyzed Stack Overflow population did not or only slightly changed their expertise over time.

All data sets used in this work are made publicly available on Zenodo[1]. Currently, all source code that was used to obtain these results is hosted on a private GitHub repository [2], but it will be made public after the publication of this work.

[1] `https://zenodo.org/record/3696079`

[2] `https://github.com/norberte/Mining-StackOverflow-Github`

Chapter 5

Discussion

In this chapter, we present the overall findings in Section 5.1, offer the implications of this study in Section 5.2, and discuss potential threats to validity in Section 5.3.

5.1 Findings

Chapter 3 and Chapter 4, i.e., Methodology and Results, respectively, have shown that it is possible to learn and extract the expertise of software developers through topic modeling and data analysis. We have demonstrated how user expertise profiles can be constructed from user activity data on collaborative platforms, which then could be leveraged into developer expertise. Multiple novel, robust, data-driven techniques have been designed to successfully extract the contextual expertise GitHub and Stack Overflow users. Based on the results from Section 4.1, the overall best performing technique is **W2V_AVG**, *Expertise Extraction using Pre-trained Word2Vec based User and Topic Embeddings obtained by Average-pooling.*

One important discovery is that for Experiment *1A* and *2A* performed on GitHub data the Word2Vec based models (**W2V_AVG** and **W2V_MAX**) outperform the LDA based models (**LDA_AVG** and **LDA_MAX**), but this is not necessary true for Experiment *1B* and *2B* performed on Stack Overflow data. This discovery is intuitive from one perspective: Word2Vec based models use higher dimensional user and topic embeddings than the LDA based models, thus they can learn more contextual details from the text. The counter-intuitive fact about this insight is that the Word2Vec model was trained on Stack Overflow data, but it performs better on Experiments *1A* and *2A*, which are based on GitHub user activity data, not Stack Overflow.

Cross-platform developer expertise has been thoroughly studied as well. Our analysis conducted on the similarity of expertise profiles across two separate collaborative platforms can enrich and better support the software engineering community's understanding of developer expertise built across multiple platforms. The most surprising finding, in our opinion, is that expertise developed by users on GitHub and Stack Overflow is mostly different, with only a small portion of the population having similar expertise profiles on both collaborative platforms. Furthermore, our cross-platform developer expertise study also focused on what are the similarities between the two platforms and what kind of skills are the most transferable from one platform to another. The software engineering community, as well as software developers, can benefit from knowing that *source code, version control and web development* related skills are most transferable knowledge, and they can be learnt by using both Stack Overflow and GitHub. The cross-platform developer expertise study also focused on a better understanding of the expertise evolution on Stack Overflow and GitHub. This study discovered that most of the analyzed Stack Overflow and GitHub developers have largely changed their expertise over time by adapting to new APIs, frameworks and libraries. We are not able to answer if the change is in the right direction, but the analyzed Stack Overflow and GitHub population are likely to improve their skills, gaining knowledge and building expertise, instead of diminishing their current expertise.

Throughout the results of the *Expertise Extraction* task one interesting discovery was made. Each technique applied to GitHub user profile data produces significantly higher *Cosine Similarity* scores than when applied to Stack Overflow user profile data. For the GitHub data set most models output *Cosine Similarity* scores between 0.70 and 0.80, while for the Stack Overflow data set the same similarity metric is between 0.50 and 0.60. This represents a notable difference in contextual semantic similarity, especially if one looks at the sample cosine similarity scores in Table 5. The novel, data-driven techniques presented in Section 3.7 cannot explain why the same technique applied to two different data sets results in a significant difference in performance. Each technique is built upon a trained LDA model, which is sensitive to the quality and even order of input data. Treude and Wagner [17]'s study concluded that GitHub and Stack Overflow text corpora have different characteristics due to their software artifact related content and require careful model selection to achieve a good model fit. One can safely assume that the proposed techniques depend on the cleanliness of training data and careful LDA model fitting. We are hypothesizing that the difference in performance on GitHub and Stack Overflow data sets comes from the differences in characteristics of the two data sets. More specifically, we believe that the Stack Overflow data contains messier, less clean textual data, as Stack Overflow posts tend to contain both a textual description of a problem and in most cases source code is present in the post. We also hypothesize that the blend of natural text and source code in Stack Overflow posts is not cleaned up properly by the text pre-processing routine from Section 3.4. Unfortunately, as a consequence code snippets or keywords referring to source code do not get filtered out properly, and this could cause the drop in the contextual semantic similarity of the techniques when applied on the Stack Overflow data set.

The differences in characteristics of the two data sets are backed up by Treude and Wagner [17], who have studied the characteristics of GitHub and Stack Overflow corpora. They noticed that GitHub textual data contains fewer stop words than Stack Overflow data, which they believe indicates the difference of technical descriptions present in GitHub, contrary to Stack Overflow posts, which contain more casual discussions. Based on Treude and Wagner [17]'s study one other distinguishing feature between GitHub and Stack Overflow textual data is the median number of unique words (not stop words) in a document. Surprisingly, documents from GitHub on average contain almost twice as many unique words per document. It is worth noting that Treude and Wagner defined a document from GitHub as a repository's README file, not as an aggregated user activity profile. Nonetheless, Treude and Wagner showed that GitHub and Stack Overflow text corpora have different characteristics, and this claim affects the fitting of LDA models, as in their study GitHub corpora contained a significantly higher number of topics than Stack Overflow corpora. Coincidentally, this is the case in our study too, as the number of topics present in best performing GitHub and Stack Overflow models mentioned in Section 4.1.3 can support this claim.

Our LDA model configuration and hyper-parameter tuning can be compared to the results of Treude and Wagner [17]'s study on good topic modeling configurations. Similarities include both studies performing hyper-parameter tuning with almost identical parameter search-space, both studies using similar text pre-processing routines and the above mentioned fact that GitHub data contained a significantly higher number of topics than Stack Overflow data. The main difference between our work and Treude and Wagner's work is the choice of evaluation of a good model fit. While Treude and Wagner used perplexity, our approach can be defined as a task-based evaluation of model selection. Our work is focused on model interpretability and usefulness to software developers and other practitioners, thus we evaluate an LDA model based on how semantically similar expertise terms it can extract compared to human annotations. This type of task-based model evaluation is more in line with Campbell et al. [16]'s recommendation on the use of LDA models for software engineering tasks, which we agree on. We believe that an arbitrary evaluation metric such as perplexity would not be able to provide an accurate or proper model selection process for the task of expertise extraction. This is the reason why we chose to perform a task-based LDA model evaluation.

5.2 Implications

Implications of this research are numerous, with a variety of applications. We have defined four main target audiences for this research work: 1) recruiters and recruitment oriented services, 2) project managers and software companies, 3) Stack Overflow and GitHub users, and 4) research scientists. The implications for each one of these target audiences are explained next.

5.2.1 Recruiters and Recruitment Oriented Services

The first target audience group to consider is recruiters and recruitment oriented services. This research work has closed the gap between potential job candidates and their expertise by providing a data-driven alternative to simply reading the candidate's CV or resume. The implementation of our expertise extraction algorithm into a recruitment agency's internal system would allow our research to be used by recruiters for finding and hiring the talent with the desired expertise.

Our recommendation for recruiters is to not limit themselves only to employment-oriented platforms such as *LinkedIn*[1] (which could contain self-reporting bias; see Section 1.2), and instead consider expertise profiles obtained via data-driven approaches, which theoretically should be less biased and more reliable. The advantage of looking at the big picture via expertise profiles is that the top experts in a specific area can be identified, which is important to companies.

[1] `https://www.linkedin.com`

5.2.2 Project Managers and Software Companies

The second target audience of this research work is software companies and project managers who work on task assignments. Project managers are often faced with the task of deciding who is the best developer to handle a certain bug fix, code review or pull request. A task at hand needs to be assigned to one or more developers in the company. The optimal task assignment would allow the developer(s) with most expertise and experience in the domain to complete the task at hand. To achieve such an optimized task assignment the implementation of expertise detection and recommendation algorithms would be needed.

Our recommendation for this target audience is to consider integrating the task assignment system described above into their task management tools, as software companies could benefit from an advanced bug fixing or code review assignment system by increasing the effectiveness and efficiency of their work.

5.2.3 Stack Overflow and GitHub Users

The third group of target audiences are Stack Overflow and GitHub users, as the entire research work is about analyzing and better understanding their skills, progress made, behaviour and learning process. The ultimate goal of this research work is to make developers aware of their expertise, how to gain more of it, and become an expert.

After extensive analysis of *cross-platform developer expertise* we suggest that using multiple collaborative platforms is the optimal path towards gaining more knowledge and becoming an expert, as cross-platform developer expertise tends to be more diverse, thus creating further opportunities for faster and more effective ways of learning by collaboration.

5.2.4 Researchers and Research Scientists

Lastly, this research work is highly relevant to other researchers, data scientists and research scientists in the field of empirical software engineering and software artifact related data mining. More specifically, the use of LDA models and its combination with deep learning algorithms and other machine learning techniques is novel and innovative. This research work is a great example of how to convert large amounts of unstructured data into valuable insight by combining elements of both statistical learning and deep learning. In novel Technique 2 and Technique 3 from Section 3.7.1.2 and 3.7.1.3 elements of statistical learning, deep learning, feature extraction, downsampling and thresholding are combined to generate user and topic embeddings, which then can be leveraged into the mapping of users belonging to one or more topics.

Our recommendation for researchers and research scientists would be to consider using this type of ensemble algorithm design more in their research work, as it can be a valuable, innovative and effective way of combining powerful algorithms to create purely data-driven approaches. Another recommendation for researchers and research scientists working with textual data would be to consider using LDA models for more tasks. Campbell et al. [16] stated that LDA models can be used to "summarize, cluster, link, and pre-process" large amounts of unstructured data. Based on this claim the number of opportunities and use cases should be countless. Furthermore, LDA models have an advantage over other models by scaling very well even for enormous amounts of data, thus we would advise the researchers to consider using more topic modeling in their work.

5.3 Threats to Validity

Our study is subject to several threats to validity which we address in this section.

Data pre-processing: First, as previously mentioned, the blend of natural text and source code in Stack Overflow posts causes some challenges to the text pre-processing routine from Section 3.4, thus not all code snippets are cleaned up and filtered out properly. One might say, why was not a better text pre-processing routine applied to Stack Overflow posts. As a fact, the pre-processing of Stack Overflow posts was re-done several times, each time improving the routine iteratively to filter out as much noise as possible. We converged to the best possible text pre-processing routine (see Section 3.4.2) which contains a few compromises concerning filtering out the noise associated with code snippets.

Known LDA limitations: Boyd-Graber et al. [19] stated that there are a few problems with topics in LDA models. More specifically, they said it is likely that some topics will contain too general and specific words, or they might contain "stop-word-like" words. Other potential issues include mixed and chained topics, identical or nonsensical topics. Unfortunately, these risks exist when employing topic models. Our way of making sure that no such issues affected the results was to manually validate the topics generated by the best performing models and discard nonsensical topics. When it comes to the use of LDA models, we followed the best practices presented in Section 2.1.4, and carefully avoided all the pitfalls outlined in Section 2.1.4.3.

Use of relevance for ranking expertise terms: The next threat to validity is related to the suitability of our topic word ranking metric commonly used in topic modeling called *relevance* [80]. We are aware that *relevance* is not the only metric which can be used to rank words, but we chose this word ranking metric based on Sievert et al. [80]'s convincing novel method for choosing which topic words allow the best interpretation of a topic.

Expertise study limitations: During the expertise study (see Section 3.6) we ran into several challenges with processing and aggregating the human annotations. These challenges created some limitations for the ground truth data set. Firstly, the support of only a handful of frequent phrases (found in Section A.1) represents a limitation as full support of any phrases would be desired. Secondly, the inconsistencies in annotation length represent a limitation too since evaluating such ground truth data set could be challenging. To address these annotation length inconsistencies we had no choice but to merge the two annotations into one final annotation using a set union, and this could be a threat to validity. Other limitations related to the expertise study include the relatively small number of ground truth annotations that we could collect. We are aware that labelling 100 out of 83,550 Stack Overflow and GitHub users' expertise is a relatively small ratio, but that is not the actual ratio. We wanted to select only active users to be part of our expertise study, thus we narrowed down our random selection to a population of 675 users who are currently active on both GitHub and Stack Overflow. From those 675 users, we randomly selected 100 users to be labelled by our human annotators, thus we argue that 100 users are a large enough sample, considering the circumstances related to user activity.

Model selection using topic coherence: We chose the most promising coherence measures (*C_v, C_umass, C_npmi and C_uci*) from Röder et al. [42]'s work as evaluation metrics for four text corpora (GH-past, GH-recent, SO-past and SO-recent) explained in Section 3.3.3. One potential threat to validity could be that the above mentioned coherence measures are not the most suitable metrics for model selection. We are aware of other evaluation metrics, such as perplexity, and log-likelihood of held-out data set, but we believed that Röder et al.'s state-of-the-art topic coherence metrics are the best choice for model evaluation.

Data quality: A potential threat to validity could be related to data quality, especially in the SO-recent data set. This data set lacks active users, as only less than 1,000 out of 83,550 users have a significant amount of user activity, and the rest are inactive users. The same issue applies to the LDA model fitted on the SO-recent data set, as lack of data could lead to misleading topic trends in SO-recent. This is the nature of the data set, thus we, unfortunately, can not mitigate this data quality issue on the SO-recent data set. However, we completely avoided using the past and recent data sets for the *expertise extraction* task in the first research question, and used the SO-full and GH-full data sets for this task instead.

Unknown words: The last potential threat to validity is related to the handling of out-of-dictionary words in Efstathiou et al. [18]'s pre-trained Word2Vec model. We make use of this pre-trained model in our third novel technique (see Section 3.7.1.3) to look up software artifact related contextually aware vector representation of words. When we encountered out-of-dictionary words, rather than assigning a random vector or a vector of zeros, we skipped the word and did not assign any vector representation to it, so it does not corrupt the average or max-pooling calculations executed on the matrix containing the pre-trained embeddings. Fortunately, the percentage of unknown words with respect to the number of unique words in the dictionary is small. For the GitHub text corpus only 7.315 % of the words, while for the Stack Overflow text corpus 9.649 % of the words had no word2vec embeddings.

Chapter 6

Conclusion

Lastly, we conclude our work with a *summary of our contributions* in Section 6.1, then define *future research paths* in Section 6.2.

6.1 Summary of Contributions

This research work introduced novel, data-driven innovative ways into software developer expertise learning and made seven contributions to the research community. Firstly, we developed three novel techniques to extract developer expertise topics from Stack Overflow and GitHub. We have evaluated the proposed techniques using two separate experiments on two different data sets and found that technique **W2V_AVG** performs the best. Secondly, we completed a ground truth study to collect human annotations on developer expertise. Thirdly, we analyzed developer expertise trends on Stack Overflow and GitHub and discovered that expertise areas on GitHub tend to be few, and more general, while expertise areas on Stack Overflow tend to be numerous, and more specific.

Fourthly, we created four new data sets for user activity profiles on GitHub and Stack Overflow. Fifthly, we compared developer expertise across the above mentioned collaborative platforms and showed that the expertise developed by users on GitHub and Stack Overflow is mostly different. As our sixth contribution, we provided empirical evidence about knowledge transfer between the two online collaborative platforms and made the discovery that *source code, version control and web development* related skills are the most transferable knowledge between Stack Overflow and GitHub. Finally, our last contribution was the results of the analysis conducted on developer expertise evolution trends from GitHub and Stack Overflow. Our study suggests that most of the analyzed GitHub population has largely changed their expertise over time, while most analyzed Stack Overflow population did not, or only slightly changed their expertise over time.

Lastly, the implications of our work are versatile, with numerous opportunities for real-world applications. We identified four main target audiences that could benefit from our work: recruiters & recruitment oriented services, project managers & software companies, Stack Overflow & GitHub users, and research scientists. We believe that this research will positively impact their work and potentially help them solve their problems.

6.2 Future Work

There are plenty of research opportunities to be built on top of the current work. First, we recommend that future research should include the separation of natural text from source code snippets, especially when working with Stack Overflow posts. This task can be achieved by including a filtering stage in the data cleaning pipeline, potentially using the NLoN[1] library developed by Mäntylä et al. [88].

Second, there are several innovative alternatives for discovering better user and topic vector representation, such as Moody [89]'s work on combining the LDA and Word2Vec models into LDA2Vec[2] or Dieng et al. [90]'s experimental work on topic

[1] `https://github.com/M3SOulu/NLoN`

[2] `https://github.com/cemoody/lda2vec`

modeling in embedding spaces implemented as ETM[3].

Third, author-topic models [91] could be considered to model user activities as data being generated from multiple authors. In this case, the author-topic model would learn the distribution of topics as a mixture of distributions associated with each user's data.

Fourth, follow-up research could be conducted on measuring how accurately can we learn cross-platform expertise in software engineering. This research avenue would require a survey to be conducted, asking Stack Overflow and GitHub users how accurately our expertise terms extracted represent their actual perceived expertise. Further work along this line could be conducted on designing machine learning algorithms to predict, summarize or classify a user's expertise area.

Last, potential follow up work could be on conducting an empirical analysis of similarities and differences between different expertise area topical distributions obtained from the already trained LDA models.

[3] https://github.com/adjidieng/ETM

List of References

[1] I. Sommerville, *Software Engineering GE.* Pearson Australia Pty Limited, 2016.

[2] M. Nisen, "Here's why you only have a 0.2% chance of getting hired at google," Oct 2014.

[3] A. Mockus and J. D. Herbsleb, "Expertise browser: a quantitative approach to identifying expertise," in *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, pp. 503–512, IEEE, 2002.

[4] S. Baltes and S. Diehl, "Towards a theory of software development expertise," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 187–200, ACM, 2018.

[5] C. Teyton, J.-R. Falleri, F. Morandat, and X. Blanc, "Find your library experts," in *2013 20th Working Conference on Reverse Engineering (WCRE)*, pp. 202–211, IEEE, 2013.

[6] G. J. Greene and B. Fischer, "Cvexplorer: Identifying candidate developers by mining and exploring their open source contributions," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pp. 804–809, 2016.

[7] C. Hauff and G. Gousios, "Matching github developer profiles to job advertisements," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pp. 362–366, IEEE, 2015.

[8] J. C. S. M. Yijun Tian, Waii Ng, "Geek talents: Who are the top experts on github and stack overflow?," *Computers, Materials & Continua*, vol. 61, no. 2, pp. 465–479, 2019.

[9] X. Wang, C. Huang, L. Yao, B. Benatallah, and M. Dong, "A survey on expert recommendation in community question answering," *Journal of Computer Science and Technology*, vol. 33, no. 4, pp. 625–653, 2018.

[10] L. Zhang, Y. Zou, B. Xie, and Z. Zhu, "Recommending relevant projects via user behaviour: an exploratory study on github," in *Proceedings of the 1st International Workshop on Crowd-based Software Development Methods and Technologies*, pp. 25–30, 2014.

[11] K. Johnson, "Github passes 100 million repositories," Nov 2018.

[12] G. Gousios, B. Vasilescu, A. Serebrenik, and A. Zaidman, "Lean ghtorrent: Github data on demand," in *Proceedings of the 11th working conference on mining software repositories*, pp. 384–387, 2014.

[13] "The state of the octoverse," Oct 2019.

[14] J. Liao, G. Yang, D. Kavaler, V. Filkov, and P. Devanbu, "Status, identity, and language: A study of issue discussions in github," *PloS one*, vol. 14, no. 6, p. e0215059, 2019.

[15] Y. Tian, P. S. Kochhar, E.-P. Lim, F. Zhu, and D. Lo, "Predicting best answerers for new questions: An approach leveraging topic modeling and collaborative voting," in *International Conference on Social Informatics*, pp. 55–68, Springer, 2013.

[16] J. C. Campbell, A. Hindle, and E. Stroulia, "Latent dirichlet allocation: extracting topics from software engineering data," in *The art and science of analyzing software data*, pp. 139–159, Elsevier, 2015.

[17] C. Treude and M. Wagner, "Predicting good configurations for github and stack overflow topic models," in *Proceedings of the 16th International Conference on Mining Software Repositories*, pp. 84–95, IEEE Press, 2019.

[18] V. Efstathiou, C. Chatzilenas, and D. Spinellis, "Word embeddings for the software engineering domain," in *Proceedings of the 15th International Conference on Mining Software Repositories*, pp. 38–41, ACM, 2018.

[19] J. Boyd-Graber, D. Mimno, and D. Newman, "Care and feeding of topic models: Problems, diagnostics, and improvements," *Handbook of mixed membership models and their applications*, vol. 225255, 2014.

[20] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[21] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.

[22] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," *arXiv preprint arXiv:1607.01759*, 2016.

[23] N. S. Sarwan, "An intuitive understanding of word embeddings: From count vectors to word2vec," June 2017.

[24] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[25] T. L. Griffiths and M. Steyvers, "Finding scientific topics," *Proceedings of the National academy of Sciences*, vol. 101, no. suppl 1, pp. 5228–5235, 2004.

[26] A. Agrawal, W. Fu, and T. Menzies, "What is wrong with topic modeling? and how to fix it using search-based software engineering," *Information and Software Technology*, vol. 98, pp. 74–88, 2018.

[27] H. M. Wallach, D. M. Mimno, and A. McCallum, "Rethinking lda: Why priors matter," in *Advances in neural information processing systems*, pp. 1973–1981, 2009.

[28] A. A. Bangash, H. Sahar, S. Chowdhury, A. W. Wong, A. Hindle, and K. Ali, "What do developers know about machine learning: a study of ml discussions on stackoverflow," in *Proceedings of the 16th International Conference on Mining Software Repositories*, pp. 260–264, IEEE Press, 2019.

[29] A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, "How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms," in *Proceedings of the 2013 International Conference on Software Engineering*, pp. 522–531, IEEE Press, 2013.

[30] A. Hindle, C. Bird, T. Zimmermann, and N. Nagappan, "Relating requirements to implementation via topic analysis: Do topics extracted from requirements make sense to managers and developers?," in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pp. 243–252, IEEE, 2012.

[31] R. Arun, V. Suresh, C. V. Madhavan, and M. N. Murthy, "On finding the natural number of topics with latent dirichlet allocation: Some observations," in *Pacific-Asia conference on knowledge discovery and data mining*, pp. 391–402, Springer, 2010.

[32] J. Cao, T. Xia, J. Li, Y. Zhang, and S. Tang, "A density-based method for adaptive lda model selection," *Neurocomputing*, vol. 72, no. 7-9, pp. 1775–1781, 2009.

[33] R. Deveaud, E. SanJuan, and P. Bellot, "Accurate and effective latent concept modeling for ad hoc information retrieval," *Document numérique*, vol. 17, no. 1, pp. 61–84, 2014.

[34] W. Zhao, J. J. Chen, R. Perkins, Z. Liu, W. Ge, Y. Ding, and W. Zou, "A heuristic approach to determine an appropriate number of topics in topic modeling," in *BMC bioinformatics*, vol. 16, p. S8, BioMed Central, 2015.

[35] J. Chang, S. Gerrish, C. Wang, J. L. Boyd-Graber, and D. M. Blei, "Reading tea leaves: How humans interpret topic models," in *Advances in neural information processing systems*, pp. 288–296, 2009.

[36] A. Arwan, S. Rochimah, and R. J. Akbar, "Source code retrieval on stackoverflow using lda," in *2015 3rd International Conference on Information and Communication Technology (ICoICT)*, pp. 295–299, IEEE, 2015.

[37] K.-Y. M. Chen and Y. Wang, "Latent dirichlet allocation," tech. rep., Technical report, Department of Electrical and Computing engineering . . . , 2011.

[38] D. Newman, T. Baldwin, L. Cavedon, E. Huang, S. Karimi, D. Martinez, F. Scholer, and J. Zobel, "Visualizing search results and document collections using topic maps," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 8, no. 2-3, pp. 169–175, 2010.

[39] D. Newman, J. H. Lau, K. Grieser, and T. Baldwin, "Automatic evaluation of topic coherence," in *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 100–108, Association for Computational Linguistics, 2010.

[40] D. Mimno, H. M. Wallach, E. Talley, M. Leenders, and A. McCallum, "Optimizing semantic coherence in topic models," in *Proceedings of the conference on empirical methods in natural language processing*, pp. 262–272, Association for Computational Linguistics, 2011.

[41] P. F. Baldi, C. V. Lopes, E. J. Linstead, and S. K. Bajracharya, “A theory of aspects as latent topics,” in *ACM Sigplan Notices*, vol. 43, pp. 543–562, ACM, 2008.

[42] M. Röder, A. Both, and A. Hinneburg, “Exploring the space of topic coherence measures,” in *Proceedings of the eighth ACM international conference on Web search and data mining*, pp. 399–408, ACM, 2015.

[43] “Jaccard index,” Oct 2019.

[44] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, “Bleu: a method for automatic evaluation of machine translation,” in *Proceedings of the 40th annual meeting on association for computational linguistics*, pp. 311–318, Association for Computational Linguistics, 2002.

[45] “Cosine similarity,” May 2019.

[46] B. Vasilescu, V. Filkov, and A. Serebrenik, “Stackoverflow and github: Associations between software development and crowdsourced knowledge,” in *2013 International Conference on Social Computing*, pp. 188–195, IEEE, 2013.

[47] A. S. Badashian, A. Esteki, A. Gholipour, A. Hindle, and E. Stroulia, “Involvement, contribution and influence in github and stack overflow,” in *Proceedings of 24th Annual International Conference on Computer Science and Software Engineering*, pp. 19–33, IBM Corp., 2014.

[48] R. K.-W. Lee and D. Lo, “Github and stack overflow: Analyzing developer interests across multiple social collaborative platforms,” in *International Conference on Social Informatics*, pp. 245–256, Springer, 2017.

[49] B. Vasilescu, A. Serebrenik, P. Devanbu, and V. Filkov, “How social q&a sites are changing knowledge sharing in open source software communities,” in *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*, pp. 342–354, ACM, 2014.

[50] A. Barua, S. W. Thomas, and A. E. Hassan, “What are developers talking about? an analysis of topics and trends in stack overflow,” *Empirical Software Engineering*, vol. 19, no. 3, pp. 619–654, 2014.

[51] G. Gousios, “The ghtorent dataset and tool suite,” in *Proceedings of the 10th working conference on mining software repositories*, pp. 233–236, IEEE Press, 2013.

[52] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining github," in *Proceedings of the 11th working conference on mining software repositories*, pp. 92–101, ACM, 2014.

[53] S. A. Chowdhury and A. Hindle, "Mining stackoverflow to filter out off-topic irc discussion," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, pp. 422–425, IEEE Press, 2015.

[54] D. Yang, P. Martins, V. Saini, and C. Lopes, "Stack overflow in github: any snippets there?," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pp. 280–290, IEEE, 2017.

[55] S. Baltes, L. Dumani, C. Treude, and S. Diehl, "Sotorrent: Reconstructing and analyzing the evolution of stack overflow posts," in *Proceedings of the 15th International Conference on Mining Software Repositories*, pp. 319–330, ACM, 2018.

[56] S. Baltes, C. Treude, and S. Diehl, "Sotorrent: Studying the origin, evolution, and usage of stack overflow code snippets," in *Proceedings of the 16th International Conference on Mining Software Repositories*, pp. 191–194, IEEE Press, 2019.

[57] S. S. Manes and O. Baysal, "How often and what stackoverflow posts do developers reference in their github projects?," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pp. 235–239, IEEE, 2019.

[58] P. Chatterjee, K. Damevski, L. Pollock, V. Augustine, and N. A. Kraft, "Exploratory study of slack q&a chats as a mining source for software engineering tools," in *Proceedings of the 16th International Conference on Mining Software Repositories*, pp. 490–501, IEEE Press, 2019.

[59] G. A. A. Prana, C. Treude, F. Thung, T. Atapattu, and D. Lo, "Categorizing the content of github readme files," *Empirical Software Engineering*, vol. 24, no. 3, pp. 1296–1327, 2019.

[60] A. S. Badashian, A. Hindle, and E. Stroulia, "Crowdsourced bug triaging: Leveraging q&a platforms for bug assignment," in *International Conference on Fundamental Approaches to Software Engineering*, pp. 231–248, Springer, 2016.

[61] J. E. Montandon, L. L. Silva, and M. T. Valente, "Identifying experts in software libraries and frameworks among github users," in *Proceedings of the 16th International Conference on Mining Software Repositories*, pp. 276–287, IEEE Press, 2019.

[62] R. Sindhgatta, “Identifying domain expertise of developers from source code,” in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 981–989, 2008.

[63] P. L. Li, A. J. Ko, and A. Begel, “What distinguishes great software engineers?,” *Empirical Software Engineering*, pp. 1–31, 2019.

[64] Z. Liao, H. Jin, Y. Li, B. Zhao, J. Wu, and S. Liu, “Devrank: Mining influential developers in github,” in *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pp. 1–6, IEEE, 2017.

[65] M. Hammad, M. Hammad, and H. Bani-Salameh, “Identifying designers and their design knowledge,” *International Journal of Software Engineering and Its Applications*, vol. 7, no. 6, pp. 277–288, 2013.

[66] H. Kagdi, M. Gethers, D. Poshyvanyk, and M. Hammad, “Assigning change requests to software developers,” *Journal of software: Evolution and Process*, vol. 24, no. 1, pp. 3–33, 2012.

[67] O. Alonso, P. T. Devanbu, and M. Gertz, “Expertise identification and visualization from cvs,” in *Proceedings of the 2008 international working conference on Mining software repositories*, pp. 125–128, 2008.

[68] D. Surian, N. Liu, D. Lo, H. Tong, E.-P. Lim, and C. Faloutsos, “Recommending people in developers’ collaboration network,” in *2011 18th Working Conference on Reverse Engineering*, pp. 379–388, IEEE, 2011.

[69] T. T. Nguyen, T. N. Nguyen, E. Duesterwald, T. Klinger, and P. Santhanam, “Inferring developer expertise through defect analysis,” in *2012 34th International Conference on Software Engineering (ICSE)*, pp. 1297–1300, IEEE, 2012.

[70] Y. Yu, H. Wang, G. Yin, and T. Wang, “Reviewer recommendation for pull-requests in github: What can we learn from code review and bug assignment?,” *Information and Software Technology*, vol. 74, pp. 204–218, 2016.

[71] M. M. Rahman, C. K. Roy, and J. A. Collins, “Correct: code reviewer recommendation in github based on cross-project and technology experience,” in *Proceedings of the 38th International Conference on Software Engineering Companion*, pp. 222–231, 2016.

[72] E. Constantinou and G. M. Kapitsaki, “Identifying developers’ expertise in social coding platforms,” in *2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 63–67, IEEE, 2016.

[73] R. Venkataramani, A. Gupta, A. Asadullah, B. Muddu, and V. Bhat, "Discovery of technical expertise from open source code repositories," in *Proceedings of the 22nd International Conference on World Wide Web*, pp. 97–98, 2013.

[74] G. Silvestri, J. Yang, A. Bozzon, and A. Tagarelli, "Linking accounts across social networks: the case of stackoverflow, github and twitter.," in *KDWeb*, pp. 41–52, 2015.

[75] W. Huang, W. Mo, B. Shen, Y. Yang, and N. Li, "Cpdscorer: Modeling and evaluating developer programming ability across software communities.," in *SEKE*, pp. 87–92, 2016.

[76] X. Zhang, T. Wang, G. Yin, C. Yang, Y. Yu, and H. Wang, "Devrec: a developer recommendation system for open source repositories," in *International Conference on Software Reuse*, pp. 3–11, Springer, 2017.

[77] M. Hoffman, F. R. Bach, and D. M. Blei, "Online learning for latent dirichlet allocation," in *advances in neural information processing systems*, pp. 856–864, 2010.

[78] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and psychological measurement*, vol. 20, no. 1, pp. 37–46, 1960.

[79] K. Krippendorff, *Content Analysis: An Introduction to Its Methodology*, vol. 3. SAGE, 2013.

[80] C. Sievert and K. Shirley, "Ldavis: A method for visualizing and interpreting topics," in *Proceedings of the workshop on interactive language learning, visualization, and interfaces*, pp. 63–70, 2014.

[81] J. Ricco, "What is max pooling in convolutional neural networks?," Jun 2017.

[82] C. De Boom, S. Van Canneyt, T. Demeester, and B. Dhoedt, "Representation learning for very short texts using weighted word embedding aggregation," *Pattern Recognition Letters*, vol. 80, pp. 150–156, 2016.

[83] I. T. Jolliffe, "Principal component analysis and factor analysis," in *Principal component analysis*, pp. 115–128, Springer, 1986.

[84] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[85] E. Bisong, “Google colaboratory,” in *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, pp. 59–64, Springer, 2019.

[86] R. C. Team *et al.*, “R: A language and environment for statistical computing,” 2013.

[87] H. Wickham, *ggplot2: elegant graphics for data analysis.* Springer, 2016.

[88] M. V. Mäntylä, F. Calefato, and M. Claes, “Natural language or not (nlon) a package for software engineering text analysis pipeline,” in *Proceedings of the 15th International Conference on Mining Software Repositories*, pp. 387–391, 2018.

[89] C. E. Moody, “Mixing dirichlet topic models and word embeddings to make lda2vec,” *arXiv preprint arXiv:1605.02019*, 2016.

[90] A. B. Dieng, F. J. Ruiz, and D. M. Blei, “Topic modeling in embedding spaces,” *arXiv preprint arXiv:1907.04907*, 2019.

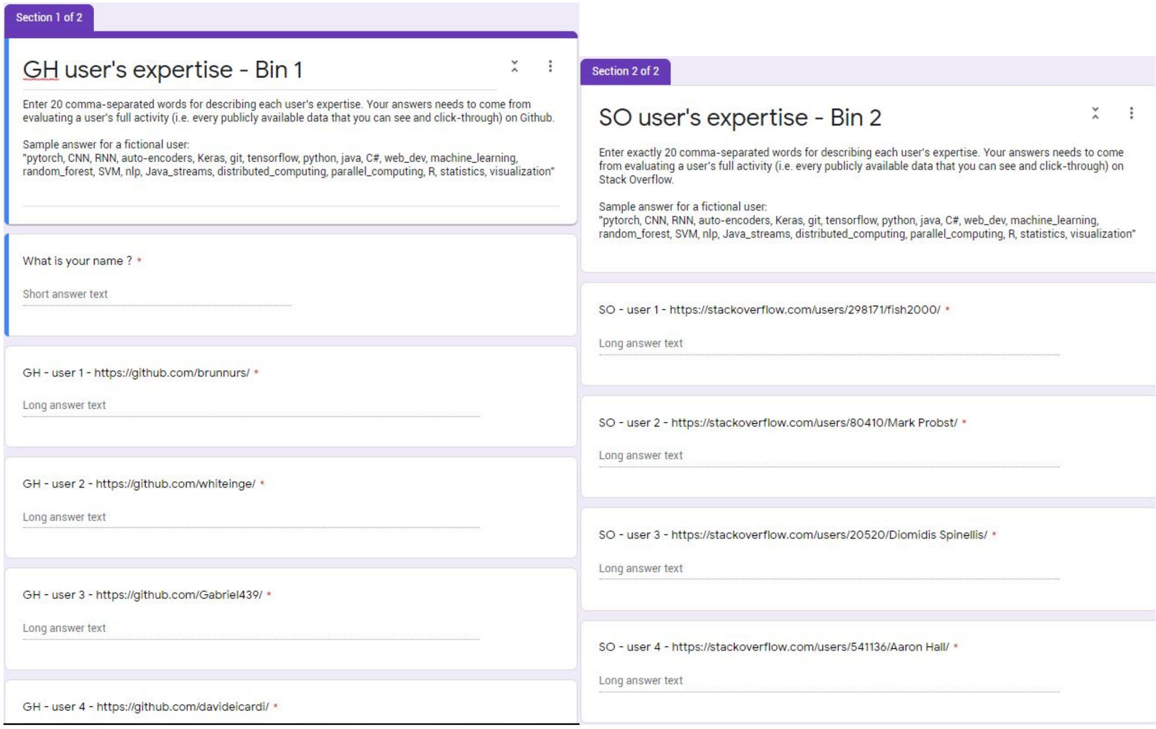
[91] M. Rosen-Zvi, T. Griffiths, M. Steyvers, and P. Smyth, “The author-topic model for authors and documents,” *arXiv preprint arXiv:1207.4169*, 2012.

Appendix A

Appendix

A.1 Table of Frequently Used Expressions

List of Frequent Expressions in Stack Overflow and GitHub data			
web_services	jupyter_notebook	machine_learning	visual_studio
objective_c	unit_testing	deep_learning	artificial_intelligence
vim_script	data_visualization	data_science	software_engineering
front_end	android_sdk	google_analytics	android_development
mac_os	web_application	data_analysis	functional_programming
web_design	restful_api	image_processing	web_development
emacs_lisp	software_design	data_structures	app_development



Section 1 of 2

GH user's expertise - Bin 1

Enter 20 comma-separated words for describing each user's expertise. Your answers needs to come from evaluating a user's full activity (i.e. every publicly available data that you can see and click-through) on Github.

Sample answer for a fictional user:
"pytorch, CNN, RNN, auto-encoders, Keras, git, tensorflow, python, java, C#, web_dev, machine_learning, random_forest, SVM, nlp, Java_streams, distributed_computing, parallel_computing, R, statistics, visualization"

What is your name ? *

Short answer text

GH - user 1 - https://github.com/brunnurs/ *

Long answer text

GH - user 2 - https://github.com/whiteinge/ *

Long answer text

GH - user 3 - https://github.com/Gabriel439/ *

Long answer text

GH - user 4 - https://github.com/davideicardi/ *

Section 2 of 2

SO user's expertise - Bin 2

Enter exactly 20 comma-separated words for describing each user's expertise. Your answers needs to come from evaluating a user's full activity (i.e. every publicly available data that you can see and click-through) on Stack Overflow.

Sample answer for a fictional user:
"pytorch, CNN, RNN, auto-encoders, Keras, git, tensorflow, python, java, C#, web_dev, machine_learning, random_forest, SVM, nlp, Java_streams, distributed_computing, parallel_computing, R, statistics, visualization"

SO - user 1 - https://stackoverflow.com/users/298171/fish2000/ *

Long answer text

SO - user 2 - https://stackoverflow.com/users/80410/Mark Probst/ *

Long answer text

SO - user 3 - https://stackoverflow.com/users/20520/Diomidis Spinellis/ *

Long answer text

SO - user 4 - https://stackoverflow.com/users/541136/Aaron Hall/ *

Long answer text

(a) Section I

(b) Section II

Figure 17: Sample Google form expertise study.

A.2 Expertise Study Details

A.2.1 Sample Expertise Study

A.2.2 Stack Overflow user profile URLs in expertise study.

Table 15: List of Stack Overflow user profile URLs in study.

Sample	Stack Overflow User Profile URL
1	https://stackoverflow.com/users/1081551/u_b/
2	https://stackoverflow.com/users/127816/whiteinge/
3	https://stackoverflow.com/users/1026598/Gabriel Gonzalez/
4	https://stackoverflow.com/users/209727/Davide Icardi/
5	https://stackoverflow.com/users/562883/Jonathan Stray/
6	https://stackoverflow.com/users/369874/Paul Kehrer/
7	https://stackoverflow.com/users/94559/smarx/
8	https://stackoverflow.com/users/102441/Eric/
9	https://stackoverflow.com/users/1270695/A5C1D2H2I1M1N2O1R2T1/
10	https://stackoverflow.com/users/456809/Breedly/
11	https://stackoverflow.com/users/298171/fish2000/
12	https://stackoverflow.com/users/80410/Mark Probst/

13	https://stackoverflow.com/users/20520/Diomidis Spinellis/
14	https://stackoverflow.com/users/541136/Aaron Hall/
15	https://stackoverflow.com/users/14955/Thilo/
16	https://stackoverflow.com/users/272689/vimalloc/
17	https://stackoverflow.com/users/1006669/Giovanni Lovato/
18	https://stackoverflow.com/users/101152/Karel Bílek/
19	https://stackoverflow.com/users/65311/devth/
20	https://stackoverflow.com/users/529187/Xiao Peng - ZenUML.com/
21	https://stackoverflow.com/users/470682/mkso/
22	https://stackoverflow.com/users/763799/basgys/
23	https://stackoverflow.com/users/446536/geon/
24	https://stackoverflow.com/users/1162018/Daniel Iser/
25	https://stackoverflow.com/users/257568/ArtemGr/
26	https://stackoverflow.com/users/43839/Tom Swirly/
27	https://stackoverflow.com/users/24998/Jack/
28	https://stackoverflow.com/users/1190453/danbst/
29	https://stackoverflow.com/users/870615/qwtel/
30	https://stackoverflow.com/users/557368/rosenfeld/
31	https://stackoverflow.com/users/33518/Tomas Petricek/

32	https://stackoverflow.com/users/492393/Dmitrii Sorin/
33	https://stackoverflow.com/users/177275/Yurik/
34	https://stackoverflow.com/users/324790/Ivan Nikolaev/
35	https://stackoverflow.com/users/554531/Keith Hughitt/
36	https://stackoverflow.com/users/1074377/Kage/
37	https://stackoverflow.com/users/20394/Mike Samuel/
38	https://stackoverflow.com/users/94148/aleung/
39	https://stackoverflow.com/users/1051598/Roman Elizarov/
40	https://stackoverflow.com/users/736714/deitch/
41	https://stackoverflow.com/users/239247/anatoly techtonik/
42	https://stackoverflow.com/users/747768/Kirill Dmitrenko/
43	https://stackoverflow.com/users/663505/pirxpilot/
44	https://stackoverflow.com/users/227267/Matti Virkkunen/
45	https://stackoverflow.com/users/1968/Konrad Rudolph/
46	https://stackoverflow.com/users/511200/danneu/
47	https://stackoverflow.com/users/161801/asmeurer/
48	https://stackoverflow.com/users/613198/rsp/
49	https://stackoverflow.com/users/1457000/Adam D. Ruppe/
50	https://stackoverflow.com/users/1294964/Marko Lukša/

51	https://stackoverflow.com/users/7122/David Arno/
52	https://stackoverflow.com/users/214488/memo/
53	https://stackoverflow.com/users/246616/mVChr/
54	https://stackoverflow.com/users/180783/astrofrog/
55	https://stackoverflow.com/users/159695/Galaxy/
56	https://stackoverflow.com/users/512596/Joey/
57	https://stackoverflow.com/users/248296/warvariuc/
58	https://stackoverflow.com/users/173314/wjt/
59	https://stackoverflow.com/users/1305344/Jacek Laskowski/
60	https://stackoverflow.com/users/561698/Andrew/
61	https://stackoverflow.com/users/875915/Rob Bednark/
62	https://stackoverflow.com/users/135786/Ben Lesh/
63	https://stackoverflow.com/users/11543/mjs/
64	https://stackoverflow.com/users/1048571/Jon Douglas/
65	https://stackoverflow.com/users/1463744/fagiani/
66	https://stackoverflow.com/users/308174/Larry Cai/
67	https://stackoverflow.com/users/267416/Islam Wazery/
68	https://stackoverflow.com/users/3109/MatthieuGD/
69	https://stackoverflow.com/users/297131/Evgenii/

70	https://stackoverflow.com/users/1240763/Gary Russell/
71	https://stackoverflow.com/users/11238/Randy Sugianto 'Yuku'/
72	https://stackoverflow.com/users/477476/Cactus/
73	https://stackoverflow.com/users/211327/Alan H./
74	https://stackoverflow.com/users/146041/Aaron McDaid/
75	https://stackoverflow.com/users/503899/kraenhansen/
76	https://stackoverflow.com/users/81071/Koraktor/
77	https://stackoverflow.com/users/632242/Leszek/
78	https://stackoverflow.com/users/537554/ryenus/
79	https://stackoverflow.com/users/242457/David Jones/
80	https://stackoverflow.com/users/62600/Todd Menier/
81	https://stackoverflow.com/users/136285/malat/
82	https://stackoverflow.com/users/163197/danielkza/
83	https://stackoverflow.com/users/744520/Jacques Kvam/
84	https://stackoverflow.com/users/14405/Robin like the bird/
85	https://stackoverflow.com/users/1282247/dodomorandi/
86	https://stackoverflow.com/users/771848/alecxe/
87	https://stackoverflow.com/users/397020/Adron/
88	https://stackoverflow.com/users/1209004/johan/

89	https://stackoverflow.com/users/435093/slhck/
90	https://stackoverflow.com/users/408195/Diogo Melo/
91	https://stackoverflow.com/users/432903/prayagupd/
92	https://stackoverflow.com/users/341772/Max/
93	https://stackoverflow.com/users/189599/Antonello/
94	https://stackoverflow.com/users/501266/Keith Bennett/
95	https://stackoverflow.com/users/208809/Gordon/
96	https://stackoverflow.com/users/541949/ivoba/
97	https://stackoverflow.com/users/241510/Ruudjah/
98	https://stackoverflow.com/users/771838/rossta/
99	https://stackoverflow.com/users/933228/petrelharp/
100	https://stackoverflow.com/users/552182/Muaz Khan/

A.2.3 GitHub User Profile URLs in Expertise Study.

Table 16: List of GitHub user profile URLs in study

Sample ID	**GitHub User Profile URL**
1	https://github.com/brunnurs/

2	https://github.com/whiteinge/
3	https://github.com/Gabriel439/
4	https://github.com/davideicardi/
5	https://github.com/jstray/
6	https://github.com/reaperhulk/
7	https://github.com/smarx/
8	https://github.com/eric-wieser/
9	https://github.com/mrdwab/
10	https://github.com/Freyert/
11	https://github.com/fish2000/
12	https://github.com/schani/
13	https://github.com/dspinellis/
14	https://github.com/aaronchall/
15	https://github.com/thiloplanz/
16	https://github.com/vimalloc/
17	https://github.com/heruan/
18	https://github.com/runn1ng/
19	https://github.com/devth/
20	https://github.com/MrCoder/

21	https://github.com/khanmurtuza/
22	https://github.com/basgys/
23	https://github.com/geon/
24	https://github.com/danieliser/
25	https://github.com/ArtemGr/
26	https://github.com/rec/
27	https://github.com/jacktasia/
28	https://github.com/danbst/
29	https://github.com/qwtel/
30	https://github.com/rosenfeld/
31	https://github.com/tpetricek/
32	https://github.com/1999/
33	https://github.com/nyurik/
34	https://github.com/V0idExp/
35	https://github.com/khughitt/
36	https://github.com/kjjgibson/
37	https://github.com/mikesamuel/
38	https://github.com/aleung/
39	https://github.com/elizarov/

40	https://github.com/deitch/
41	https://github.com/techtonik/
42	https://github.com/dmikis/
43	https://github.com/pirxpilot/
44	https://github.com/mvirkkunen/
45	https://github.com/klmr/
46	https://github.com/danneu/
47	https://github.com/asmeurer/
48	https://github.com/rsp/
49	https://github.com/adamdruppe/
50	https://github.com/luksa/
51	https://github.com/DavidArno/
52	https://github.com/memo/
53	https://github.com/mreinhardt/
54	https://github.com/astrofrog/
55	https://github.com/galaxy001/
56	https://github.com/joeyh/
57	https://github.com/warvariuc/
58	https://github.com/wjt/

59	https://github.com/jaceklaskowski/
60	https://github.com/almartin82/
61	https://github.com/RobBednark/
62	https://github.com/blesh/
63	https://github.com/ithinkihaveacat/
64	https://github.com/JonDouglas/
65	https://github.com/fagiani/
66	https://github.com/larrycai/
67	https://github.com/wazery/
68	https://github.com/matthieugd/
69	https://github.com/evgenyneu/
70	https://github.com/garyrussell/
71	https://github.com/yukuku/
72	https://github.com/gergoerdi/
73	https://github.com/alanhogan/
74	https://github.com/aaronmcdaid/
75	https://github.com/kraenhansen/
76	https://github.com/koraktor/
77	https://github.com/lpryszcz/

78	https://github.com/ryenus/
79	https://github.com/drj11/
80	https://github.com/tmenier/
81	https://github.com/malaterre/
82	https://github.com/danielkza/
83	https://github.com/jwkvam/
84	https://github.com/RobinIsTheBird/
85	https://github.com/dodomorandi/
86	https://github.com/alecxe/
87	https://github.com/Adron/
88	https://github.com/bitsgalore/
89	https://github.com/slhck/
90	https://github.com/dmelo/
91	https://github.com/prayagupd/
92	https://github.com/nanodeath/
93	https://github.com/tsutomi/
94	https://github.com/keithrbennett/
95	https://github.com/gooh/
96	https://github.com/ivoba/

97	https://github.com/generateui/
98	https://github.com/rossta/
99	https://github.com/petrelharp/
100	https://github.com/muaz-khan/

A.2.4 Ground Truth Expertise Human Annotations

Table 17: Ground truth human annotations for Stack Overflow users.

Sample	Human Annotation
1	['aspdot_net', 'ui', 'visual_studio', 'ios', 'localization', 'audiotoolbox', 'debugging', 'json', 'oriented', 'jpa', 'pcl', 'swift', 'svg', 'cisco', 'ruby', 'angularjs', 'mvc', 'object', 'xamarin', 'xml', 'android', 'sql', 'javascript', 'ajax', 'bootstrap', 'jquery', 'spring', 'router', 'sqlite', 'identicon', 'ubuntu', 'java', 'html']
2	['routing', 'server', 'linux', 'cron', 'json', 'cryptography', 'based', 'dvcs', 'packages', 'event', 'macros', 'cryptographically', 'python', 'random', 'source', 'openvpn', 'sql', 'javascript', 'desktop', 'latex', 'macos', 'hash', 'drive', 'browser', 'open', 'mercurial', 'remote', 'debugging', 'audio', 'tex', 'firefox', 'inheritance', 'rebooting', 'enough', 'hard', 'ajax', 'unix', 'files', 'raspbian', 'rxjs', 'networking', 'ubuntu', 'windows', 'vcs', 'git']

3	[’scala’, ’interpreters’, ’classes’, ’c’, ’polymorphic’, ’monads’, ’data’, ’python’, ’random’, ’parsing’, ’datatypes’, ’concurrency’, ’type’, ’replicate’, ’generate’, ’pattern’, ’conduit’, ’haskell’, ’monad’, ’transformers’, ’types’, ’matching’, ’functional_programming’, ’clojure’, ’isomorphism’, ’io’, ’extensions’, ’algebraic’, ’interface’, ’state’, ’class’, ’lazy’, ’wrap’, ’function’, ’sequences’, ’interference’, ’evaluation’, ’compositions’, ’data_structures’, ’monad_transformers’, ’namespaces’, ’encoding’]
4	[’aspdot_net’, ’unit_testing’, ’webserver’, ’visual_studio’, ’entity’, ’html’, ’database’, ’debugging’, ’nodejs’, ’expression’, ’entity_framework’, ’docker’, ’powershell’, ’async’, ’web_development’, ’nvm’, ’angularjs’, ’mvc’, ’dot_net’, ’entity_sql’, ’c#’, ’sql’, ’javascript’, ’mvc3’, ’wcf’, ’system’, ’threadpool’, ’framework’, ’azure’, ’mock’, ’msbuild’, ’windows’, ’css’, ’cloud’]
5	[’asynchronous’, ’unit_testing’, ’pandas’, ’scala’, ’unit’, ’nodejs’, ’netty’, ’cpp’, ’iframe’, ’selenium’, ’docker’, ’ruby’, ’karma’, ’parser’, ’dailog’, ’python’, ’csv’, ’jest’, ’multiparadigm’, ’javascript’, ’dataframe’, ’bootstrap’, ’reactjs’, ’react’, ’test’, ’memsql’, ’enzyme’, ’modal’, ’hash’, ’webpack’, ’http’, ’java’, ’sandbox’]
6	[’webserver’, ’cryptographic’, ’server’, ’safari’, ’scrapy’, ’oriented’, ’centos’, ’cryptography’, ’python’, ’ssl’, ’web_development’, ’source’, ’cpython’, ’javascript’, ’toolkit’, ’protocol’, ’web’, ’openssl’, ’jquery’, ’macos’, ’xcode’, ’open’, ’html’, ’enyption’, ’hashing’, ’php’, ’mavericks’, ’nodejs’, ’docker’, ’ruby’, ’object’, ’jenkins’, ’pip’, ’pips’, ’side’, ’osv’, ’ubuntu’]

7	['php', 'arrays', 'json', 'guru', 'fanatic', 'windows', 'slack', 'python', 'node', 'phoenix', 'ethereum', 'mvc', 'api', 'dot_net', 'microsoft', 'android', 'media', 'c#', 'blob', 'javascript', 'ecmascript', 'container', 'jquery', 'blockchain', 'dropbox', 'storage', 'azure', 'rendering', 'jwplayer', 'java', 'html', 'cloud']
8	['xslt', 'git', 'php', 'array', 'scipy', 'arrays', 'electorate', 'json', 'built', 'selectors', 'regex', 'serialization', 'overriding', 'list', 'math', 'python', 'c++', 'dictionary', 'groovy', 'post', 'object', 'css', 'javascript', 'numpy', 'ecmascript', 'jquery', 'algorithm', 'hyperlink', 'java', 'html', 'in']
9	['replacing', 'data_manipulation', 'json', 'frame', 'management', 'plot', 'data', 'string', 'python', 'reshape', 'random', 'tidyr', 'numpy', 'manipulation', 'msword', 'datatables', 'aggregate', 'pattern', 'latex', 'loops', 'r', 'bioinformatics', 'knitr', 'melt', 'count', 'regex', 'sorting', 'list', 'matrix', 'excel', 'csv', 'time', 'strsplit', 'dataframe', 'grep', 'series', 'visualization', 'ubuntu']
10	['php', 'webserver', 'database', 'debugging', 'json', 'nodejs', 'c', 'clojure', 'oriented', 'cloud', 'selenium', 'docker', 'ruby', 'ansible', 'python', 'elixir', 'angularjs', 'mvc', 'object', 'css', 'sql', 'javascript', 'ajax', 'java_streams', 'jquery', 'macos', 'verilog', 'sqlite', 'java', 'html', 'git']

11	[’git’, ’tweepy’, ’web_design’, ’modules’, ’c’, ’halide’, ’programming’, ’python’, ’web_development’, ’django’, ’memory’, ’sql’, ’javascript’, ’numpy’, ’web’, ’jquery’, ’mobile’, ’models’, ’html’, ’aspdot_net’, ’parts’, ’sharepoint’, ’swift’, ’virtual’, ’c++’, ’api’, ’design’, ’objective_c’, ’front_end’, ’cython’, ’fullstack’, ’cpython’, ’graphic’]
12	[’linux’, ’single’, ’json’, ’c’, ’sign’, ’lisp’, ’oauth’, ’python’, ’c#’, ’web’, ’mobile’, ’jvm’, ’bash’, ’macos’, ’java’, ’database’, ’mutithreading’, ’ssh’, ’ios’, ’functional_programming’, ’clojure’, ’multithreading’, ’scripting’, ’swift’, ’jackson’, ’codable’, ’security’, ’objective_c’, ’unix’, ’ocaml’, ’iphone’, ’git’]
13	[’testing’, ’cygwin’, ’code’, ’server’, ’kernel’, ’linux’, ’debugging’, ’c_lang’, ’c’, ’software_engineering’, ’rstudio’, ’verification’, ’programming’, ’mac_os’, ’python’, ’c++’, ’source’, ’leaks’, ’shell’, ’memory’, ’sql’, ’validation’, ’uml’, ’review’, ’pointers’, ’unix’, ’bash’, ’quality’, ’macos’, ’sqlite’, ’r’, ’open’, ’data_structures’, ’java’, ’git’]
14	[’unit_testing’, ’oop’, ’linux’, ’oriented’, ’c’, ’exception’, ’management’, ’programming’, ’devops’, ’python’, ’web_development’, ’dictionary’, ’memory’, ’shell’, ’pypi’, ’numpy’, ’operating’, ’bash’, ’macos’, ’haskell’, ’java’, ’pandas’, ’data_science’, ’debugging’, ’serialization’, ’software_engineering’, ’scripting’, ’computer’, ’emacs’, ’language’, ’list’, ’object’, ’pip’, ’system’, ’science’, ’data_structures’, ’scipy’, ’anaconda’]

15	[’github’, ’oop’, ’server’, ’oriented’, ’google’, ’programming’, ’version’, ’python’, ’mac’, ’c#’, ’sql’, ’javascript’, ’plsql’, ’oracle’, ’jvm’, ’mysql’, ’java’, ’app’, ’engine’, ’mercurial’, ’servlets’, ’database’, ’eclipse’, ’multithreading’, ’software_engineering’, ’computer’, ’sorting’, ’coffeescript’, ’control’, ’security’, ’object’, ’android’, ’maven’, ’mongodb’, ’science’, ’data_structures’, ’git’]
16	[’unit_testing’, ’authorization’, ’redis’, ’eclipse’, ’web_security’, ’c’, ’software_engineering’, ’cookie’, ’computer’, ’interface’, ’oauth’, ’c++’, ’python’, ’web_development’, ’mongoengine’, ’api’, ’sqlalchemy’, ’security’, ’restful_api’, ’flask’, ’c#’, ’user’, ’rust’, ’swagger’, ’restful’, ’caching’, ’jwt’, ’pytest’, ’mysql’, ’http’, ’science’, ’java’, ’csrf’]
17	[’php’, ’server’, ’eclipse’, ’postgresql’, ’open_source’, ’jax_rs’, ’json’, ’database’, ’software_engineering’, ’jpa’, ’computer’, ’rest’, ’wildfly’, ’jackson’, ’ssl’, ’api’, ’glassfish’, ’software_development’, ’maven’, ’reactive’, ’jersey’, ’javascript’, ’sql’, ’mongodb’, ’orm’, ’hibernate’, ’websocket’, ’jvm’, ’restful’, ’latex’, ’http’, ’mysql’, ’science’, ’java’, ’html’]
18	[’php’, ’scala’, ’linux’, ’data_science’, ’functional_programming’, ’perl’, ’nodejs’, ’software_engineering’, ’datascience’, ’encryption’, ’chrome’, ’computer’, ’google’, ’babeljs’, ’utf8’, ’science’, ’extension’, ’unicode’, ’c++’, ’python’, ’web_development’, ’source’, ’c#’, ’javascript’, ’bitcoin’, ’babel’, ’unix’, ’machine_learning’, ’jvm’, ’mysql’, ’r’, ’open’, ’ubuntu’, ’java’, ’html’, ’git’]

19	['github', 'scala', 'jesjs', 'ios', 'cookies', 'ftp', 'capistrano', 'clojure', 'localization', 'gem', 'cloud', 'native', 'cardova', 'ruby', 'rubygems', 'distributed', 'web_development', 'apache', 'rake', 'computing', 'jest', 'vim', 'javascript', 'jquery', 'react', 'kubernetes', 'cms', 'jvm', 'rails', 'mysql', 'reactjs', 'java', 'git']
20	['testing', 'github', 'oop', 'linux', 'web_applications', 'npm', 'oriented', 'programming', 'oauth', 'apache', 'javascript', 'uml', 'hibernate', 'jvm', 'kubernetes', 'java', 'dockerfile', 'cloud', 'intellij', 'visual_studio', 'eclipse', 'nodejs', 'software_engineering', 'docker', 'microservices', 'hibernete', 'amazon', 'sequence', 'rabbitmq', 'angularjs', 'api', 'microservice', 'object', 'maven', 'web_services', 'unix', 'spring', 'framework', 'diagram', 'ubuntu', 'windows']
21	['aspdot_net', 'php', 'arrays', 'html', 'ios', 'application', 'c', 'nodejs', 'multithreading', 'programming', 'ruby', 'python', 'c++', 'android', 'c#', 'sql', 'javascript', 'reactive', 'mobile', 'jquery', 'mysql', 'r', 'java', 'css']
22	['aspdot_net', 'php', 'ios', 'html', 'postgresql', 'database', 'nodejs', 'c', 'google', 'ruby', 'swift', 'c++', 'python', 'angularjs', 'api', 'android', 'c#', 'javascript', 'objective_c', 'web', 'jquery', 'andriod', 'mysql', 'r', 'java', 'css']
23	['aspdot_net', 'php', 'database', 'analysis', 'nodejs', 'c', 'expression', 'ruby', 'swift', 'c++', 'python', 'processing', 'django', 'dot_net', 'c#', 'sql', 'javascript', 'objective_c', 'ajax', 'jquery', 'mysql', 'image', 'r', 'regular', 'java', 'css']

24	[’aspdot_net’, ’php’, ’nodejs’, ’c’, ’algorithms’, ’ruby’, ’swift’, ’c++’, ’python’, ’django’, ’dot_net’, ’c#’, ’javascript’, ’objective_c’, ’ajax’, ’jquery’, ’web_application’, ’mysql’, ’wordpress’, ’r’, ’java’, ’css’]
25	[’aspdot_net’, ’php’, ’scala’, ’linux’, ’postgresql’, ’nodejs’, ’c’, ’hashmap’, ’ruby’, ’swift’, ’c++’, ’python’, ’django’, ’dot_net’, ’c#’, ’javascript’, ’objective_c’, ’ajax’, ’jquery’, ’mysql’, ’r’, ’ubuntu’, ’java’, ’css’]
26	[’aspdot_net’, ’php’, ’visual_studio’, ’functional_programming’, ’nodejs’, ’c’, ’multiprocessing’, ’visual’, ’ruby’, ’swift’, ’c++’, ’python’, ’buffers’, ’django’, ’dot_net’, ’c#’, ’javascript’, ’objective_c’, ’ajax’, ’protocol’, ’jquery’, ’mysql’, ’r’, ’java’, ’css’]
27	[’aspdot_net’, ’php’, ’html’, ’arrays’, ’linux’, ’json’, ’nodejs’, ’c’, ’d3’, ’ruby’, ’swift’, ’c++’, ’python’, ’django’, ’dot_net’, ’c#’, ’javascript’, ’objective_c’, ’ajax’, ’jquery’, ’mysql’, ’r’, ’java’, ’css’, ’multidimensional’]
28	[’aspdot_net’, ’php’, ’ssh’, ’eclipse’, ’linux’, ’postgresql’, ’nodejs’, ’c’, ’ruby’, ’swift’, ’c++’, ’python’, ’haskel’, ’optimization’, ’nix’, ’django’, ’dot_net’, ’c#’, ’sql’, ’javascript’, ’development’, ’objective_c’, ’ajax’, ’environment’, ’jquery’, ’mysql’, ’r’, ’networking’, ’java’, ’css’, ’opengl’]
29	[’aspdot_net’, ’read’, ’php’, ’nodejs’, ’c’, ’programming’, ’ruby’, ’swift’, ’markdown’, ’c++’, ’python’, ’django’, ’dot_net’, ’dir’, ’c#’, ’reactive’, ’javascript’, ’objective_c’, ’ajax’, ’jquery’, ’mysql’, ’r’, ’java’, ’css’]

30	[’aspdot_net’, ’php’, ’coding’, ’postgresql’, ’nodejs’, ’c’, ’ruby’, ’swift’, ’c++’, ’python’, ’django’, ’dot_net’, ’style’, ’c#’, ’javascript’, ’objective_c’, ’ajax’, ’jquery’, ’mysql’, ’webpack’, ’r’, ’java’, ’css’]
31	[’performance’, ’scala’, ’partial’, ’intermediate’, ’programming’, ’algorithms’, ’data’, ’modeling’, ’concurrent’, ’c#’, ’specialization’, ’access’, ’machine_learning’, ’parallel’, ’common’, ’haskell’, ’mapreduce’, ’domain’, ’asynchronous’, ’recursion’, ’monad’, ’visual_studio’, ’matching’, ’functional_programming’, ’data_structures’, ’gui’, ’tail’, ’language’, ’list’, ’async’, ’processing’, ’computing’, ’dot_net’, ’pattren’, ’codedom’, ’scientific’, ’f#’, ’ocaml’, ’visualization’, ’linq’, ’generics’, ’xml’]
32	[’redis’, ’request’, ’json’, ’chrome’, ’erlang’, ’google’, ’python’, ’nosql’, ’javascript’, ’ecmascript’, ’chromium’, ’jquery’, ’websocket’, ’reactjs’, ’cradle’, ’browser’, ’ejavascript’, ’html’, ’css’, ’express’, ’php’, ’nodejs’, ’indexeddb’, ’extensions’, ’jasmine’, ’long’, ’firefox’, ’android’, ’ajax’, ’couchdb’, ’databases’, ’polling’, ’http’, ’networking’, ’xml’]
33	[’elasticsearch’, ’linux’, ’postgresql’, ’meta’, ’json’, ’programming’, ’statistics’, ’data’, ’python’, ’openjdk’, ’marshalling’, ’c#’, ’javascript’, ’websocket’, ’kibana’, ’java’, ’database’, ’asynchronous’, ’aspdot_net’, ’php’, ’visual_studio’, ’nodejs’, ’utf8’, ’regex’, ’docker’, ’math’, ’c++’, ’dot_net’, ’javafx’, ’data_visualization’, ’storage’, ’gzip’, ’linq’, ’git’]
34	[’linux’, ’c’, ’cg’, ’glsl’, ’python’, ’waf’, ’qt’, ’jquery’, ’blender’, ’sockets’, ’model’, ’gtkmm’, ’graphics’, ’multithreading’, ’pyside’, ’boost’, ’c++’, ’component’, ’cmake’, ’object’, ’animation’, ’windows’, ’opengl’]

35	[’text’, ’github’, ’linear’, ’linux’, ’shiny’, ’rstudio’, ’statistics’, ’algebra’, ’jupyter’, ’python’, ’random’, ’apache’, ’igraph’, ’forest’, ’dplyr’, ’machine_learning’, ’caching’, ’latex’, ’bookdown’, ’r’, ’html’, ’css’, ’aptana’, ’mining’, ’knitr’, ’php’, ’readr’, ’data_science’, ’analysis’, ’sapply’, ’matplotlib’, ’clustering’, ’ggplot2’, ’android’, ’unix’, ’visualization’, ’slidify’, ’networking’, ’cluster’, ’git’, ’encoding’]
36	[’testing’, ’unit_testing’, ’intellij’, ’nlp’, ’ios’, ’bitmap’, ’watermark’, ’studio’, ’yuv’, ’ruby’, ’processing’, ’collections’, ’robospice’, ’robotium’, ’android’, ’overlay’, ’objective_c’, ’machine_learning’, ’jruby’, ’video’, ’idea’, ’filter’, ’xcode’, ’sqlite’]
37	[’oop’, ’scala’, ’perl’, ’c’, ’python’, ’go’, ’parsing’, ’c#’, ’javascript’, ’garbage’, ’ecmascript’, ’jquery’, ’web_application’, ’mysql’, ’java’, ’html’, ’css’, ’collection’, ’php’, ’xss’, ’multithreading’, ’nodejs’, ’ruby’, ’c++’, ’node’, ’security’, ’dot_net’, ’android’, ’maven’, ’web_services’, ’ajax’, ’objective_c’, ’synchronization’]
38	[’scala’, ’oauth’, ’rest’, ’gulp’, ’shell’, ’javascript’, ’uml’, ’swagger’, ’search’, ’typescript’, ’haskell’, ’shellscript’, ’cordova’, ’java’, ’express’, ’mocha’, ’php’, ’nodejs’, ’ejb’, ’patterns’, ’docker’, ’node’, ’cassandra’, ’mvc’, ’design’, ’security’, ’android’, ’maven’, ’web_services’, ’chai’, ’elastic’, ’spring’, ’networking’]

39	[’oop’, ’interop’, ’programming’, ’brainfuck’, ’algorithms’, ’aws’, ’python’, ’go’, ’multi’, ’sports’, ’concurrency’, ’c#’, ’unity’, ’memory’, ’garbage’, ’unityscript’, ’jvm’, ’bash’, ’java’, ’collection’, ’asynchronous’, ’threading’, ’kotlin’, ’gradle’, ’multithreading’, ’guava’, ’3d’, ’retrofitting’, ’coroutines’, ’quantitative’, ’anko’, ’android’, ’goroutine’, ’compression’, ’finance’, ’retrofit’, ’mutex’]
40	[’firebase’, ’routing’, ’linux’, ’c’, ’coffeecript’, ’authentication’, ’rest’, ’architecture’, ’go’, ’c#’, ’nosql’, ’javascript’, ’jquery’, ’mysql’, ’sockets’, ’express’, ’html’, ’css’, ’database’, ’angular’, ’asynchronous’, ’java’, ’nodejs’, ’software’, ’docker’, ’jasmine’, ’ruby’, ’angularjs’, ’security’, ’design’, ’expressjs’, ’webpack’, ’git’]
41	[’opencv’, ’git’, ’php’, ’ios’, ’html’, ’linux’, ’json’, ’nodejs’, ’c’, ’docker’, ’ruby’, ’swift’, ’c++’, ’python’, ’ssl’, ’go’, ’angularjs’, ’dot_net’, ’android’, ’c#’, ’shell’, ’javascript’, ’objective_c’, ’unix’, ’jquery’, ’latex’, ’mysql’, ’java’, ’css’, ’opengl’]
42	[’angular’, ’php’, ’scala’, ’json’, ’c’, ’computer’, ’docker’, ’opengl’, ’3d’, ’geojson’, ’aw’, ’amazon’, ’python’, ’matlab’, ’c++’, ’django’, ’shader’, ’android’, ’webgl’, ’c#’, ’web_services’, ’javascript’, ’mongodb’, ’c++ios’, ’unity’, ’jquery’, ’bootstrap’, ’typescript’, ’mysql’, ’macos’, ’r’, ’vision’, ’java’, ’html’, ’cocoa’]
43	[’php’, ’nodejs’, ’perl’, ’google’, ’selenium’, ’docker’, ’maps’, ’analytics’, ’python’, ’apache’, ’angularjs’, ’worker’, ’expressjs’, ’android’, ’c#’, ’sql’, ’javascript’, ’mongodb’, ’ajax’, ’unix’, ’bootstrap’, ’react’, ’typescript’, ’service’, ’cordova’, ’java’]

44	[’aspdot_net’, ’angular’, ’wireshark’, ’php’, ’visual_studio’, ’linux’, ’json’, ’c’, ’dom’, ’scripting’, ’arduino’, ’python’, ’c++’, ’mvc’, ’django’, ’angularjs’, ’dot_net’, ’xml’, ’css’, ’android’, ’c#’, ’sql’, ’javascript’, ’ajax’, ’jquery’, ’bash’, ’mysql’, ’linq’, ’java’, ’html’]
45	[’cuda’, ’oop’, ’scala’, ’c’, ’hig’, ’gcc’, ’python’, ’web_development’, ’software_development’, ’c#’, ’javascript’, ’gnu’, ’g++’, ’templates’, ’latex’, ’xcode’, ’r’, ’java’, ’css’, ’html’, ’editor’, ’pearl’, ’swing’, ’php’, ’sdd’, ’analysis’, ’matplotlib’, ’vb’, ’ruby’, ’c++’, ’dot_net’, ’vim’, ’statistical’, ’objective_c’, ’winforms’, ’linq’, ’vector’]
46	[’ring’, ’mongoose’, ’kotlin’, ’cancan’, ’postgresql’, ’json’, ’clojure’, ’nodejs’, ’regex’, ’gem’, ’ruby’, ’python’, ’db’, ’web_development’, ’rake’, ’beautifulsoap’, ’expressjs’, ’css’, ’javascript’, ’mongodb’, ’koa’, ’bootstrap’, ’factory_bot’, ’co’, ’mongo’, ’html’]
47	[’apple’, ’travis’, ’panda’, ’julia’, ’aquamac’, ’html’, ’elisp’, ’matplotlib’, ’conda’, ’c’, ’encryption’, ’statistics’, ’emacs’, ’python’, ’css’, ’shell’, ’pip’, ’sage’, ’numpy’, ’yaml’, ’gnu’, ’machine_learning’, ’script’, ’ci’, ’latex’, ’visualization’, ’macos’, ’ipython’, ’anaconda’, ’scipy’, ’git’, ’sympy’]
48	[’mongoose’, ’github’, ’html’, ’postgresql’, ’json’, ’npm’, ’nodejs’, ’express’, ’scripting’, ’management’, ’programming’, ’java’, ’cache’, ’couchbase’, ’expressjs’, ’shell’, ’nosql’, ’javascript’, ’mongodb’, ’ajax’, ’ecmascript’, ’couchdb’, ’jquery’, ’react’, ’bash’, ’sha’, ’typescript’, ’sqlite’, ’networking’, ’socket’, ’css’]

49	[’unit_testing’, ’oop’, ’linux’, ’c’, ’management’, ’programming’, ’visual’, ’alias’, ’go’, ’phobos’, ’d’, ’c#’, ’memory’, ’javascript’, ’jquery’, ’static’, ’socket’, ’html’, ’css’, ’cstring’, ’java’, ’php’, ’analysis’, ’multithreading’, ’regex’, ’c++’, ’xlib’, ’dot_net’, ’dmd’, ’pointers’, ’http’]
50	[’linux’, ’html’, ’json’, ’pipelines’, ’centos’, ’google’, ’scripting’, ’java’, ’docker’, ’rest’, ’architecture’, ’python’, ’app’, ’computing’, ’apps’, ’shell’, ’sql’, ’datastore’, ’gnu’, ’swagger’, ’system’, ’kubernetes’, ’bash’, ’engine’, ’css’, ’cloud’]
51	[’unit_testing’, ’oop’, ’valuetuple’, ’string’, ’c#’, ’javascript’, ’pattern’, ’tuples’, ’java’, ’aspdot_net’, ’visual_studio’, ’matching’, ’multithreading’, ’patterns’, ’regex’, ’language’, ’c++’, ’local’, ’optimization’, ’inheritance’, ’dot_net’, ’design’, ’lamba’, ’reshaper’, ’ini’, ’web_services’, ’variables’, ’winforms’, ’generics’, ’windows’, ’xml’]
52	[’udp’, ’performance’, ’arrays’, ’rnn’, ’qtkit’, ’c’, ’opengl’, ’rename’, ’python’, ’buffers’, ’dictionary’, ’numpy’, ’protocol’, ’machine_learning’, ’file’, ’network’, ’smart’, ’batch’, ’buffer’, ’pandas’, ’recurrent’, ’ios’, ’neural’, ’regex’, ’keras’, ’list’, ’c++’, ’tensorflow’, ’cocoa’, ’vectorization’, ’optimization’, ’lstm’, ’quicktime’, ’objective_c’, ’pointers’, ’iphone’, ’anaconda’]
53	[’jsfiddle’, ’ui’, ’arrays’, ’json’, ’chrome’, ’google’, ’string’, ’python’, ’javascriptfiddle’, ’django’, ’javascript’, ’sed’, ’tables’, ’reges’, ’jquery’, ’bash’, ’devtools’, ’backbonejs’, ’mootools’, ’html’, ’css’, ’callback’, ’forms’, ’nodejs’, ’selectors’, ’regex’, ’class’, ’plugins’, ’pseudo’, ’data_structures’, ’setinterval’]

54	[’triangulation’, ’arrays’, ’delaunay’, ’warnings’, ’c’, ’library’, ’multiprocessing’, ’python’, ’mpi’, ’gnuplot’, ’numpy’, ’configuration’, ’broadcasting’, ’parallel’, ’fits’, ’macos’, ’astropy’, ’fitting’, ’read’, ’imaging’, ’pandas’, ’astronomy’, ’setuptools’, ’matplotlib’, ’fortran’, ’basemap’, ’emacs’, ’c++’, ’docs’, ’processing’, ’vtk’, ’pip’, ’fit’, ’curve’, ’ipython’, ’scipy’]
55	[’oop’, ’kernel’, ’linux’, ’queue’, ’perl’, ’c’, ’module’, ’python’, ’struct’, ’select’, ’parsing’, ’pool’, ’config’, ’c#’, ’path’, ’shell’, ’sql’, ’parallel’, ’bash’, ’dbi’, ’r’, ’member’, ’css’, ’database’, ’process’, ’pandas’, ’indexing’, ’boost’, ’markdown’, ’class’, ’int’, ’csv’, ’c++’, ’processing’, ’cs’, ’vaccum’, ’pointers’, ’makefile’, ’sqlite’, ’ubuntu’]
56	[’git’, ’github’, ’yearling’, ’scholar’, ’question’, ’c’, ’debian’, ’programming’, ’commentator’, ’schem’, ’tcl’, ’python’, ’teacher’, ’supporter’, ’type’, ’shell’, ’javascript’, ’large’, ’haskell’, ’student’, ’java’, ’editor’, ’functional’, ’types’, ’dpkg’, ’installer’, ’inference’, ’ruby’, ’c++’, ’repack’, ’svn’, ’annex’, ’repository’, ’reactive’, ’nice’, ’files’, ’singleton’, ’necromancer’, ’curious’]
57	[’redis’, ’packaging’, ’scrapy’, ’perl’, ’json’, ’methods’, ’base’, ’decorator’, ’python’, ’go’, ’getattr’, ’screen’, ’django’, ’qt’, ’sql’, ’setup’, ’web’, ’jquery’, ’nosetest’, ’static’, ’mysql’, ’return’, ’nosetests’, ’value’, ’pyqt’, ’setuptools’, ’regex’, ’matplotlib’, ’metaclass’, ’ruby’, ’list’, ’class’, ’node’, ’scraping’, ’mongodb’, ’function’, ’singleton’, ’crawler’, ’ubuntu’]

58	[’ejabberd’, ’linux’, ’glib’, ’c’, ’microphone’, ’python’, ’pkg’, ’ipc’, ’gdb’, ’config’, ’javascript’, ’xmpp’, ’strophe’, ’com’, ’avatat’, ’php’, ’ios’, ’debugging’, ’errors’, ’linker’, ’pkglconfig’, ’dbus’, ’c++’, ’api’, ’rhythmbox’, ’ubuntu’, ’xmppframework’, ’rythmbox’]
59	[’kafka’, ’scala’, ’assembly’, ’json’, ’sbt’, ’data’, ’python’, ’streaming’, ’apache’, ’packager’, ’sql’, ’javascript’, ’hive’, ’yarn’, ’spark’, ’hadoop’, ’macos’, ’mysql’, ’java’, ’php’, ’eclipse’, ’clojure’, ’play’, ’pipelines’, ’dataset’, ’native’, ’playframework’, ’structured’, ’structure’, ’framework’, ’pyspark’, ’git’]
60	[’geocoding’, ’key’, ’server’, ’openstreesmap’, ’sspi’, ’multiline’, ’python’, ’config’, ’sql’, ’histogram’, ’oracle’, ’rjava’, ’public’, ’word’, ’rodbc’, ’mysql’, ’r’, ’java’, ’aspdot_net’, ’legend’, ’sys’, ’ado’, ’openstreetmap’, ’ggplot2’, ’amazon’, ’security’, ’time’, ’web_services’, ’wrap’, ’syntax’, ’series’, ’newline’, ’git’]
61	[’git’, ’process’, ’ssh’, ’postgresql’, ’linux’, ’zsh’, ’perl’, ’management’, ’aws’, ’pipe’, ’buffers’, ’python’, ’sublime’, ’api’, ’sqlalchemy’, ’django’, ’shell’, ’sql’, ’vim’, ’operating’, ’system’, ’bash’, ’pytest’, ’xcode’, ’windows’, ’css’, ’dwg’]
62	[’aspdot_net’, ’angular’, ’ui’, ’visual_studio’, ’html’, ’nodejs’, ’ip’, ’web_development’, ’mvc’, ’angularjs’, ’restful_api’, ’android’, ’c#’, ’booster’, ’javascript’, ’objective_c’, ’frameworks’, ’jquery’, ’typescript’, ’rxjs’, ’tcp’, ’css’]

63	[’unit_testing’, ’oop’, ’classroom’, ’xdebug’, ’php’, ’eclipse’, ’compiler’, ’closure’, ’nodejs’, ’chrome’, ’google’, ’ruby’, ’rest’, ’cache’, ’cdn’, ’python’, ’applescript’, ’web_development’, ’pp’, ’javascript’, ’iot’, ’frameworks’, ’databases’, ’typescript’, ’mysql’, ’service’, ’worker’, ’css’, ’git’, ’netbeans’]
64	[’aspdot_net’, ’visual_studio’, ’entity’, ’nuget’, ’ios’, ’json’, ’handling’, ’exception’, ’ioc’, ’authentication’, ’windows’, ’web_development’, ’cli’, ’mvc’, ’xml’, ’xamarin’, ’android’, ’mvvm’, ’c#’, ’unity’, ’framework’, ’azure’, ’java’, ’httppost’]
65	[’circle’, ’php’, ’https’, ’ssh’, ’postgresql’, ’c’, ’delphi’, ’aws’, ’zone’, ’ruby’, ’amazon’, ’data’, ’devops’, ’web_development’, ’time’, ’web_services’, ’javascript’, ’postgres’, ’cloudfront’, ’jquery’, ’storage’, ’binary’, ’ci’, ’wordpress’, ’ubuntu’, ’git’]
66	[’code’, ’ssh’, ’linux’, ’deployment’, ’management’, ’docker’, ’devops’, ’ssl’, ’python’, ’plugins’, ’api’, ’logging’, ’circleci’, ’jenkins’, ’maven’, ’junit’, ’xmpp’, ’artifactory’, ’ubuntu’, ’windows’, ’curl’, ’git’, ’tar’]
67	[’github’, ’linux’, ’deployment’, ’c’, ’regex’, ’programming’, ’drone’, ’ruby’, ’gcc’, ’c++’, ’editors’, ’api’, ’qt’, ’css’, ’vim’, ’javascript’, ’sql’, ’g++’, ’web’, ’jquery’, ’html’, ’git’]
68	[’aspdot_net’, ’ios’, ’software’, ’tfs’, ’phone’, ’powershell’, ’ruby’, ’windows’, ’web_development’, ’mvc’, ’silverlight’, ’website’, ’xamarin’, ’android’, ’c#’, ’sql’, ’ajax’, ’platform’, ’developer’, ’nhibernate’, ’web_application’, ’mysql’, ’azure’, ’java’, ’html’, ’git’]

69	['testing', 'aspdot_net', 'ios', 'json', 'google', 'razor', 'ruby', 'swift', 'mvc', 'angularjs', 'mac', 'android', 'css', 'url', 'javascript', 'objective_c', 'services', 'xcode', 'html']
70	['dsl', 'unit_testing', 'udp', 'kafka', 'integration', 'ftp', 'json', 'amqp', 'jdbc', 'serialization', 'soap', 'rest', 'imap', 'sftp', 'ssl', 'dataflow', 'apache', 'jms', 'stream', 'junit', 'httmp', 'spring', 'email', 'sockets', 'tcp', 'java', 'cloud']
71	['pull', 'layout', 'firebase', 'linux', 'ndk', 'tools', 'google', 'management', 'programming', 'oauth', 'data', 'android_sdk', 'string', 'python', 'go', 'shell', 'sdk', 'image', 'algorithm', 'engine', 'database', 'app', 'css', 'java', 'html', 'refresh', 'process', 'php', 'kotlin', 'integration', 'cloud', 'studio', 'c++', 'processing', 'design', 'android', 'android_development', 'data_structures', 'xml']
72	['c', 'programming', 'ghc', 'monads', 'data', 'shake', 'parser', 'computation', 'python', 'analyses', 'django', 'type', 'c#', 'refactoring', 'sql', 'javascript', 'combinators', 'machine_learning', 'reactjs', 'haskell', 'java', 'template', 'documentation', 'html', 'pandas', 'kotlin', 'stack', 'functional_programming', 'multithreading', 'software_engineering', 'inference', 'level', 'extensions', 'studio', 'build', 'optimization', 'android', 'generic', 'dependent', 'system', 'drupal', 'app_development']

73	[’kafka’, ’npm’, ’google’, ’data’, ’applications’, ’python’, ’ssl’, ’django’, ’yaml’, ’javascript’, ’sql’, ’web’, ’machine_learning’, ’jquery’, ’reactjs’, ’mobile’, ’react’, ’rails’, ’hadoop’, ’certificate’, ’algorithm’, ’app’, ’css’, ’engine’, ’html’, ’php’, ’pandas’, ’ios’, ’headers’, ’nodejs’, ’regex’, ’ruby’, ’format’, ’tensorflow’, ’angularjs’, ’jenkins’, ’development’, ’ajax’, ’data_visualization’, ’android_development’, ’http’]
74	[’linux’, ’arrays’, ’deduction’, ’assert’, ’algorithms’, ’data’, ’python’, ’django’, ’type’, ’shell’, ’sql’, ’javascript’, ’numpy’, ’machine_learning’, ’templates’, ’reactjs’, ’bash’, ’static’, ’algorithm’, ’haskell’, ’r’, ’bigquery’, ’java’, ’focus’, ’template’, ’html’, ’cloud’, ’lambda’, ’process’, ’php’, ’problem’, ’nodejs’, ’codeblocks’, ’saas’, ’microservices’, ’ruby’, ’ggplot2’, ’c++’, ’unix’, ’structure’, ’networking’, ’vector’]
75	[’performance’, ’google’, ’data’, ’python’, ’go’, ’joomla’, ’django’, ’xamarin’, ’type’, ’javascript’, ’hive’, ’web’, ’machine_learning’, ’jquery’, ’react’, ’reactjs’, ’hadoop’, ’mysql’, ’algorithm’, ’java’, ’css’, ’html’, ’engine’, ’app’, ’cloud’, ’php’, ’play’, ’saas’, ’tfs’, ’microservices’, ’ruby’, ’interaction’, ’codeigniter’, ’tensorflow’, ’flask’, ’android’, ’ajax’, ’services’, ’cms’, ’data_structures’]
76	[’unit_testing’, ’condenser’, ’git’, ’kafka’, ’kernel’, ’linux’, ’iframe’, ’steam’, ’python’, ’apache’, ’django’, ’sql’, ’javascript’, ’hive’, ’jquery’, ’reactjs’, ’hadoop’, ’xcode’, ’algorithm’, ’r’, ’java’, ’css’, ’cloud’, ’php’, ’eclipse’, ’pandas’, ’data_analysis’, ’httprequest’, ’sparkle’, ’ruby’, ’c++’, ’signature’, ’mvc’, ’jenkins’, ’gpg’, ’android’, ’maven’, ’data_visualization’, ’spring’, ’cocoa’]

77	[’focusing’, ’performance’, ’conversion’, ’management’, ’fcluster’, ’algorithms’, ’gcc’, ’python’, ’go’, ’hierarchical’, ’type’, ’c#’, ’sql’, ’javascript’, ’garbage’, ’numpy’, ’neo4j’, ’machine_learning’, ’size’, ’jquery’, ’reactjs’, ’reduction’, ’technical’, ’mysql’, ’java’, ’database’, ’css’, ’collection’, ’aspdot_net’, ’cloud’, ’data_science’, ’problem’, ’regex’, ’clustering’, ’docker’, ’microservices’, ’amazon’, ’db’, ’tensorflow’, ’api’, ’dot_net’, ’dns’, ’android’, ’web_services’, ’blob’, ’pytorch’, ’issue’, ’sqlite’, ’solving’, ’scipy’]
78	[’web_design’, ’server’, ’linux’, ’google’, ’management’, ’algorithms’, ’data’, ’python’, ’go’, ’apache’, ’type’, ’shell’, ’sql’, ’javascript’, ’numpy’, ’jquery’, ’bash’, ’caching’, ’macos’, ’ide’, ’mysql’, ’bigquery’, ’java’, ’html’, ’database’, ’css’, ’cloud’, ’code’, ’php’, ’visual_studio’, ’jpa’, ’saas’, ’docker’, ’angularjs’, ’pytorch’, ’unix’, ’cms’, ’drupal’, ’data_structures’, ’scipy’, ’git’]
79	[’metadata’, ’postgresql’, ’linux’, ’c’, ’google’, ’module’, ’data’, ’devops’, ’python’, ’apache’, ’django’, ’type’, ’numpy’, ’png’, ’machine_learning’, ’reactjs’, ’bash’, ’kubernetes’, ’mysql’, ’image’, ’engine’, ’java’, ’app’, ’process’, ’php’, ’pil’, ’characters’, ’bitmap’, ’geometry’, ’microservices’, ’line’, ’command’, ’amazon’, ’control’, ’tensorflow’, ’optimization’, ’web_services’, ’unix’, ’zlib’, ’script’, ’cms’, ’storage’, ’variable’, ’data_structures’]

80	['task', 'server', 'request', 'library', 'google', 'management', 'proxy', 'data', 'ansible', 'python', 'django', 'type', 'c#', 'sql', 'javascript', 'web', 'machine_learning', 'parallel', 'reactjs', 'algorithm', 'r', 'network', 'restsharp', 'bigquery', 'database', 'express', 'aspdot_net', 'process', 'visual_studio', 'asp', 'multithreading', 'juniper', 'saas', 'interface', 'cisco', 'mvc', 'api', 'dot_net', 'url', 'development', 'http', 'flurl', 'networking', 'httpclient', 'array', 'encoding']
81	['git', 'cross', 'machine', 'linux', 'c', 'gcc', 'python', 'shell_scripts', 'encoder', 'c#', 'image_processing', 'sql', 'oracle', 'decoder', 'parallel_proccessing', 'java', 'html', 'npapi', 'virtual', 'c++', 'cmake', 'dot_net', 'jenkins', 'platform', 'raspberry_pi', 'gdcm', 'unix', 'oracle_database', 'jpeg', 'visualization', 'matlab', 'ubuntu', 'xml', 'dicom']
82	['git', 'scala', 'meta', 'linux', 'json', 'c', 'programming', 'sbt', 'builder', 'python', 'autoit', 'parsing', 'c#', 'languages', 'javascript', 'jquery', 'search', 'macos', 'mysql', 'hardware', 'php', 'visual_studio', 'functional_programming', 'equations', 'interface', 'c++', 'dot_net', 'polymorphism', 'ajax', 'esoteric', 'unix', 'object_oriented_principles', 'cocoa']
83	['git', 'github', 'travis', 'jupyter', 'python', 'web_development', 'cvxpy', 'image_processing', 'sql', 'numpy', 'javascript', 'continuous', 'machine_learning', 'ci', 'convex', 'css', 'bioinformatics', 'html', 'database', 'graphic', 'pandas', 'scikit', 'integration', 'matplotlib', 'conda', 'equations', 'classification', 'keras', 'learn', 'tensorflow', 'optimization', 'design', 'animation', 'deep_learning', 'visualization', 'matlab', 'scipy', 'anaconda']

84	[’postgresql’, ’linux’, ’nodebox’, ’json’, ’chrome’, ’c’, ’google’, ’full’, ’statistics’, ’doumbek’, ’python’, ’applications’, ’pattern_design’, ’web_development’, ’django’, ’shell’, ’javascript’, ’djembe’, ’jquery’, ’java’, ’html’, ’css’, ’scripts’, ’php’, ’stack’, ’debugging’, ’scripting’, ’art’, ’generative’, ’ajax’, ’bootstrap’, ’http’, ’xml’]
85	[’perl’, ’c’, ’pipeline’, ’tbb’, ’programming’, ’constexpr’, ’python’, ’game’, ’tikz’, ’iterators’, ’javascript’, ’ended’, ’code_reduction’, ’mobile’, ’templates’, ’latex’, ’typescript’, ’clang++’, ’intel’, ’loops’, ’image’, ’open’, ’app’, ’java’, ’html’, ’sequencing’, ’variadic’, ’php’, ’compiler’, ’analysis’, ’resizing’, ’typecasting’, ’software’, ’parameters’, ’containers’, ’c++’, ’inheritance’, ’android’, ’generic’, ’rust’, ’clang’, ’multiple’, ’noexcept’]
86	[’internet’, ’testing’, ’scrapy’, ’json’, ’selenium’, ’python’, ’source’, ’parsing’, ’django’, ’automated’, ’ai’, ’sql’, ’javascript’, ’web’, ’things’, ’programs’, ’open’, ’java’, ’html’, ’code’, ’pandas’, ’beautifulsoup’, ’data_analysis’, ’nodejs’, ’regex’, ’jasmine’, ’angularjs’, ’tests’, ’scraping’, ’mongodb’, ’webdriver’, ’crawler’, ’protractor’, ’xpath’, ’xml’]
87	[’unit_testing’, ’systems’, ’server’, ’linux’, ’json’, ’founder’, ’programming’, ’aws’, ’guitarist’, ’post’, ’c#’, ’sql’, ’javascript’, ’jquery’, ’ec’, ’java’, ’pair’, ’database’, ’cloud’, ’aspdot_net’, ’terraform’, ’nodejs’, ’visual_basic’, ’amazon’, ’distributed’, ’mvc’, ’dot_net’, ’design’, ’web_services’, ’ajax’, ’f#’, ’pascal’, ’rpg’, ’http’, ’azure’, ’database_design’]

88	['text', 'linux', 'cryptography', 'unicode', 'python', 'decoding', 'tkinter', 'memory', 'bytes', 'manipulation', 'cd', 'schematron', 'file', 'netcdf', 'html', 'editor', 'retrieve', 'extracting', 'rom', 'errors', 'multithreading', 'setuptools', 'io', 'elementtree', 'line', 'command', 'logging', 'hex', 'lxml', 'validation', 'unix', 'iso', 'xmlstarlet', 'windows', 'xml', 'encoding']
89	['coding', 'dll', 'ffmpeg', 'plyr', 'grails', 'applications', 'python', 'web_development', 'streaming', 'shell', 'javascript', 'dplyr', 'mobile', 'video', 'typescript', 'macos', 'r', 'java', 'html', 'css', 'aspdot_net', 'experience', 'developement', 'pandas', 'php', 'visual_studio', 'audiovisual', 'regex', 'scripting', 'ruby', 'c++', 'android', 'web_services', 'quality', 'matlab', 'encoding']
90	['github', 'machine', 'linux', 'cryptography', 'tesseract', 'mariadb', 'python', 'web_development', 'apache', 'box', 'sql', 'javascript', 'oracle', 'vmware', 'jquery', 'mobile', 'zend', 'mysql', 'ec', 'css', 'html', 'php', 'virtual', 'amazon', 'c++', 'cmake', 'angularjs', 'dns', 'android', 'development', 'ajax', 'ocr', 'framework', 'git']
91	['elasticsearch', 'scala', 'linux', 'json', 'jdbc', 'grails', 'data', 'spock', 'shell', 'sql', 'javascript', 'hibernate', 'hadoop', 'search', 'ec', 'java', 'curl', 'sit', 'servlets', 'intellij', 'ssh', 'eclipse', 'functional_programming', 'clojure', 'gwt', 'amazon', 'collections', 'android', 'maven', 'powermock', 'web_services', 'elastic', 'spring', 'git']

92	[’oop’, ’chrome’, ’google’, ’game’, ’guice’, ’junit’, ’javascript’, ’jquery’, ’templates’, ’spark’, ’latex’, ’java’, ’html’, ’ceylon’, ’intellij’, ’kotlin’, ’nashorn’, ’pandoc’, ’gdax’, ’ruby’, ’extension’, ’language’, ’inheritance’, ’api’, ’security’, ’android’, ’javadoc’, ’development’, ’celon’, ’libgit’, ’english’, ’learner’, ’idea’, ’akka’]
93	[’linux’, ’library’, ’algorithms’, ’parsing’, ’portable’, ’antlr’, ’graph’, ’c#’, ’sql’, ’algorithm’, ’java’, ’database’, ’aspdot_net’, ’php’, ’loading’, ’multithreading’, ’language’, ’math’, ’class’, ’collections’, ’mono’, ’parsers’, ’dot_net’, ’lazy’, ’generic’, ’completion’, ’cms’, ’databases’, ’issue’, ’auto’, ’jta’, ’xml’, ’synchronization’]
94	[’php’, ’ssh’, ’arrays’, ’linux’, ’json’, ’multithreading’, ’hashmap’, ’regex’, ’io’, ’ruby’, ’string’, ’unicode’, ’jekyll’, ’rubygems’, ’rvm’, ’vagrant’, ’software_development’, ’concurrency’, ’shell’, ’filesystem’, ’unix’, ’bash’, ’sqlite’, ’hash’, ’nokogiri’, ’mac_os’, ’rspec’, ’java’, ’html’, ’git’]
95	[’unit_testing’, ’oop’, ’https’, ’arrays’, ’json’, ’string’, ’apache’, ’parsing’, ’javascript’, ’simplexml’, ’zend’, ’mysql’, ’html’, ’css’, ’php’, ’serialization’, ’regex’, ’dom’, ’patterns’, ’mvc’, ’datetime’, ’object’, ’design’, ’function’, ’document’, ’framework’, ’xpath’, ’xml’]
96	[’symphony’, ’autoloader’, ’php’, ’travis’, ’symfony’, ’silverstripe’, ’graphql’, ’imagick’, ’docker’, ’nginx’, ’aws’, ’twig’, ’apache’, ’javascript’, ’mongodb’, ’composer’, ’ajax’, ’jquery’, ’heroku’, ’fancybox’, ’silex’, ’caching’, ’ci’, ’spring’, ’templates’, ’doctrine’, ’html’, ’git’]

97	['oop', 'components', 'enterprise_architect', 'grammar', 'resx', 'information', 'antlr', 'cyclic', 'graph', 'c#', 'javascript', 'ecmascript', 'dart', 'review', 'web', 'templates', 'algorithm', 'vue', 'java', 'enterprise', 'html', 'architect', 'argument', 'code', 'visual_studio', 'forms', 'software_engineering', 'gwt', 'geopraphical', 'dot_net', 'polymorphism', 'validation', 'vuejs', 'generic', 'designer', 'winforms', 'system', 'form', 'windows']
98	['unit_testing', 'devise', 'cancan', 'server', 'meta', 'arrays', 'active_support', 'sinatra', 'clojure', 'rack', 'json', 'gem', 'software_engineering', 'programming', 'ruby', 'aws', 'elixir', 'fault', 'clojur', 'hacking', 'middleware', 'ember', 'javascript', 'sql', 'emberjs', 'heroku', 'caching', 'routes', 'rspec', 'hash', 'metaprogramming', 'serializers']
99	['argparse', 'cross', 'theory', 'rstudio', 'plyr', 'statistics', 'python', 'mathematical', 'random', 'latex', 'maths', 'algorithm', 'r', 'cvr', 'knitr', 'sparse', 'complexity', 'computational', 'vectors', 'overlapping', 'geometry', 'matrix', 'c++', 'optimization', 'validation', 'dataframe', 'selection', 'faq', 'makefile', 'ubuntu', 'probability', 'git']
100	['server', 'flash', 'safari', 'resolution', 'json', 'npm', 'microphone', 'multi', 'screen', 'streaming', 'canvas', 'javascript', 'sharing', 'web', 'jquery', 'video', 'cordova', 'socket', 'html', 'aspdot_net', 'php', 'webrtc', 'rtc', 'nodejs', 'software_engineering', 'audio', 'opentok', 'node', 'firefox', 'mvc', 'api', 'fault', 'media', 'user', 'record', 'connection']

Table 18: Ground truth human annotations for GitHub users.

Sample ID	**Human Annotation**
1	[’text’, ’coding’, ’entity’, ’programming’, ’python’, ’c#’, ’unity’, ’bert’, ’javascript’, ’machine_learning’, ’react’, ’java’, ’entity_text_matching’, ’html’, ’angular’, ’binary_classification’, ’nlp’, ’angularjavascript’, ’data_science’, ’matching’, ’software_engineering’, ’reactjavascript’, ’ruby’, ’tensorflow’, ’processing’, ’api’, ’lstm’, ’ionic’, ’objective_c’, ’jupyter_notebook’, ’redux’, ’deep_learning’, ’git’]
2	[’coding’, ’ui’, ’linux’, ’npm’, ’chrome’, ’programming’, ’devops’, ’python’, ’web_development’, ’xterm’, ’cabal’, ’jest’, ’shell’, ’homebrew’, ’javascript’, ’hammerspoon’, ’react’, ’typescript’, ’rxjavascript’, ’tmux’, ’haskell’, ’html’, ’zsh’, ’tidy_html’, ’software_engineering’, ’dom’, ’reactjavascript’, ’extension’, ’travis_ci’, ’salt’, ’vim’, ’development’, ’babel’, ’front_end’, ’script’, ’wget’, ’makefile’, ’fedora’, ’vim_script’, ’git’]
3	[’nix_os’, ’scala’, ’oriented’, ’c’, ’programming’, ’sbt’, ’dhall’, ’python’, ’go’, ’apache’, ’shell’, ’sql’, ’homebrew’, ’shell_programming’, ’kubernetes’, ’haskell’, ’mac_os’, ’java’, ’html’, ’css’, ’wireshark’, ’functional_programming’, ’software_engineering’, ’libraries’, ’dhall_lang’, ’hackage’, ’operating_systems’, ’ruby’, ’c++’, ’nix’, ’object’, ’airflow’, ’postgres’, ’helm’, ’postres’, ’morte_library’]

4	[’scala’, ’web_scraping’, ’npm’, ’programming’, ’nestor’, ’go’, ’c#’, ’shell’, ’javascript’, ’mongoosejavascript’, ’expressjavascript’, ’react’, ’search’, ’typescript’, ’vue’, ’java’, ’html’, ’software_engineering’, ’nginx’, ’client’, ’actor’, ’dependency_injection’, ’mongodb’, ’mqtt’, ’wget’, ’crawler’, ’alpakka’, ’http’, ’akka’, ’nodejavascript’]
5	[’journalism’, ’text’, ’web_scriping’, ’programming’, ’statistics’, ’algorithms’, ’data’, ’python’, ’data_extraction’, ’ai’, ’computational_journalism’, ’javascript’, ’extraction’, ’machine_learning’, ’regression’, ’r’, ’d3’, ’html’, ’css’, ’pandas’, ’nlp’, ’data_analysis’, ’data_science’, ’analysis’, ’r_lang’, ’computational’, ’clustering’, ’keras’, ’language’, ’graph_analysis’, ’tensorflow’, ’processing’, ’natural’, ’jupyter_notebook’, ’data_visualization’, ’artificial_intelligence’, ’cluster’, ’git’]
6	[’vorbis’, ’propery_base_testing’, ’bcyrpt’, ’infra’, ’clojure’, ’c’, ’software_engineering’, ’tls’, ’cryptography’, ’programming’, ’open_ssl’, ’powershell’, ’ruby’, ’algorithms’, ’naci’, ’ssl’, ’go’, ’python’, ’security’, ’urllib3’, ’oss_fuzz’, ’shell’, ’homebrew’, ’objective_c’, ’rust’, ’makefile’, ’azure’, ’networking’, ’cpython’, ’java’, ’html’]
7	[’testing’, ’scala’, ’perl’, ’programming’, ’python’, ’go’, ’ethereum’, ’genesis’, ’xamarin’, ’c#’, ’shell’, ’javascript’, ’dropbox’, ’java’, ’html’, ’networking’, ’cloud’, ’smart_contract’, ’distributed_ledger_technology’, ’stone’, ’play’, ’software_engineering’, ’ruby’, ’performance_testing’, ’api’, ’dot_net’, ’lua’, ’azure’, ’nodejavascript’, ’windows’]

8	[’restructuredtext’, ’nanonpb’, ’github’, ’c’, ’software_engineering’, ’smarty’, ’erlang’, ’programming’, ’matplotlib’, ’arduino’, ’data’, ’tex’, ’adruino’, ’python’, ’c++’, ’sublime’, ’css’, ’shell’, ’c#’, ’javascript’, ’numpy’, ’jupyter_notebook’, ’pygae’, ’rust’, ’extraction’, ’ganja_javascript’, ’latex’, ’visualization’, ’lua’, ’matlab’, ’cython’, ’scipy’, ’git’]
9	[’web_design’, ’kobotoolbox’, ’shiny’, ’programming’, ’management’, ’data’, ’technology’, ’inkspace’, ’data_frames’, ’batchfile’, ’stata’, ’shell’, ’javascript’, ’web’, ’machine_learning’, ’mobile’, ’r’, ’documentation’, ’html’, ’collection’, ’rshiny’, ’data_analysis’, ’data_science’, ’exploratory’, ’r_lang’, ’software_engineering’, ’software’, ’scripting’, ’project’, ’pandoc’, ’functions’, ’r_packages’, ’cran’, ’processing’, ’api’, ’computing’, ’pre’, ’statistical’, ’r_modules’, ’visualization’, ’data_processing’, ’packages’, ’education’]
10	[’scala’, ’npm’, ’c’, ’programming’, ’data’, ’python’, ’go’, ’ethereum’, ’sql’, ’javascript’, ’jquery’, ’kubernetes’, ’wordpress’, ’wrapper’, ’java’, ’database’, ’css’, ’cloud’, ’angular’, ’express’, ’terraform’, ’software_engineering’, ’computer’, ’istio’, ’organization’, ’backbone-javascript’, ’ruby’, ’api’, ’google_cloud’, ’mongodb’, ’helm’, ’sass’, ’structure’, ’ethererum’, ’grafana’, ’nodejavascript’, ’git’]
11	[’angular’, ’postgresql’, ’html’, ’conf’, ’c’, ’fortran’, ’nginx’, ’ruby’, ’swift’, ’computation’, ’python’, ’c++’, ’apache’, ’django’, ’shell’, ’image_processing’, ’javascript’, ’objective_c’, ’parallel’, ’sqlite’, ’haskell’, ’java’, ’css’]

12	[’forth’, ’pearl’, ’purescript’, ’linear’, ’cjour’, ’javascripton’, ’clojure’, ’c’, ’perl’, ’lisp’, ’scheme’, ’swift’, ’ruby’, ’python’, ’go’, ’c++’, ’c#’, ’shell’, ’javascript’, ’image_processing’, ’rust’, ’f#’, ’machine_learning’, ’ocaml’, ’common’, ’binary’, ’typescript’, ’coffescript’, ’search’, ’deep_learning’, ’flamegraph’, ’html’]
13	[’mining’, ’php’, ’assembly’, ’perl’, ’c’, ’machine_learning.’, ’ruby’, ’tex’, ’data’, ’matrix’, ’puppet’, ’c++’, ’python’, ’django’, ’cmd’, ’css’, ’shell’, ’javascript’, ’uml’, ’unix’, ’makefile’, ’roff’, ’typescript’, ’latex’, ’quality’, ’diagram’, ’java’, ’html’]
14	[’xslt’, ’pandas’, ’webapplication’, ’linux’, ’r’, ’c’, ’matplotlib’, ’module’, ’functions’, ’tex’, ’ruby’, ’data’, ’emacs_lisp’, ’c++’, ’python’, ’haskel’, ’sceince’, ’apache’, ’structures’, ’css’, ’statistical’, ’javascript’, ’sql’, ’makefiile’, ’jupyter_notebook’, ’machine_learning’, ’deep_learning’, ’haskell’, ’html’]
15	[’compressor’, ’scala’, ’assembly’, ’projects’, ’c’, ’javascriptmin’, ’google’, ’data’, ’dojo’, ’go’, ’apache’, ’javascript’, ’web’, ’spark’, ’typescript’, ’r’, ’cordova’, ’java’, ’css’, ’shrinksafe’, ’blueprint’, ’data_analysis’, ’closure’, ’analysis’, ’exploratory’, ’plotting’, ’shint’, ’uglifyjavascript’, ’api’, ’yui’, ’csslint’, ’mongodb’, ’objective_c’, ’variables’, ’opensource’, ’fyrevm’]
16	[’oauth2’, ’ui’, ’linux’, ’javascripton’, ’openapi’, ’c’, ’lab’, ’fastapi’, ’ruby’, ’python’, ’c++’, ’elixir’, ’app’, ’image_processing’, ’sql’, ’rust’, ’dynamodb’, ’swagger’, ’jwt’, ’weechat’, ’javascipt’, ’sdl’, ’haskell’, ’java’, ’css’, ’git’]

17	[’angular’, ’php’, ’scala’, ’linux’, ’html’, ’c’, ’graphql’, ’jax’, ’apache’, ’api’, ’security’, ’tooltip’, ’rs’, ’sql’, ’javascript’, ’boot’, ’jquery’, ’web_application’, ’spring’, ’typescript’, ’ide’, ’http’, ’framework’, ’nodejavascript’, ’java’, ’css’]
18	[’development’, ’javascript’, ’python’, ’game’, ’html’]
19	[’php’, ’scala’, ’clojure’, ’c’, ’smarty’, ’obective’, ’ruby’, ’swift’, ’python’, ’go’, ’c++’, ’django’, ’css’, ’type’, ’shell’, ’javascript’, ’script’, ’typescript’, ’haskell’, ’vim_script’, ’java’, ’html’, ’dockerfile’]
20	[’git’, ’code’, ’php’, ’angularjavascript’, ’c’, ’coffeescript’, ’algorithms’, ’apacheconf’, ’spaced’, ’web_development’, ’objective’, ’css’, ’c#’, ’sql’, ’development’, ’uml’, ’repeitition’, ’review’, ’typescripct’, ’vue’, ’nodejavascript’, ’cybernetics’, ’java’, ’html’, ’cloud’]
21	[’ui’, ’travis’, ’okhttp’, ’management’, ’rest’, ’scrolling’, ’web_development’, ’decoding’, ’proguard’, ’communication’, ’ux’, ’tdd’, ’javascript’, ’javascriptp’, ’web’, ’caching’, ’bluetooth’, ’image’, ’java’, ’html’, ’j2ee’, ’kotlin’, ’loading’, ’gradle’, ’bar’, ’javascripton’, ’webscraping’, ’rxjava’, ’volley’, ’smooth’, ’api’, ’design’, ’media’, ’android’, ’maven’, ’action’, ’email’, ’scraper’, ’spring’, ’mobility’, ’xml’]
22	[’redis’, ’semantic’, ’grpc’, ’sinatra’, ’electron’, ’mailchimp’, ’rest’, ’ssl’, ’python’, ’go’, ’versioning’, ’guard’, ’shell’, ’javascript’, ’imagemagick’, ’dokku’, ’kubernetes’, ’react’, ’rails’, ’mysql’, ’app’, ’html’, ’software_design’, ’javascripton’, ’docker’, ’automation’, ’intelligence’, ’ruby’, ’couchbase’, ’objective_c’, ’paas’, ’redux’, ’xml’, ’git’]

23	[’gloss’, ’ui’, ’evolving’, ’c’, ’tracing’, ’game’, ’web_development’, ’box’, ’canvas’, ’webgl’, ’shell’, ’networks’, ’javascript’, ’path’, ’ux’, ’typescript’, ’haskell’, ’d3’, ’css’, ’html’, ’database’, ’php’, ’ios’, ’neural’, ’cornell’, ’ascii’, ’less’, ’ruby’, ’backbonejavascript’, ’c++’, ’bidirectional’, ’objective_c’, ’jade’, ’nodejavascript’, ’git’]
24	[’testing’, ’commerce’, ’edd’, ’npm’, ’beaver’, ’glsl’, ’google’, ’opengl’, ’builder’, ’seo’, ’rest’, ’analytics’, ’game’, ’unity3d’, ’cors’, ’vagrant’, ’social’, ’shell’, ’javascript’, ’jquery’, ’react’, ’wordpress’, ’split’, ’css’, ’html’, ’php’, ’themer’, ’kotlin’, ’google_analytics’, ’digital’, ’e’, ’marketing’, ’csv’, ’oops’, ’popup’, ’media’, ’ajax’, ’databases’, ’tsql’, ’maker’, ’optimize’]
25	[’postgresql’, ’rpc’, ’application’, ’c’, ’encryption’, ’rest’, ’smtp’, ’imap’, ’python’, ’ethereum’, ’gnuplot’, ’shell’, ’networks’, ’javascript’, ’websocket’, ’yarn’, ’binary’, ’typescript’, ’vuepress’, ’certificate’, ’html’, ’php’, ’javascripton’, ’serialization’, ’docker’, ’interface’, ’scgi’, ’websockets’, ’c++’, ’api’, ’hacking’, ’rust’, ’mqtt’, ’clang’, ’nodejavascript’]
26	[’midi’, ’filtering’, ’ui’, ’server’, ’c’, ’rad’, ’music’, ’maps’, ’emacs_lisp’, ’python’, ’epub2’, ’shell’, ’javascript’, ’epub3’, ’bibliopixel’, ’websocket’, ’bash’, ’file’, ’java’, ’max’, ’pixelweb’, ’css’, ’sort’, ’utf8’, ’io’, ’c++’, ’api’, ’parsers’, ’hacking’, ’animation’, ’bootstrap’, ’kivy’, ’lua’, ’nodejavascript’, ’cython’, ’git’, ’encoding’]

27	['git', 'scala', 'aws', 'emacs_lisp', 'python', 'game', 'go', 'web_development', 'xhtml', 'shell', 'sql', 'javascript', 'react', 'regular', 'rails', 'java', 'html', 'php', 'angularjavascript', 'coffee', 'coffeescript', 'ruby', 'emacs', 'cassandra', 'expressions', 'web_services', 'ajax', 'script', 'storm', 'artificial_intelligence', 'vision', 'xml', 'leiningen']
28	['pearl', 'actionscript', 'php', 'assembly', 'html', 'perl', 'c', 'clojure', 'powershell', 'ruby', 'tex', 'hml', 'c++', 'python', 'go', 'nix', 'django', 'xml', 'haskell', 'shell', 'c#', 'openscad', 'jupyter_notebook', 'objective_c', 'sql', 'rust', 'subversion', 'mysql', 'r', 'java', 'css']
29	['performance', 'scala', 'stencil', 'workers', 'components', 'python', 'animations', 'clojurescript', 'apps', 'charts', 'javascript', 'responsive', 'profiling', 'web', 'progressive', 'react', 'static', 'rxjavascript', 'typescript', 'service', 'shellscript', 'java', 'html', 'css', 'es6', 'functional_programming', 'clojure', 'reactjavascript', 'coffeescript', 'netlify', 'flame', 'ruby', 'tex', 'jekyll', 'c++', 'design', 'vim', 'animation', 'rust', 'sites', 'lua', 'webpack', 'nodejavascript']
30	['performance', 'postgresql', 'linux', 'c', 'pipeline', 'management', 'shiro', 'grails', 'python', 'javascript', 'jruby', 'jquery', 'react', 'rails', 'javscript', 'mysql', 'solr', 'rspec', 'java', 'html', 'css', 'asset', 'es6', 'asp', 'knockout_javascript', 'coffeescript', 'docker', 'nginx', 'ruby', 'cache', 'rabbitmq', 'groovy', 'system', 'mail', 'vim_script', 'git']

31	[’testing’, ’aspdot_net’, ’notebook’, ’data_science’, ’functional_programming’, ’web_applications’, ’c’, ’time_series’, ’clustering’, ’giraffe’, ’modelling’, ’visual’, ’powershell’, ’web_dev’, ’tex’, ’python’, ’dot_net’, ’css’, ’haskell’, ’ai’, ’c#’, ’shell’, ’javascript’, ’data_exploration’, ’histogram’, ’jupyter_notebook’, ’f#’, ’machine_learning’, ’typescript’, ’azure’, ’r’, ’html’]
32	[’mocha’, ’php’, ’javascripton’, ’vanillajavascript’, ’indexeddb’, ’graph_ql’, ’web_dev’, ’dep_javascript’, ’c++’, ’ui_testing’, ’cli’, ’terminal’, ’shell’, ’yaml’, ’javascript’, ’development’, ’mongodb’, ’couchdb’, ’apollo’, ’react’, ’parallel_processing’, ’redux’, ’jquery’, ’typescript’, ’nodejavascript’, ’preact’, ’dockerfile’, ’html’]
33	[’php’, ’html’, ’postgresql’, ’mediawiki’, ’search_engine’, ’programming’, ’docker’, ’ruby’, ’devops’, ’python’, ’go’, ’c++’, ’memory_optimization’, ’shell’, ’c#’, ’javascript’, ’development’, ’plsql’, ’restful’, ’plpgsql’, ’visualization’, ’tileserver’, ’typescript’, ’lua’, ’kibana’, ’nodejavascript’, ’java’, ’css’, ’dockerfile’]
34	[’game_developer’, ’linux’, ’images’, ’camera’, ’c’, ’debian’, ’scripting’, ’automation’, ’kerbal’, ’c++’, ’python’, ’go’, ’distrbution_computing’, ’cap2re’, ’c#’, ’javascript’, ’unix’, ’developer’, ’feature_engineering’, ’backend’, ’opengl’]
35	[’lasso’, ’git’, ’linux’, ’html’, ’perl’, ’pipelines’, ’c’, ’feature’, ’tex’, ’python’, ’c++’, ’tensorflow’, ’(exploratory_data_analysis)’, ’ruffus’, ’shell’, ’jupyter_notebook’, ’gnu’, ’eda_’, ’machine_learning’, ’neural_networking’, ’selection’, ’jquery’, ’neural_style_transfer’, ’shell_script’, ’lua’, ’r’, ’vim_script’, ’css’, ’bioinformatics’]

36	[’rest_api’, ’nlp’, ’multithreading’, ’dns_ttl’, ’software_engineering’, ’apiape’, ’graphql’, ’java’, ’hapi’, ’ruby’, ’python’, ’web_development’, ’graphql_server’, ’client_server’, ’javascript’, ’koa’, ’apollo’, ’rails’, ’typescript’, ’nodejavascript’, ’express’, ’css’]
37	[’php’, ’html’, ’web_security’, ’web_applications’, ’npm’, ’javascripton’, ’c’, ’reactjavascript’, ’ruby’, ’whatwg’, ’python’, ’go’, ’web_development’, ’apache’, ’security’, ’w3c’, ’sql’, ’javascript’, ’cache_handling’, ’ecmascript’, ’jquery’, ’dom_xss’, ’bash_scripting’, ’ocaml’, ’typescript’, ’nodejavascript’, ’shell_scripting’, ’java’, ’css’, ’(10)’]
38	[’html’, ’zookeeper’, ’clojure’, ’c’, ’erlang’, ’coffeescript’, ’docker’, ’soap’, ’markdown’, ’ruby’, ’go’, ’firmware’, ’management_system’, ’opescad’, ’c#’, ’javascript’, ’tslint’, ’openscad’, ’uml’, ’makefile’, ’gitlab’, ’typescript’, ’vue’, ’nodejavascript’, ’java’, ’css’, ’git’]
39	[’unit_testing’, ’kotlin’, ’sports_programming’, ’c’, ’debian’, ’programming’, ’arduino’, ’automation’, ’java’, ’c++’, ’python’, ’web_development’, ’mocking_framework’, ’sports’, ’software_development’, ’javascript’, ’jvm’, ’icpc’, ’embedded_software’, ’package_builder’, ’serealization’, ’algorithm’, ’command_line’, ’deadlock_detection’, ’backend_development’, ’xml’]
40	[’php’, ’linux’, ’c’, ’boosterjavascript’, ’operating_system’, ’gcp’, ’cloud_computing’, ’aws’, ’powershell’, ’ruby’, ’python’, ’openstack’, ’web_development’, ’go’, ’calico’, ’javascripttree’, ’javascriptormdb’, ’dot_net’, ’dockerruby’, ’shell’, ’javascriptmx’, ’javascript’, ’iot’, ’kubernetes’, ’makefile’, ’azure’, ’nodejavascript’, ’dockerfile’, ’html’]

41	[”, ’linux’, ’signal’, ’sphinx’, ’version’, ’python’, ’virtualization’, ’pypack’, ’javascript’, ’toolkit’, ’gitless’, ’pyparsing’, ’indicator’, ’browntruck’, ’prompt’, ’viewer’, ’mercurial’, ’pysdl2’, ’patch’, ’interactive_latencies’, ’php’, ’boto3’, ’setuptools’, ’docker’, ’ruby’, ’parallel_computing’, ’snapcraft’, ’vnc’, ’c++’, ’control’, ’warehouse’, ’workstation’, ’processing’, ’hashin’, ’pip’, ’pseps’, ’http’, ’ctypesgen’, ’nodejavascript’, ’windows’, ’dos2unix’, ’git’, ’encoding’]
42	[”, ’highload’, ’gamejam’, ’demo’, ’maps’, ’yandex’, ’bla’, ’fountain’, ’python’, ’20151102’, ’shri’, ’webgl’, ’canvas’, ’shell’, ’javascript’, ’baby’, ’definitelytyped’, ’s3tc’, ’jquery’, ’typescript’, ’html’, ’publicsuffixlist’, ’angular’, ’enb’, ’debugging’, ’javascripton’, ’msk’, ’computer’, ’extensions’, ’exec’, ’c++’, ’foopish’, ’firefox’, ’api’, ’bem’, ’vision’, ’android’, ’subbotnik’, ’vow’, ’20151017’, ’stylus’, ’videom’, ’nodejavascript’, ’opengl’]
43	[’content’, ’google’, ’popover’, ’mapbox’, ’maps’, ’policy’, ’python’, ’gl’, ’apps’, ’calendar’, ’javascript’, ’web’, ’progressive’, ’kubernetes’, ’pug’, ’test’, ’polyline’, ’tileserver’, ’service’, ’bromba’, ’confirmation’, ’express’, ’css’, ’html’, ’worker’, ’php’, ’javascripton’, ’codemod’, ’zsyp’, ’docker’, ’interface’, ’ruby’, ’cache’, ’c++’, ’async’, ’getlet’, ’control’, ’postal’, ’api’, ’antiscroll’, ’tile’, ’exorcist’, ’security’, ’helm’, ’tip’, ’dialog’, ’sqlite’, ’nodejavascript’, ’git’]

44	[’graohics’, ’stm32f103xx’, ’c’, ’google’, ’paths’, ’maps’, ’python’, ’private’, ’canvas’, ’mfrc522’, ’hid’, ’reader’, ’javascript’, ’sql’, ’device’, ’threadmark’, ’hacklab’, ’bash’, ’mysql’, ’rfid’, ’hal’, ’usb’, ’embedded’, ’java’, ’css’, ’database’, ’html’, ’aspdot_net’, ’php’, ’m’, ’javascripton’, ’selectors’, ’cortex’, ’regex’, ’sh1106’, ’dwprog’, ’c++’, ’website’, ’dot_net’, ’serial’, ’design’, ’usbd’, ’rust’, ’rimoio2’, ’xml’, ’casterm’]
45	[”, ’oop’, ’huffman’, ’linux’, ’modules’, ’jelly’, ’decorator’, ’python’, ’shell’, ’c#’, ’javascript’, ’latex’, ’tmux’, ’r’, ’java’, ’css’, ’html’, ’pkgload’, ’php’, ’amplimap’, ’regex’, ’nextflow’, ’tree’, ’vi’, ’tex’, ’alterpisode’, ’ruby’, ’math’, ’c++’, ’multifuction’, ’optimization’, ’dot_net’, ’vim’, ’jupyter_notebook’, ’unix’, ’files’, ’gzip’, ’royal’, ’vim_script’, ’snakemake’]
46	[”, ’blocky’, ’scala’, ’postgresql’, ’ffmpeg’, ’demo’, ’python’, ’guilded’, ’generator’, ’icejaw’, ’javascript’, ’bouncer’, ’koa’, ’react’, ’pug’, ’h.264’, ’typescript’, ’sortedset’, ’mysql’, ’browser’, ’java’, ’html’, ’template’, ’css’, ’minimal’, ’window’, ’skeleton’, ’javascripton’, ’regex’, ’swift’, ’ruby’, ’libx264’, ’elm’, ’mongodb’, ’render’, ’bootstrap’, ’twitter’, ’gulid’, ’sqlite’, ’schnorr’, ’nodejavascript’]

47	[’bot’, ’git’, ’systems’, ’linear’, ’blog’, ’symbolic’, ’brown’, ’removestar’, ’statistics’, ’sphinx’, ’version’, ’algebra’, ’computation’, ’emacs_lisp’, ’python’, ’mypython’, ’numpy’, ’dollar’, ’toolkit’, ’dotfiles’, ’bash’, ’latex’, ’1’, ’pyflyby’, ’macos’, ’prompt’, ’anaconda’, ’html’, ’sympy’, ’pandas’, ’labs’, ’matplotlib’, ’jupter_console’, ’computer’, ’numba’, ’emacs’, ’math’, ’quansight’, ’c++’, ’control’, ’pip’, ’jupyter_notebook’, ’staged’, ’workflow’, ’recipes’, ’pudb’, ’water’, ’scipy’, ’site’]
48	[’31’, ’dirname’, ’books’, ’warsawjavascript’, ’npm’, ’programming’, ’version’, ’ts’, ’new’, ’timber’, ’calendar’, ’canvas’, ’shell’, ’nosql’, ’javascript’, ’mjml’, ’jquery’, ’bash’, ’typescript’, ’workshop’, ’cmi’, ’socket’, ’css’, ’html’, ’scripts’, ’ios’, ’javascripton’, ’functional_programming’, ’no’, ’registry’, ’amazon’, ’ntd’, ’node’, ’dps’, ’deno’, ’control’, ’ntnd’, ’api’, ’free’, ’web_services’, ’mongodb’, ’puppeteer’, ’couchdb’, ’doxdox’, ’youtube’, ’hello’, ’nodejavascript’, ’git’]
49	[’braven_2’, ’oop’, ’server’, ’unit’, ’assembly’, ’linux’, ’c’, ’library’, ’kits’, ’visual’, ’struct’, ’taskbar’, ’phobos’, ’d’, ’canvas’, ’c#’, ’javascript’, ’beyondz’, ’sockets’, ’java’, ’css’, ’html’, ’rubycas’, ’php’, ’selectors’, ’regex’, ’juno’, ’threaded’, ’arsd’, ’ruby’, ’lms’, ’interface’, ’class’, ’c++’, ’libclang’, ’terminal’, ’dot_net’, ’ldc’, ’user’, ’dpp’, ’platform’, ’adrdox’, ’emulator’, ’sqlite’, ’windows’]

50	[’kubia’, ’jboss’, ’images’, ’linux’, ’cni’, ’centos’, ’example’, ’tools’, ’google’, ’rintime’, ’maistra’, ’istion’, ’go’, ’rpms’, ’shell’, ’javascript’, ’datastore’, ’web’, ’dotfiles’, ’swagger’, ’kubernetes’, ’2ed’, ’kubectl’, ’engine’, ’html’, ’in’, ’rback’, ’operator’, ’app’, ’cloud’, ’controller’, ’javascripton’, ’istio’, ’docker’, ’plugins’, ’api’, ’cdi’, ’website’, ’dot’, ’centos7’, ’javascriptonschema’, ’unix’, ’action’, ’kind’, ’2.0’]
51	[’unit_testing’, ’actionscript’, ’visual_studio’, ’nuget’, ’matching’, ’application’, ’discriminated’, ’basic’, ’programming’, ’visual’, ’statistics’, ’web_dev’, ’async’, ’node’, ’sublime’, ’dot_net’, ’xml’, ’c#’, ’javascript’, ’development’, ’unions’, ’pattern’, ’windows’, ’html’, ’abstraction’]
52	[’machine’, ’integration’, ’neural’, ’vectors’, ’c’, ’word2vec’, ’gui’, ’tex’, ’jupyter’, ’trees’, ’python’, ’c++’, ’tensorflow’, ’embeddings’, ’vision’, ’javascript’, ’numpy’, ’carlo’, ’machine_learning’, ’pointers’, ’unsupervised’, ’word’, ’deep_learning’, ’visualization’, ’artificial_intelligence’, ’monte’, ’tensor’, ’wrappers’, ’network’, ’flow’, ’opengl’]
53	[’clojure’, ’web_dev’, ’tex’, ’devops’, ’reddit’, ’sublime’, ’node’, ’python’, ’go’, ’vim_script’, ’configs’, ’css’, ’shell’, ’vim’, ’javascript’, ’gnu’, ’web’, ’machine_learning’, ’bootstrap’, ’wrapper’, ’html’]
54	[’testing’, ’scipy’, ’data_analysis’, ’html’, ’c’, ’matplotlib’, ’conda’, ’ascii’, ’sphinx’, ’web_dev’, ’visualizations’, ’jupyter’, ’python’, ’pypi’, ’pip’, ’javascript’, ’histogram’, ’plotly’, ’visualization’, ’azure’, ’css’, ’git’]

55	[’php’, ’linux’, ’books’, ’perl’, ’c’, ’pipelines’, ’conda’, ’note’, ’management’, ’web_dev’, ’ruby’, ’jupyter’, ’devops’, ’python’, ’c++’, ’decoding’, ’simulation’, ’shell’, ’c#’, ’javascript’, ’web_services’, ’objective_c’, ’memory’, ’pypi’, ’binary’, ’typescript’, ’azure’, ’mac_os’, ’java’, ’xml’, ’encoding’]
56	[’ui’, ’layout’, ’application’, ’gui’, ’dom’, ’web_dev’, ’haskel’, ’sublime’, ’wiki’, ’api’, ’css’, ’development’, ’javascript’, ’frameworks’, ’web’, ’websocket’, ’visualization’, ’haskell’, ’browser’, ’documentation’, ’html’]
57	[’linux’, ’html’, ’rpc’, ’database’, ’chrome’, ’c’, ’google’, ’management’, ’programming’, ’soap’, ’python’, ’c++’, ’pypy’, ’django’, ’mvc’, ’api’, ’object’, ’xml’, ’memory’, ’sql’, ’relationship’, ’javascript’, ’kubuntu’, ’mongodb’, ’os’, ’web’, ’network’, ’ubuntu’, ’socket’, ’sandbox’]
58	[’php’, ’linux’, ’c’, ’gui’, ’skype’, ’io’, ’docker’, ’devops’, ’python’, ’go’, ’bit’, ’c++’, ’css’, ’shell’, ’javascript’, ’os’, ’gnu’, ’web’, ’torrent’, ’twitter’, ’binary’, ’nodejavascript’, ’wrapper’, ’mac_os’, ’vim_script’, ’java’, ’html’, ’git’]
59	[’kafka’, ’scala’, ’compiler’, ’splicesql’, ’html’, ’engineering’, ’chain’, ’knn’, ’data’, ’analytics’, ’applications’, ’tensorflow’, ’bigtable’, ’plugin’, ’apache’, ’streaming’, ’block’, ’ingestion’, ’c#’, ’sql’, ’javascript’, ’f#’, ’jvm’, ’spark’, ’algorithm’, ’css’]

60	[’github’, ’data_analysis’, ’database’, ’javascripton’, ’yahoo’, ’google’, ’statistics’, ’interface’, ’maps’, ’tex’, ’analytics’, ’python’, ’parsing’, ’opencpu’, ’bayesdb’, ’sql’, ’javascript’, ’bayesian’, ’gnu’, ’machine_learning’, ’formatting’, ’visualization’, ’sqlite’, ’r’, ’html’]
61	[’testing’, ’software_design’, ’cross’, ’data_analysis’, ’ios’, ’linux’, ’debugging’, ’web_applications’, ’database’, ’fuzzy_search’, ’data_process’, ’coffeescript’, ’metadata_files’, ’tex’, ’platform_applications’, ’ruby’, ’ruby_design’, ’python’, ’go’, ’csv_process’, ’django’, ’communication’, ’cloud_infrastructure_management’, ’viml’, ’javascript’, ’vim’, ’orm’, ’sql’, ’pytorch’, ’data_visualization’, ’android_development’, ’mogo’, ’mysql’, ’vim_script’, ’end_to_end_testing’, ’html’, ’collaboration’]
62	[’large_dynamic_graphs’, ’oop’, ’doubly_linked_list’, ’html’, ’front_end_libraries’, ’d3’, ’data_structure’, ’rwd’, ’algorithms’, ’data’, ’python’, ’tree_structure’, ’core_maintenance’, ’structures’, ’curriculum’, ’order_based_approach’, ’javascript’, ’graphs’, ’data_visualization’, ’react’, ’k_core_decomposition’, ’nodejavascript’, ’css’]
63	[’web_design’, ’c’, ’tools’, ’hyper’, ’go’, ’web_development’, ’apache’, ’shell’, ’javascript’, ’amp’, ’jruby’, ’typescript’, ’wordpress’, ’java’, ’css’, ’html’, ’documentation’, ’networking’, ’php’, ’media_packaging’, ’dom’, ’shel’, ’ruby’, ’c++’, ’http_proxy’, ’web_push_library’, ’node’, ’api’, ’url_validation’, ’front_end’, ’api_applications’, ’script’, ’cms’, ’automation_dependency_update’, ’android_development’, ’rubinius’, ’nodejavascript’, ’xml’]

64	[’web_design’, ’c’, ’algorithms’, ’android_sdk’, ’data’, ’xamarin’, ’power’, ’c#’, ’unity’, ’javascript’, ’shell’, ’data_management’, ’java’, ’css’, ’html’, ’documentation’, ’software_design’, ’visual_studio’, ’mvvmcross’, ’eclipse’, ’ios’, ’vb’, ’powershell’, ’swift’, ’plugins’, ’structures’, ’dot_net’, ’maven’, ’objective_c’, ’game_development’, ’front_end’, ’android_development’, ’model_view_viewmodel_framework’, ’app_development’]
65	[’site_generator’, ’postgresql’, ’linux’, ’perl’, ’rest’, ’python’, ’go’, ’plugin’, ’apache’, ’vim_script’, ’wordpress_plugins’, ’shell’, ’c#’, ’javascript’, ’rails’, ’java’, ’css’, ’html’, ’cli_utility’, ’php’, ’apis’, ’ruby’, ’interface’, ’http_libraries’, ’vim’, ’development’, ’data_search’, ’bootstrap’, ’paas’, ’heroku’, ’theme_design’, ’ubuntu’, ’xml’, ’git’]
66	[’ui’, ’linux’, ’images’, ’books’, ’perl’, ’c’, ’data_process’, ’proxy’, ’blogs’, ’python’, ’go’, ’shell’, ’c#’, ’javascript’, ’web’, ’maintaining’, ’creating’, ’java’, ’css’, ’html’, ’dockerfile’, ’image_process’, ’software’, ’libraries’, ’docker’, ’ruby’, ’tex’, ’c++’, ’groovy’, ’writing’, ’development’, ’makefile’, ’agile’]
67	[’angular’, ’ui’, ’php’, ’rails_api’, ’angularjavascript’, ’html’, ’rubygem’, ’npm’, ’coffeescript’, ’java’, ’ruby’, ’python’, ’go’, ’cli’, ’web_development’, ’app’, ’plugin’, ’c++’, ’shell’, ’viml’, ’javascript’, ’vim’, ’development’, ’pytorch’, ’map_api’, ’jquery’, ’web_frameworks’, ’rails’, ’typescript’, ’vim_script’, ’rubocop_plugins’, ’css’, ’git’]

68	[’testing’, ’mock_api’, ’material_design’, ’ui_design’, ’application’, ’debian’, ’corefx’, ’animations’, ’apache’, ’android_data_binding’, ’c#’, ’sql’, ’file’, ’mock’, ’java’, ’spock_tests’, ’database’, ’console’, ’software_design’, ’php’, ’ios’, ’debugging’, ’java_sdk’, ’bindingcollectionadapter’, ’cmd’, ’api’, ’dot_net’, ’opensuse’, ’app_development’, ’android’, ’development’, ’retrofit’, ’system’, ’android_development’, ’layout_design’, ’ubuntu’, ’windows’, ’xml’]
69	[’ios_development’, ’php’, ’testing_activity’, ’julia’, ’javascripton’, ’ios’, ’c’, ’fortran’, ’coffeescript’, ’ruby’, ’swift’, ’tex’, ’c++’, ’python’, ’web_development’, ’android’, ’javascript’, ’objective_c’, ’makefile’, ’rails’, ’mysql’, ’keychain_security’, ’r’, ’app_development’, ’java’, ’html’, ’image_compressor’]
70	[’google_cloud_platform_apis’, ’programming’, ’apatche’, ’salesforce’, ’web_development’, ’apache’, ’shell’, ’eclips’, ’javascript’, ’jira’, ’jvm’, ’data_flow’, ’java’, ’css’, ’cloud’, ’xslt’, ’data_analysis’, ’spring_extensions’, ’amqp’, ’rubygem’, ’ruby’, ’spring_integration’, ’rabbitmq’, ’groovy’, ’api’, ’maven’, ’data_integration’, ’mongodb’, ’development’, ’client/server’, ’cloud_stream’, ’spring’, ’gemfire’, ’xml’, ’graph_building’]

71	[’bot’, ’server’, ’linux’, ’c’, ’google’, ’programming’, ’mmonkey’, ’applications’, ’python’, ’web_development’, ’go’, ’source’, ’javascript’, ’datastore’, ’mobile’, ’search_engine_building’, ’gpu’, ’open’, ’java’, ’multimedia’, ’interpreter’, ’css’, ’html’, ’data_generation’, ’php’, ’kotlin’, ’text_proccessing’, ’keras’, ’language’, ’math’, ’integrating_with_3rd_party_applications’, ’c++’, ’client’, ’mobile_development’, ’android’, ’gopher’, ’telegram’, ’http’, ’graphic_design’, ’asyncronous_computing’]
72	[’web_applications’, ’adga’, ’c’, ’markup_languages’, ’lisp’, ’micro’, ’arduino’, ’8’, ’fpga’, ’emulators’, ’python’, ’clash’, ’language_translation’, ’avr’, ’cpu’, ’simulator’, ’javascript’, ’common’, ’intel’, ’simavr’, ’haskell’, ’lava’, ’html’, ’interpreter’, ’virtual_machine’, ’brainfuck(language)’, ’controller’, ’llvm’, ’readability’, ’microprocessor’, ’chip’, ’tex’, ’c++’, ’design’, ’tests’, ’rust’, ’kansas’, ’8080’, ’emulator’, ’circuit’]
73	[’angular’, ’php’, ’nlp’, ’html’, ’ui_design’, ’perl’, ’c’, ’npm’, ’coffeescript’, ’ruby’, ’web_dev’, ’python’, ’c++’, ’elm’, ’jekyll’, ’dot_net’, ’testbed’, ’c#’, ’sql’, ’javascript’, ’code_styling’, ’objective_c’, ’jquery’, ’react’, ’sass’, ’code_formatting’, ’web_developer’, ’app’, ’css’, ’bookmarklet’]

74	[’testing’, ’lambda_equations’, ’data_manipulation’, ’c’, ’racket’, ’statistics’, ’algorithms’, ’data’, ’analytics’, ’python’, ’build_automation’, ’shell’, ’bayesian’, ’typescript’, ’haskell’, ’r’, ’html’, ’random_generation’, ’lambda’, ’code’, ’data_analysis’, ’data_science’, ’clustering’, ’datascience’, ’equations’, ’math’, ’c++’, ’embedding’, ’jupyter_notebook’, ’snp’, ’ssimp_software’, ’data_processing’]
75	[’openarms’, ’eggplant’, ’realm’, ’arduino’, ’android_dev’, ’python’, ’source’, ’reporters’, ’javascript’, ’junit’, ’apollo’, ’face’, ’react’, ’web_application’, ’test’, ’typescript’, ’wrappers’, ’open’, ’detection’, ’app’, ’java’, ’css’, ’html’, ’mocha’, ’php’, ’integration’, ’inspector’, ’studio’, ’jasmine’, ’ruby’, ’websockets’, ’client’, ’c++’, ’elm’, ’bug_finding’, ’human_language_proccessing’, ’mongodb’, ’system’, ’bootstrap’, ’unit_tests’, ’nodejavascript’, ’health’]
76	[’cross’, ’bzip2’, ’projects’, ’ui_design’, ’c’, ’data’, ’steam’, ’rvm’, ’apache’, ’source’, ’industrial’, ’javascript’, ’jruby’, ’web_application’, ’macos’, ’xcode’, ’backend’, ’open’, ’app’, ’css’, ’java’, ’enterprise’, ’html’, ’php’, ’platform_dev’, ’ios’, ’jpa’, ’project’, ’ruby’, ’swift’, ’jekyll’, ’package_managment’, ’maven’, ’web_application_dev’, ’spring_framework’, ’objective_c’, ’mongodb’, ’spring’, ’git’]

77	[’cuda’, ’flye’, ’perl’, ’c’, ’polymorphic’, ’nanopore’, ’cpu_based_implementation’, ’algorithms’, ’data’, ’m4’, ’python’, ’go’, ’web_development’, ’biotechnology’, ’editing’, ’ngs’, ’genomes’, ’accelerated’, ’heterozygous’, ’molecular’, ’hi’, ’golang’, ’gpu’, ’html’, ’random_generation’, ’sequencing’, ’dna’, ’analysis’, ’emac_extension’, ’indexing’, ’biology’, ’tex’, ’molecular_biology’, ’c++’, ’research’, ’jupyter_notebook’, ’pytorch’, ’rna’, ’ubuntu’]
78	[’web_scraping’, ’linux’, ’application’, ’c’, ’perl’, ’parser’, ’python’, ’go’, ’search_algorithms’, ’shell_scripts’, ’apps’, ’shell’, ’sql’, ’javascript’, ’dart’, ’interactive’, ’mobile’, ’jquery’, ’typescript’, ’macos’, ’golang’, ’viewer’, ’java’, ’css’, ’html’, ’process’, ’code’, ’visual_studio’, ’option’, ’ruby’, ’swift’, ’command’, ’line’, ’hypersql’, ’groovy’, ’maven’, ’web_application_dev’, ’bootstrap’]
79	[’assembly’, ’web_scraping’, ’c’, ’cryptography’, ’web_dev’, ’reactome’, ’python’, ’go’, ’shell_scripts’, ’poppler’, ’javascript’, ’encoding/decoding’, ’regular_expressions’, ’image’, ’golang’, ’r’, ’cairo’, ’html’, ’css’, ’data_analysis’, ’data_science’, ’computational’, ’coffeescript’, ’biology’, ’keras’, ’c++’, ’posix’, ’hacking’, ’research’, ’jupyter_notebook’, ’unix’, ’data_visualization’, ’nodejavascript’]

80	['unit_testing', 'crm', 'tracking', 'server', 'sql_bulk_operations', 'web_scraping', 'pipeline', 'objective_c++', 'builder', 'algorithms', 'web_dev', 'azure_database', 'python', 'dataflow', 'shell_scripts', 'data_pipelining', 'mvvm', 'c#', 'ncalc', 'sql', 'javascript', 'sql-database', 'database', 'html', 'batch', 'css', 'php', 'ios', 'javascripton', 'equations', 'powershell', 'markdown', 'adodot_net', 'petapoco', 'processing', 'dot_net', 'android', 'url', 'progress', 'framework', 'azure', 'http', 'xml']
81	['boringssl', 'kodi', 'linux', 'nuget', 'perl', 'c', 'itk', 'toolkit)', 'debian', 'python', 'microsoft', 'shell', 'c#', 'javascript', 'openssl', 'turborle', 'html', 'imaging', 'xslt', 'javascripton', 'powershell', 'ruby', 'tex', 'swig', 'systemd', 'c++', 'openelec', 'cmake', 'medical', 'gdcm', 'makefile', 'biorad', 'roff', '(insight', 'xml', 'dicom']
82	['s3transfer', 'github', 'scala', 'clojure', 'perl', 'gnome', 'openapi3', 'vala', 'docker', 'ruby', 'aws', 'puppet', 'c++', 'python', 'go', 'elixir', 'openwrt', 'dark_reader', 'cassandra', 'd', 'posix', 'mozilla_firefox', 'shell', 'moto', 'javascript', 'rust', 'makefile', 'typescript', 'lua', 'nvidea', 'java', 'html', 'dockerfile', 'bcc']
83	['data_science', 'julia', 'ui_design', 'c', 'matplotlib', 'spell_check', 'ruby', 'mapbox', 'jupyter', 'c++', 'python', 'go', 'tensorflow', 'haskell', 'flask', 'shell', 'kaggle', 'javascript', 'jupyter_notebook', 'vim', 'rust', 'plotly', 'machine_learning', 'scikit_learn', 'makefile', 'react', 'typescript', 'lua', 'vue', 'vim_script', 'html']

84	[’github’, ’c’, ’protoexplorer’, ’python’, ’go’, ’usability’, ’web_development’, ’browserify’, ’versioning’, ’vagrant’, ’shell’, ’javascript’, ’fdpa_fusiontables’, ’oracle’, ’dotfiles’, ’jquery’, ’react’, ’opentreemap’, ’java’, ’css’, ’html’, ’mocha’, ’es6’, ’analysis’, ’stack’, ’topojavascripton’, ’coffeescript’, ’less’, ’ruby’, ’hci’, ’cjavascriptx’, ’front_end’, ’bootstrap’, ’clajavascript’, ’vim_script’, ’2d_mazes’, ’ckan’]
85	[’cuda’, ’cross’, ’oil_javascript’, ’rna_manipulation’, ’simutrans’, ’linear_algebra’, ’c’, ’consent_manager’, ’libvpx’, ’hatspil’, ’python’, ’xenome’, ’javascript’, ’jvm’, ’typescript’, ’intel’, ’r’, ’mypy’, ’html’, ’classify’, ’ethminer’, ’php’, ’fortran’, ’cookie_banner’, ’gaming’, ’interface’, ’dlib’, ’c++’, ’svn’, ’map’, ’platform’, ’rust’, ’makefile’, ’rna’, ’gzip’, ’invoiceplane’, ’openblas’]
86	[’space_exploration’, ’spotipy’, ’scrapy’, ’selenium’, ’fake_useragent’, ’python’, ’django’, ’gitignore’, ’javascript’, ’junit’, ’web’, ’grunt_plugin’, ’eslint_rules’, ’wrapper’, ’java’, ’html’, ’css’, ’angular’, ’mocha’, ’ssh’, ’beautifulsoup’, ’astronomy’, ’joke’, ’galen’, ’selectors’, ’automation’, ’e2e’, ’jasmine’, ’client’, ’api’, ’scraping’, ’nyc’, ’clean_code’, ’pytest’, ’protractor’, ’eslint’]
87	[’scala’, ’postgresql’, ’containerization’, ’cassie_schema_migrator’, ’c’, ’erlang’, ’coffeescript’, ’powershell’, ’swift’, ’ruby’, ’apache_cassandra’, ’geopositional’, ’c++’, ’python’, ’go’, ’cli’, ’dot_net’, ’css’, ’c#’, ’shell’, ’javascript’, ’datastax’, ’jq’, ’f#’, ’database_migrations’, ’hcl’, ’bash’, ’spring’, ’vue’, ’nodejavascript’, ’java’, ’html’, ’dockerfile’]

88	[’xslt’, ’caja’, ’validation_library’, ’skeleton’, ’perl’, ’c’, ’link_extractor’, ’jekyll_bootstrap’, ’gaming’, ’ruby’, ’coptr’, ’cca’, ’tika’, ’python’, ’c++’, ’jekyll’, ’apache’, ’httpreserve’, ’batchscript’, ’d’, ’bug_reports’, ’shell’, ’newlisp’, ’javascript’, ’dataservices’, ’kb_python_api’, ’bootstrap’, ’eaas_framework’, ’lua’, ’wrapper’, ’template’, ’java’, ’html’]
89	[’linux’, ’ffmpeg’, ’c’, ’cloud_gaming’, ’python’, ’shell’, ’homebrew’, ’javascript’, ’prometheus_monitoring’, ’bufferer’, ’web’, ’video’, ’r’, ’java’, ’multimedia’, ’css’, ’dockerfile’, ’experience’, ’mappr’, ’docker’, ’ruby’, ’swift’, ’(qoe)’, ’tex’, ’c++’, ’package_manager’, ’cli’, ’audio_normalization’, ’android’, ’quality’]
90	[’php’, ’scala’, ’perl’, ’c’, ’software_engineering’, ’java’, ’docker’, ’ruby’, ’catalog’, ’solidity’, ’python’, ’go’, ’gettext’, ’c++’, ’mvc’, ’css’, ’flask’, ’shell’, ’vim’, ’javascript’, ’openscad’, ’bootstrap’, ’makefile’, ’typescript’, ’nodejavascript’, ’gettext_catalog’, ’vim_script’, ’dockerfile’, ’html’]
91	[’kuber_netes’, ’scala_tdd’, ’scala’, ’julia’, ’cryptography’, ’data’, ’big’, ’python’, ’emacs_configuration’, ’apache’, ’elastic_bean_stalk’, ’parallel_programming’, ’expressjavascript’, ’machine_learning’, ’mockito’, ’spark’, ’hadoop’, ’mysql’, ’springboot’, ’haskell’, ’java’, ’log4j’, ’functional_programming’, ’terraform’, ’clojure’, ’pipelines’, ’docker’, ’tensorflow’, ’cassandra’, ’restful_api’, ’microservice’, ’map’, ’vim’, ’reduce’, ’jupyter_notebooks’, ’deep_learning’, ’visualization’, ’nodejavascript’]

92	[’machine’, ’crow’, ’cpp’, ’jdk’, ’arduino’, ’jdk8’, ’python’, ’recognition’, ’webgl’, ’canvas’, ’c#’, ’javascript’, ’jquery’, ’restful’, ’threads’, ’hydrate_javascript’, ’typescript’, ’libgdx’, ’java’, ’dockerfile’, ’database’, ’mongoid’, ’asynchronous’, ’android_studio’, ’tilemap’, ’kotlin’, ’gradle’, ’angularjavascript’, ’bitmap’, ’tree’, ’ruby’, ’node’, ’speech’, ’android’, ’heroku’, ’structure’, ’xml’, ’git’]
93	[’acid’, ’bug_fixing’, ’jdbc’, ’based’, ’event’, ’repl’, ’rest’, ’data’, ’user_interface’, ’graph’, ’c#’, ’sql’, ’machine_learning’, ’jquery’, ’deveeldb’, ’wrapper’, ’oss’, ’java’, ’css’, ’server_connection’, ’html’, ’flow’, ’sirius’, ’breakpoint’, ’visual_studio’, ’magnific_popup’, ’javascripton’, ’scripting’, ’google_analytics’, ’ruby_gem’, ’language’, ’jekyll’, ’tensorflow’, ’saas/scss_preprocessor’, ’dot_net’, ’openamatdot_net/deveel/’, ’developer’, ’linq’, ’msbuild’, ’sucy’]
94	[’postgresql’, ’linux’, ’aws’, ’rest’, ’ansible’, ’aws_dynamo_db’, ’python’, ’tika’, ’lambdas’, ’apache’, ’inventory’, ’toolbox’, ’shell’, ’javascript’, ’database_creation’, ’mapbox_studio’, ’web’, ’dynamodb’, ’d3’, ’html’, ’css’, ’sandbox’, ’functional’, ’significant’, ’javascripton’, ’functional_programming’, ’web_designing’, ’gem’, ’database_initialisation’, ’project’, ’ruby’, ’line’, ’command’, ’api’, ’website’, ’dns’, ’quering’, ’os’, ’unix’, ’euler’, ’ubuntu’]

95	['php', 'webserver', 'interfaces', 'html', 'agile', 'javascripton', 'sampling', 'c', 'weblog', 'interface_distiller', 'bubbly', 'tracing', 'ruby', 'php_interpreter', 'rest', 'class', 'contributed', 'd3javascript', 'api', 'span_management', 'pecl', 'commits', 'shell', 'user', 'exporters', 'javascript', 'webdevelopment', 'web', 'php_unit_schema', 'backlog', 'framework', 'semantic_versioning', 'visualization', 'stackoverflow', 'census', 'xml', 'phpunit']
96	['webserver', 'page_controllers', 'linux', 'redbean', 'cryptography', 'sqlite3', 'apache', 'shell', 'javascript', 'data_objects', 'web', 'webstorm', 'mysql', 'java', 'css', 'html', 'database', 'log', 'powerapi', 'intellij', 'php', 'javascripton', 'redbean_orm', 'graphql', 'scheme', 'google_analytics', 'docker', 'docker_file', 'websockets', 'magento', 'api', 'os', 'unix', 'laravel', 'visualization', 'idea', 'twig_extension', 'webapps', 'xml']
97	['owl', 'database', 'javascripttettlers2', 'javascripton', 'c', 'cpp', 'basic', 'webbased', 'visual', 'bpmn_graph', 'excel', 'jekyll', 'c++', 'web_development', 'mono', 'api', 'dot_net', 'user_interface', 'css', 'maven', 'c#', 'dartjavascriptonp', 'javascript', 'vue_javascript', 'uml', 'dart', 'web', 'vsto', 'annotation', 'framework', 'mysql', 'rendering', 'awesome_vue', 'java', 'html']

98	[’webpush’, ’html’, ’push’, ’javascripton’, ’phantomjavascript’, ’webpack’, ’pipeline’, ’selenium’, ’webdrivers’, ’ruby’, ’authenticating’, ’ssl’, ’phoenix_base’, ’dotenov’, ’elixir’, ’serviceworker’, ’strave_api’, ’coffee_script’, ’shell’, ’javascript’, ’webpacker’, ’protocol’, ’web’, ’tacoki’, ’apitome’, ’rails’, ’wtf’, ’teaspoon’, ’vue’, ’nodejavascript’, ’css’, ’flatpickr’]
99	[’xslt’, ’linux’, ’html’, ’genetic’, ’d3’, ’c’, ’cpp’, ’based’, ’tex’, ’visualizations’, ’python’, ’jekyll’, ’sckit’, ’c++’, ’tskit’, ’gene_floe_interface’, ’linkage’, ’website’, ’skelml’, ’shell’, ’javascript’, ’pca’, ’jupyter_notebook’, ’slim’, ’toolkit’, ’plotly’, ’interactive’, ’effects’, ’latex’, ’raster’, ’r’, ’nodejavascript’, ’java’, ’xml’, ’bioinformatics’, ’xlst’]
100	[’translator’, ’recording’, ’linux’, ’library’, ’data’, ’conversation_javascript’, ’curvature’, ’getstats’, ’screen’, ’streaming’, ’webgl’, ’websync’, ’collaborative’, ’canvas’, ’javascript’, ’datachannel’, ’sharing’, ’video’, ’file’, ’rtcmulticonnection’, ’socket’, ’html’, ’css’, ’bandwidth’, ’file_buffer_reader’, ’capturing’, ’javascripton’, ’webrtc’, ’conversation’, ’getscreenld’, ’extensions’, ’conferencing’, ’multistreammixer’, ’media’, ’android’, ’unix’, ’channels’, ’nodejavascript’, ’xml’]

Table 19: Parameters of models from experiment 1A.

Top 3 Results	Model Parameters	**TDistr**	**LDA_AVG**	**LDA_MAX**	**W2V_AVG**	**W2V_MAX**
1	Topic Number	11	10	11	**16**	11
	Beta Value	0.01	0.005	0.05	**0.05**	0.05
2	Topic Number	11	10	11	16	11
	Beta Value	0.005	0.001	0.001	0.001	0.001
3	Topic Number	11	10	11	16	11
	Beta Value	0.05	0.01	0.005	0.005	0.005

Table 20: Parameters of models from experiment 1B.

Top 3 Results	Model Parameter	**TDistr**	**LDA_AVG**	**LDA_MAX**	**W2V_AVG**	**W2V_MAX**
1	Topic Number	40	33	**33**	31	48
	Beta Value	0.01	1	**1**	0.1	1
2	Topic Number	29	27	27	48	27
	Beta Value	0.1	0.5	0.5	1	1
3	Topic Number	45	16	27	14	31
	Beta Value	0.1	0.5	1	1	1

Table 21: Parameters of models from experiment 2A.

Top 3 Results	Model Parameters	**TDistr**	**LDA_AVG**	**LDA_MAX**	**W2V_AVG**	**W2V_MAX**
1	Topic Number	11	21	16	16	16
	Beta Value	0.005	0.05	0.05	0.01	0.05
2	Topic Number	11	11	10	16	16
	Beta Value	0.01	0.01	0.001	0.005	0.01
3	Topic Number	11	11	10	16	16
	Beta Value	0.001	0.001	0.005	0.05	0.001

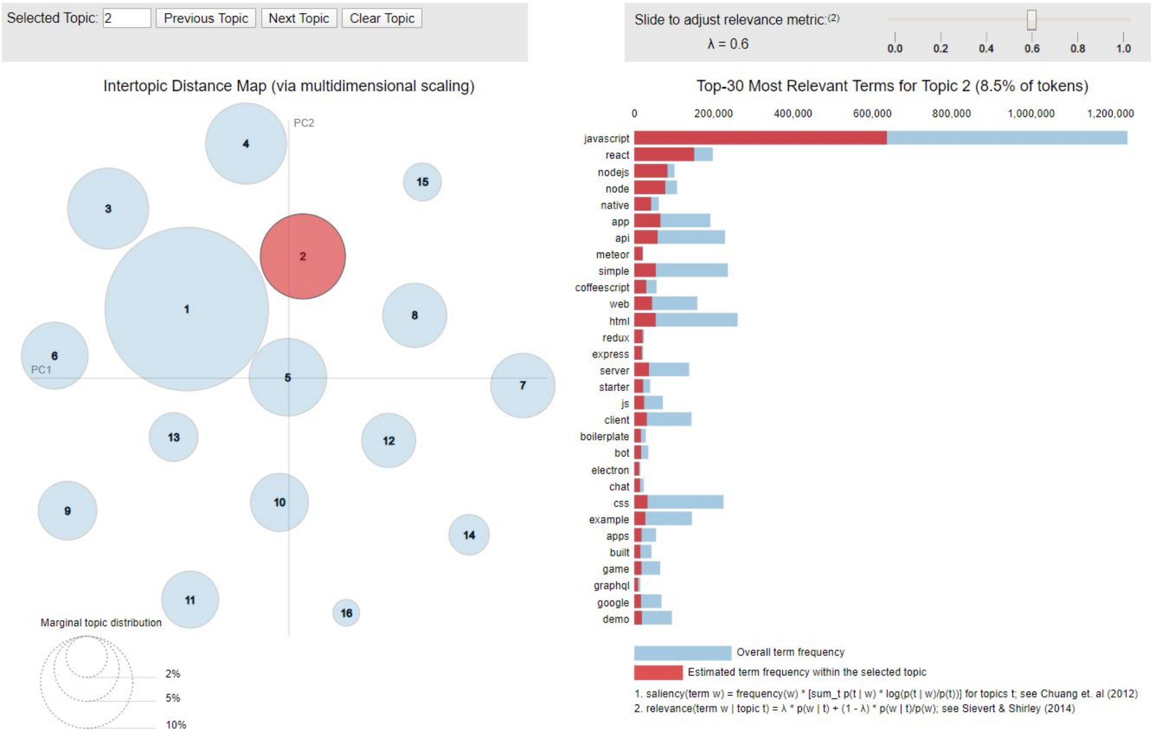
Table 22: Parameters of models from experiment 2B.

Top 3 Results	Model Parameter	**TDistr**	**LDA_AVG**	**LDA_MAX**	**W2V_AVG**	**W2V_MAX**
1	Topic Number	40	13	33	14	30
	Beta Value	0.01	0.1	1	1.0	0.001
2	Topic Number	13	36	27	32	30
	Beta Value	0.1	1.0	0.5	0.1	0.1
3	Topic Number	45	27	36	16	30
	Beta Value	0.1	0.01	1	0.5	0.005

A.3 Results

A.3.1 Parameters of Models from Experiments

A.3.2 Data Visualization for Best LDA Models

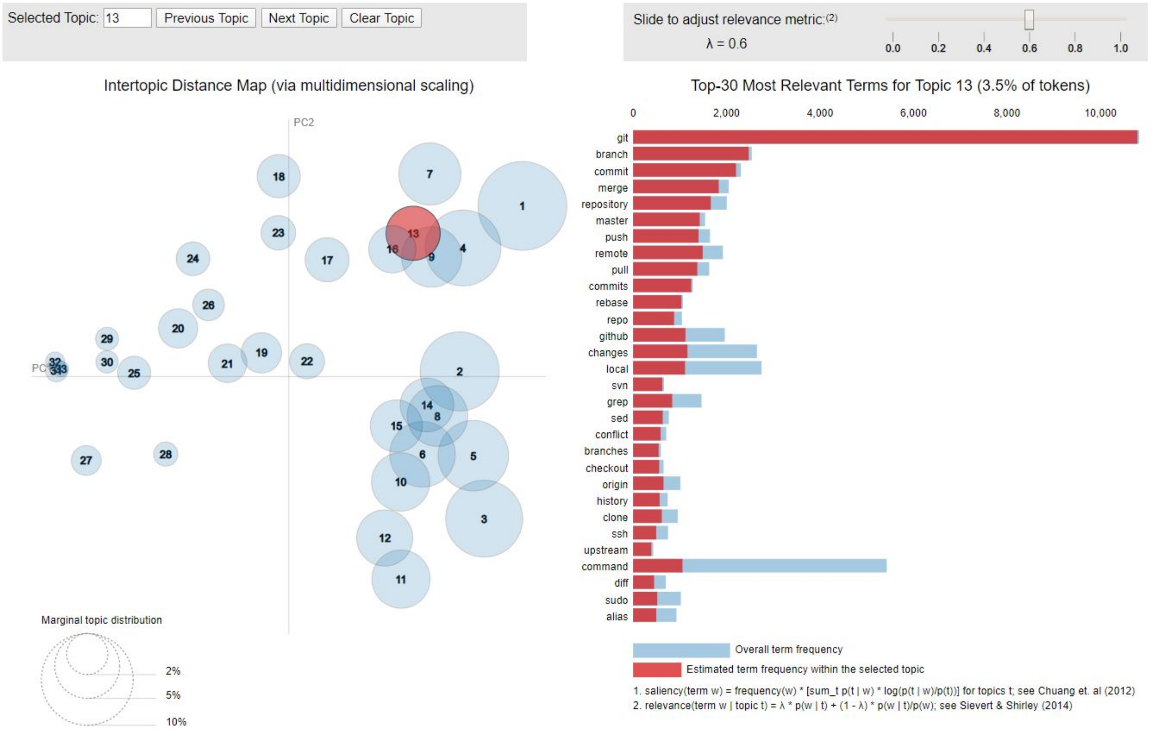


Selected Topic: 2

Slide to adjust relevance metric:(2) λ = 0.6

1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

Figure 18: Visualization of LDA model for GitHub data.



Selected Topic: 13

Slide to adjust relevance metric:(2) λ = 0.6

1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

Figure 19: Visualization of LDA model for Stack Overflow data.