## Graphical Abstract

**GitRanking: A Ranking of GitHub Topics for Software Classification using Active Sampling**

Cezar Sas, Andrea Capiluppi, Claudio Di Sipio, Juri Di Rocco, Davide Di Ruscio

## Highlights

**GitRanking: A Ranking of GitHub Topics for Software Classification using Active Sampling**

Cezar Sas, Andrea Capiluppi, Claudio Di Sipio, Juri Di Rocco, Davide Di Ruscio

- We create a new hierarchical taxonomy for software classification;
- Our taxonomy is expandable and grounded in a knowledge base;
- We study developers' behaviour when labelling projects;
- We study classification models' behaviour at various levels in the hierarchy;
- We release a large-scale, easy-to-expand dataset.

# GitRanking: A Ranking of GitHub Topics for Software Classification using Active Sampling

Cezar Sas[a,*], Andrea Capiluppi[a], Claudio Di Sipio[b], Juri Di Rocco[b], Davide Di Ruscio[b]

[a] *University of Groningen, Nijenborgh 9, Groningen, 9747 AG, Netherlands*
[b] *University of L'Aquila, Via Vetoio, L'Aquila, 67100, Italy*

## Abstract

**Context**: GitHub is the world's most prominent host of source code, with more than 327M repositories. However, most of these repositories are not labelled or inadequately, making it harder for users to find relevant projects. Various proposals for software application domain classification over the past years have been proposed. However, these several of those approaches suffer from multiple issues, called *antipatterns* of software classification, that reduce their usability.

**Objective:** In this paper, we propose a new taxonomy in the GitHub ecosystem, called GitRanking, starting from a well-structured dataset, composed of curated repositories annotated with topics. The main objective is to create a baseline methodology for software classification that is expandable, hierarchical, grounded in a knowledge base, and free of antipatterns.

**Method**: We collected 121K topics from GitHub and used GitRank-

*Corresponding Author
*Email addresses:* `c.a.sas@rug.nl` (Cezar Sas), `a.capiluppi@rug.nl` (Andrea Capiluppi), `claudio.disipio@graduate.univaq.it` (Claudio Di Sipio), `juri.dirocco@univaq.it` (Juri Di Rocco), `davide.diruscio@univaq.it` (Davide Di Ruscio)

ing to create a taxonomy of 301 ranked application domains. GitRanking 1) uses *active sampling* to ensure a minimal number of annotations to create the ranking; and 2) links each topic to Wikidata, reducing ambiguities and improving the reusability of the taxonomy. Furthermore, we adopt the conceived taxonomy in a classification task by considering a state-of-the-art classifier.

**Results**: Our results show that GitRanking can effectively rank terms in a hierarchy according to how general or specific their meaning is. Furthermore, we show that GitRanking is a dynamically extensible method: it can currently accept further terms to be ranked, and with a minimum number of annotations ($\sim 15$). Concerning the classification task, we show that the model achieves an F1-score of 34%, with a precision of 54%.

**Conclusion**: This paper is the first collective attempt at building a ground-up taxonomy of software domains. Our vision is that our taxonomy, and its extensibility, can be used to better and more precisely label software projects.

*Keywords:* Software Classification, Active Sampling, Taxonomy, GitHub

## 1. Introduction

GitHub is the world's largest host of source code, with more than 327M repositories in July 2022[1]; moreover, the number of repositories increased by 60M+ in the previous year[2]. However, these repositories are not easy to find: while GitHub allows developers to annotate their projects manually and

[1]https://github.com/search
[2]https://octoverse.github.com/

other users to search software via *Topics*[3], not all projects are making use of it, or use it inefficiently, by just annotating with one or two topics, which can be very generic and not descriptive. Additionally, developers are allowed to annotate a project with any string they want: this inevitably generates a variety of alternative forms for the same concept, reducing the ease with which a repository can be found.

Various works in the literature have attempted to automatically classify the domains of software applications, many proposing their datasets with custom taxonomies [1, 2, 3], and more recently, using a subset of GitHub Topics [4, 5].

However, these resources suffer from various recurring problems, that we termed *antipatterns* of software classification in a previous work [6], and that affect the usability of these taxonomies in real-world applications. Here we give a brief recap of the issues that accompany them: first, no current taxonomy explicitly defines a hierarchical relation among their labels, making it problematic when dealing with their single label annotation and '`IS-A`' relationships among labels (*mixed granularity* antipattern). A second issue is the mix of different taxonomies in the same categorization (*mixed taxonomies* antipattern), for example, when labels from various application domains (e.g., '*Security*') are present as well as programming languages (e.g., '*Python*'). This is an issue when performing single-task classification as opposed to multi-task [7], which would not be possible given the lack of separation between the taxonomies. Furthermore, these categorizations are not complete,

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

as they do not cover the entire spectrum of software categories (e.g., having the '*Compiler*' category, but missing the '*Interpreter*' one), making it easier for a model to distinguish some classes. The incompleteness is also aggravated by the fact that no work is grounded in a knowledge base (KB): this is highly problematic because it does not resolve the ambiguity of an arbitrarily defined categorization (i.e., top-down), reducing its usability and the possibility to add new terms to it (*ambiguity* issue). All the issues above make these categorizations less valuable to be used in a real-world scenario.

As an alternative to the pre-defined taxonomies, presented by previous works in software application domain classification, previous works in the Natural Language Processing field have focused on building taxonomies from data [8, 9]. While some solutions focus on creating a taxonomy for the Computer Science domain using papers from the bibliography service DBLP [10, 11] or other literature [12], these solutions require a large amount of data to create the taxonomy [13]; however, a similar amount of data is not always available for GitHub Topics. Furthermore, many solutions are not deterministic, requiring multiple runs and a lucky seed to get a good starting point for annotators to build upon. Finally, the results of these solutions are not always made available by the authors, and for other researchers to inspect, utilise or even extend.

In order to solve the above-mentioned issues, in this paper, we present GitRanking, a framework for ranking software application domains. Differently from previous work, we defined a pipeline for selecting the topics, using 121K GitHub Topics as our initial seed. GitRanking uses an active sampling method, combined with a Bayesian inference algorithm, to create the ranking of the topics. To reduce the intrinsic ambiguity of natural language, and make the taxonomy more usable, we have linked each term to its Wikidata entity. Finally, one key feature of GitRanking is the ability to expand the ranked taxonomy easily, and by anyone interested, with a minimal amount of examples (≈15 for each new topic added).

As an extended finding, GitRanking has also allowed us to extract insights about the performance of state-of-the-art classification models, and on how developers use GitHub Topics in practice, using the research questions below:

> ***RQ1****: Are the topics used to annotate GitHub projects* ***evenly distributed*** *in the levels of a taxonomy?*
>
> ***RQ2****: Does the classification models* ***performance*** *vary across different levels in a taxonomy?*

To answer the **RQ1**, we conduct a well-structured analysis on the results obtained by adopting GitRanking on a dataset composed of Github repositories and their related topics, belonging to heterogeneous application domains, e.g., computer science, physics, and mathematics, to name a few. Concerning the classification task presented in **RQ2**, we reuse as a baseline our prior work in the classification Github topics, the approach is based on a stochastic multi-label classifier [4]. To assess the contribution of GitRanking, we measure the overall accuracy of the model by using a well-known set of metrics, i.e., success rate, precision, recall, and F1-score. These results help developers better annotate their projects and make them more discoverable and easier to find. This improved discoverability will also help other developers, as it will be easier and faster to find the best component or library for a specific task, improving software's reusability.

In summary, our contributions are:

– an online framework for creating software categorizations, and expanding them;

– a list of 301 application domains extracted from GitHub Topic, and disambiguated by linking them to Wikidata;

– a ranking of the 301 topics into discrete levels of the same taxonomy, and based on their meaning;

– a new large-scale and expandable dataset for software classification;

For the sake of inspection by other researchers, and easier reproducibility of our results, we made our code[4] and data[5] available.

This paper is articulated as follows: in Section 2 we analyze the past works in terms of existing taxonomies, and the approaches that were used to extract one from data. In Section 3 we present the problem statement, the ASAP (*Active SAmpling for Pairwise comparisons*) algorithm, which we used for reducing the annotations required for the ranking, and the TrueSkill ranking system which creates a ranking of the annotated topics. Section 4 describes the pipeline of GitRanking, and its activities. Section 5 presents the results of the work performed by the annotators, and the TrueSkill output.

[4]https://github.com/SasCezar/GitHubClassificationDataset

[5]https://zenodo.org/record/6854770

In Section 6 we present our classification dataset and baseline scores: we discuss these findings and scores in Section 7. We analyze the threats to validity that we encountered in Section 8. Finally, we present the conclusion and future works in Section 9.

## 2. Related Work

In this section we present the relevant related work. We address works dealing with software classification problems, in particular the classification of application domains, and research focusing on the construction (or induction) of taxonomies.

### *2.1. Software Classification Taxonomies*

Various attempts in the literature have focused on software classification, from application domains [14, 4], to bugs [15], and vulnerabilities [16]. This paper focuses on works performing classifications based on software application domains.

One of the initial works on software classification is MUDABlue [17]. The authors proposed a dataset of 41 projects written in C and divided them into six categories. They also presented a model based on information retrieval techniques, specifically Latent Semantic Analysis (LSA), to classify software based on its source code identifiers.

Tian et al. proposed LACT [18]. As for MUDABlue, the authors proposed both a new dataset and a new approach to classification. The dataset consists of 43 examples divided into 6 SourceForge categories. The list of projects is available in their paper. Their classification model combines Latent Dirichlet Allocation (LDA), a generative probabilistic model that retrieves topics from textual datasets, and heuristics. The authors used the identifiers and comments in the source code as input to their model.

Again, in [14], the authors propose a new dataset using SourceForge as seed. The dataset consists of words extracted from API packages, classes, and methods names using naming conventions. However, the dataset containing 3,286 Java projects annotated into 22 categories is no longer available. Using the example in [19], the authors use information gain to select the best attributes as input to different machine learning methods.

LeClair et al. [20] worked on a dataset of C/C++ projects from the Debian package repository. The dataset consists of 9,804 software projects divided into 75 categories: many of these categories have only a few examples, and 19 are the same categories with different surface forms, more specifically 'contrib/X', where X is a category present in the list. For the classification, they used a neural network approach. The authors used the project name, function name, and function content as inputs to a C-LSTM [21], a combined convolutional and recurrent neural networks model.

In [2] the authors proposed an approach to generate tag clouds starting from bytecodes, external dependencies of projects, and information extracted from Stack Overflow. Unfortunately, their dataset is not available.

Sharma et al. [1] released a list of 10,000 examples annotated by their model into 22 categories, evaluated using 400 manually annotated projects. It is interesting to notice that half of the projects eventually end up in the 'Other' category, which means that they are not helpful when training a new model. They used a combined solution of topic modelling and genetic algorithms called LDA-GA for the classification [22]. The authors applied

LDA topic modelling on the `README` files and optimize the genetic algorithms' hyper-parameters. While LDA is an unsupervised solution, humans are needed to annotate the topics from the identified keywords.

In ClassifyHub [23], the authors used the InformatiCup 2017[6] dataset, which contains 221 projects unevenly divided into seven categories. For the classification, they propose an ensemble of 8 naïve classifiers, each using different features (e.g., file extensions, `README`, GitHub metadata and more).

In [3], the authors released two datasets spanning two domains: an 'artificial intelligence' taxonomy with 1,600 examples and a 'bioinformatics' one with 876 projects. The datasets have been annotated according to a hierarchical classification that is given as an input with keywords for each leaf node. Furthermore, they proposed HiGitClass, an approach for modeling the co-occurrence of multimodal signals in a repository (e.g., user, repository name, `README`, and more) to perform the classification.

Focusing on unsupervised approaches, we find CLAN [24], which provides a way to detect similar apps based on the idea that similar apps share some semantic anchors. As in previous work, the authors also proposed a dataset which is no longer available. Given a set of applications, the authors created two terms-document matrices: the structural information using the package and API calls, and the other for textual information using the class and API calls. Both matrices are reduced using LSA, and then the similarity across all applications is computed. Lastly, the authors combined the package and class similarities by summing the entries. In [25], the authors proposed

[6]https://github.com/informatiCup/informatiCup2017

CLANdroid, a CLAN adaptation to the Android apps domain, and evaluated the solution on 14,450 Android apps. Unfortunately, their dataset is not available, and thus, we cannot conduct a comparison with GitRanking.

Another unsupervised approach was adopted by LASCAD [26]. However, unlike other unsupervised methods, the authors proposed an annotated dataset consisting of 103 projects divided into six categories (from GitHub Collections) with 16 programming languages (although many languages have only 1 example) and an unlabeled one which is not available. Their approach uses a language-agnostic classification and similarity tool. As in LACT, the authors used LDA over the source code and further applied hierarchical clustering with cosine similarity on the output topic terms matrix of LDA to merge similar topics.

More recent works focus on using GitHub as the source of their classification. In our prior work [4], we propose a multi-label classifier that predicts a curated list of topics given a README of a GitHub repository. The used dataset contains around 10,000 labelled projects in different programming languages. As a preprocessing step, we encode the content of `README` files by employing the TF-IDF weighting scheme. Afterwards, we feed a probabilistic model called Multinomial Naïve Bayesian Network (MNB) to recommend new possible topics for the project. In this paper, we opt to reuse a modified version of the MNB as a baseline to highlight the contribution of GitRanking. We succeeded in extending this work by proposing TopFilter [27], which combines the MNB network with a collaborative filtering engine to include non-featured topics in the list of recommendations. The system encodes the repositories and topics in a graph-based structure. Afterwards, the underpinning recommendation algorithm computes the Cosine similarity using featured vectors to recommend the most similar topics.

Similarly, Repologue [28] proposed a dataset based on popular GitHub Topics; however, the dataset is unavailable. For the classification, they also adopted a multimodal approach. They feed as input to a fully connected neural network, the dense vector representation (i.e., embeddings) created by BERT [29], a neural language model. BERT creates the embedding of the project names, descriptions, `README`s, wiki pages, and file names concatenated. Unfortunately, we cannot compare directly GitRanking due to the lack of a working replication package. Nevertheless, we reuse part of the Repologue dataset to show the contribution of our taxonomy in the classification task.

GHTRec [5] has been proposed to recommend personalized trending repositories, i.e., a list of most starred repositories, by relying on the BERT language model (LM) and GitHub Topics. Given a repository, the system predicts the list of topics using the preprocessed `README` content. Afterwards, GHTRec infers the user's topic preferences from the historical data, i.e., commits. The tool eventually suggests the most similar trending repositories by computing the similarity on the topic vectors, i.e., cosine similarity and shared similarity between the developer and a trending repository. They use the dataset of [4].

## *2.2. Automatic Taxonomy Construction*

Automatic taxonomy construction or induction is a challenging task in the field of natural language processing as it requires models' understanding of the *hypernym* relation. Hypernymy, or '`IS-A`' relation, is a lexical-semantic relation in natural languages, which associates general terms to their instances or subtypes.

With the large Web data available, many taxonomies are constructed from human resources such as Wikidata. However, even these huge taxonomies may lack domain-specific knowledge. Therefore, many automatic approaches to construct domain-specific ones have been proposed. From hypernymy discovery and lexical entailment [8] approaches to instance-based taxonomy [9, 30], and clustering-based taxonomy methods [10, 11].

An example of approaches focusing on the hypernymy discovery task is [8]; they propose a distributional approach that fixes some of the issues present with such methods, making them achieve comparable performance with respect to the simple, pattern-based methods.

While shifting a bit from the hypernymy discovery methods, [9], and [30] make use of the patterns matching for creating their datasets. In [30], the authors use a pre-trained language model and distantly annotated data collected by scraping the web. They finetune BERT to learn a hypernymy relation between words. In [9], they use a dataset built by pattern matching to construct a noisy hypernymy graph and train a Graph Neural Network [31] using a set of taxonomies for some known domains. The learned model is then used to generate a taxonomy for a new unknown domain given a set of terms for the new domain.

Examples of clustering-based approaches include TaxoGen [10], NetTaxo [11], and Corel [32]. Their work is similar in nature; all focus on creating a taxonomy from DBLP's bibliography, making it relevant to our research. However, attempts at reproducing their results have failed [7,8], and their results are not publicly available, except for the small samples included in the paper. Their approaches are similar and are based on learning semantic vectors (embedding) for the words of interest. They perform an iterative sequence of learning embeddings: perform clustering and subsequently, for each cluster, repeat the steps to create each time a better representation that is more discriminative. However, this requires a large quantity of data, which is hard to collect [13]; moreover, for each newly added term, a new run of the algorithms is required, which are heavily demanding in terms of computation and time. Furthermore, our attempts at reproducing their results failed.

Lastly, the Computer Science Ontology (CSO) [12] proposes a domain specific ontology. It is a large-scale ontology of research areas that was automatically generated using the Klink-2 algorithm [33] on a dataset of scientific publications, with the latest version, 3.3, using data from 2021. Like previous works, it requires a large amount of data to train its algorithm.

A more comprehensive study of the taxonomy construction research area is presented in [13].

## 3. Motivation and Background

In this section we describe the problem statement and the background knowledge of methods we use in our work.

[7]https://github.com/xinyangz/NetTaxo/issues
[8]https://github.com/teapot123/CoRel/issues

## 3.1. Problem Statement

Software developers use open-source software (OSS) repositories to publish and disseminate their work. GitHub is among the most popular platforms that offer open environments where developers can share their code and interact with each other. To assist developers in approaching projects of their interest, GitHub offers the possibility of manually tagging their own repositories with a set of keywords, called *topics*. Although useful, this list of hand-written tags can contain errors that could compromise the discoverability of the project, as can be seen in the left side of Figure 1.

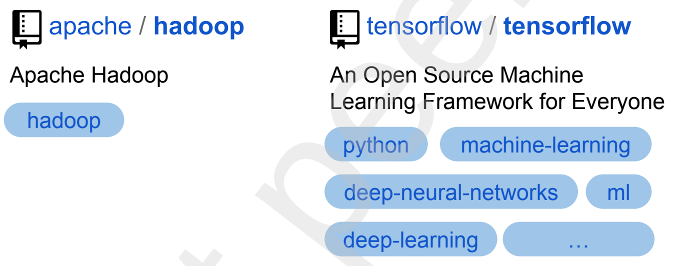

Figure 1: Two exemplar GitHub projects with their annotations.

As seen in Figure 1, the `apache/hadoop` project is poorly annotated with just a single GitHub Topic, which incidentally is the project name. `tensorflow/tensorflow` on the other hand is well annotated with various relevant Topics. We can also see that the Topic '*Machine Learning*' is present in two forms: '*machine-learning*' and '*ml*'. Considering how is it is to discover one of these projects, the latter is clearly easier to find by other developers, than the former, and based on the chosen labels.

## *3.2. Active Sampling for Pairwise Comparisons*

Modelling subjective characteristics of items, e.g., the quality of an image or user preferences, requires (i) subjective assessment and (ii) preference aggregation techniques, in order to combine the human annotations. Usually, these consist of either rating a set of items based on some criteria, or creating a ranking of a subset of the overall items. While the latter is better suited for crowd-sourcing scenarios as it is less complex for the annotators [34], compared to rating, it requires the inference of *latent scores*, that represent the position of the items in the rank, which involves comparison pairs samples.

The ranking task is defined as a comparison of $n$ items, evaluated using subjective features and without ground truth scores. The most straightforward experimental protocol is to compare pairs, referred to as 'pairwise comparison'; however, this approach requires too many evaluations, more precisely $\binom{n}{2} = n(n-1)/2$, where $n$ is the total number of items to rank. As a practical solution, the *active sampling* algorithm selects the most informative pairs to compare, thus reducing the number of total comparisons while maintaining good results.

*Active SAmpling for Pairwise comparisons* (ASAP) [35] is a state-of-the-art active sampling algorithm based on information gain, that finds the best pairs to compare in ranking experiments. The use of ASAP in this paper supports the annotators' work, since this algorithm minimizes the number of comparisons needed to obtain a complete ranking.

ASAP consists of four steps:

(i) starting with the annotated pairs collected so far, build the annotation matrix. The annotation matrix, contains the number of times an item has been selected over the other;

(ii) ASAP simulates all possible comparisons outcomes for the pairs;

(iii) the new rankings are created, using TrueSkill (described below), for each possible outcome;

(iv) using all possible ranking, find pairs that return the highest information gain and return them.

These steps become very computational demanding, therefore there is a need to optimize the process. Past active sampling solutions reduced the computational complexity with a sub-optimal approach, only updating the probabilities of the pairs selected for the subsequent comparison, which might not converge to the best optimum. Instead, the ASAP algorithm reduces the overhead by using *approximate message passing*, and it only computes the information gain of the most informative pairs updating the probabilities of all the pairs, thus making it efficient and correct.

ASAP uses TrueSkill [36] for the ranking of the annotated pairs. TrueSkill is a ranking system for evaluating players' relative skills in zero-sum games. TrueSkill is similar to Elo [37], one of the first algorithms developed for ranking in two-player games. Elo models the probability of the possible game outcomes as a function of the two players' skills represented as a single scalar. However, unlike Elo, TrueSkill uses Bayesian inference to evaluate a player's skill. Therefore, a player's skill is defined using a normal distribution $\mathcal{N}(\mu, \sigma)$, where $\mu$, the mean, is the perceived skill, and the variance $\sigma$ represents how uncertain the system is about the player's skill value. As such, $\mathcal{N}(x)$ can be interpreted as the probability that the player's 'true' skill is $x$.

TrueSkill, given its nature as an online game ranking system, supports the addition of new players, without needing to recompute pre-existing players' scores. This aspect is very important for our work too, since we aim to build an extensible classification of terms. Moreover, for pairwise comparisons, TrueSkill should be able to correctly rank newly added element with some 12 comparisons[9]. In the next section we validate whether this claim is correct in the context of our classification.

## 4. Proposed Approach

GitRanking is our proposed approach for generating a hierarchical taxonomy of software application domains. It is a bottom-up ranking, based on GitHub Topics and grounded in Wikidata. It aims to solve some of the antipatterns present in current datasets for software classification [6], including: *mixed taxonomies*, *mixed granularity*, and *ambiguity*.

In this section, we present the pipeline used to create the ranking of the GitHub Topics. The pipeline is visually represented in Figure 2, and its activities are described in more detail below.

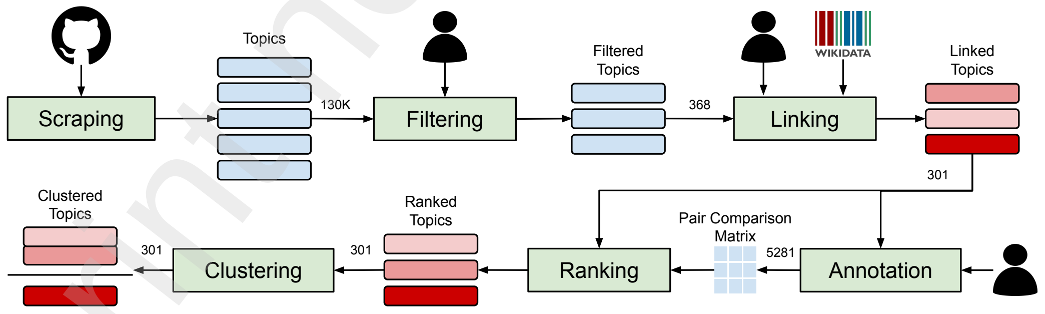

Figure 2: GitRanking's pipeline for creating the ranking of GitHub Topics.

[9]https://trueskill.org/#rating-the-model-for-skill

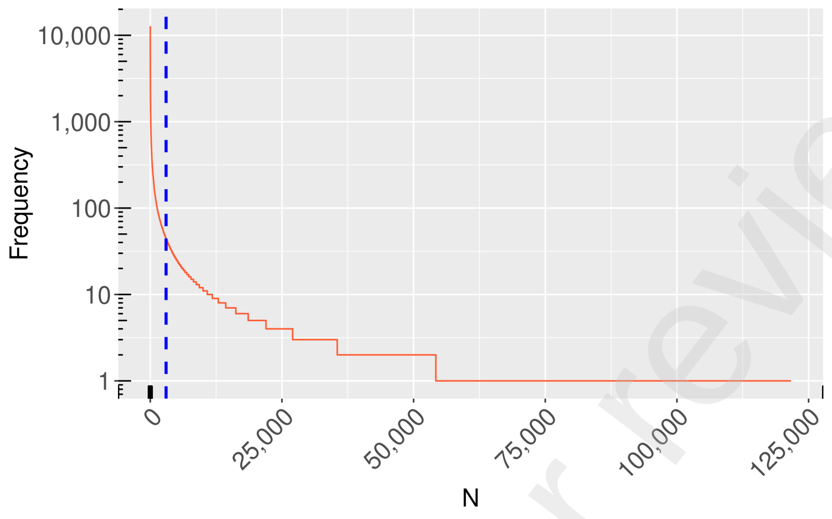

Figure 3: Distribution (log scale) of the frequency of the scraped topics from GitHub. The blue dotted line represents the cut line we picked for our initial filtering at $N = 3,000$.

## *4.1. Topic Collection - Scraping*

We collected the GitHub Topics by following the same approach used in [38]: we scraped repositories containing (1) at least one GitHub Topic, (2) a `README` file, (3) a description, and (4) at least ten stars. This way, we retrieved 135K projects, totalling 121K distinct topics, with a combined frequency of 1 million.

The distribution of the frequency and usage of the topics is highly skewed, as depicted in Figure 3, with around 50% of topics used only once. Less than 2,500 topics represent 50% of the distribution of use. The skewness is caused by many factors, including the widespread use of specific trends (e.g., '*Machine Learning*' as a topic), and the use of programming languages as topics (15 programming languages are listed in the top 50 topics, resulting in 7% of the frequency of the overall topics).

## *4.2. Topic Filtering*

In this activity, we attempted to reduce the number of topics by filtering and manual annotation. Given the large variety of terms, an automatic approach based on the semantics of the topics would be preferred. However, given the absence of a precise context, and the ambiguity of the task and terms, this approach is not optimal and will produce poor results. Therefore, we opted for manually annotating a subset of the topics as a solution.

For the annotation, we selected the 3,000 most frequent topics: this was a good balance between the number of topics and the number of examples for the topic. However, the cumulative frequency of the 3,000 topics is 60% of the total frequency, and each of those topics has a number of examples close to 50 (with a minimum of 44). Using more than 3,000 topics as a baseline would result in topics with less and less examples: this would make it very challenging for a classification model to learn good features for the class.

With the help of three annotators, we assigned a binary label (0, 1) to each of the 3,000 topics. The annotators were instructed to positively label (i.e., 1) the GitHub Topics that can be considered as software *application domains* (e.g., '*deep learning*', '*common line interface*', etc.). At the same time, the annotators were instructed to assign a null label (i.e., 0) for programming languages, companies, technology, or any other spurious case.

As the final step of this activity, the selection of the resulting topics was carried out using a majority voting (e.g., at least two annotators agreeing on a topic being an application domain); this was done to ensure higher quality in the creation of the initial taxonomy and to remove noise, while still allowing for a good recall. This activity resulted in 368 topics that can be considered as application domains: this set was carried forward to the reconciliation activity below in section 4.3.

## *4.3. Topic Linking*

The topics resulting from the filtering and the manual annotations are intrinsically ambiguous (e.g., '*rna-seq*', or '*ci*'); to help with the disambiguation of these terms, we linked each of the topics to Wikidata [39], Wikimedia Foundation's knowledge base. The linking is performed in a semi-automatic way, by using the Wikidata reconciliation API, and humans to check and fix any errors.

Wikidata offers a reconciliation API, a service that, given a text representing a name or label for an entity (and optional, additional information to narrow down and refine the search to entities), returns a ranked list of potential entities matching the input text. As per the examples above, the API returns the '*RNA sequencing*' entity (with Wikidata ID `Q2542347`[10]) for the '*rna-seq*' topic; and the '*Continuous Integration*' entity (ID `Q965769`) for the '*ci*' topic. The reconciliation uses fuzzy matching to find the most likely entity in the knowledge base that matches the input string. Hence, the candidate text does not have to match each entity's name perfectly, meaning that we transform ambiguous text names into precisely identified entities within a knowledge base.

The topics resulting from the previous activity were fed as an input to the Wikidata API: to increase the retrieval precision, we exploit the *github-topic*

[10]https://www.wikidata.org/wiki/Q2542347

Wikidata property, with ID `P9100`[11], that helps in the linking of terms that are already linked in Wikidata (e.g., the entity '*Convolutional Neural Network*', has an entry with property `P9000`, where the value is '*convolutional-neural-network*').

This reconciliation activity returns a list of 10 candidates for each term. Each candidate has a list of types describing the candidate: e.g., for *'Science'*, there are various candidates with the same name, but different types, some include: '*academic discipline*', which would link to the correct entity `Q336`, and the other will link to '*television channel*' with ID `Q845056`. Next, we manually annotated the highly irrelevant types to the task (e.g., *human*, *television channel*, or any location) and exclude candidates belonging to these types for a more automated process of linking. After filtering the candidates of an irrelevant type, we link the term by picking the first candidate in the filtered list. Lastly, we check the correctness of the linking and fix improperly linked topics. This resulted in the correction of 25 topics out of 368.

Now that our topics are disambiguated, we can use the unique ID from Wikidata to reduce duplicates, as some topics are just different *surface forms*, or aliases, for the same entity. The number of unique topics left after the removal of duplicates is 301.

## *4.4. Annotation*

For the creation of the ranking, the algorithm requires annotated pairs of topics. In this activity, the final list of 301 topics were presented in pairs on a web application for the manual annotation. The 8 annotators working on

[11]https://www.wikidata.org/wiki/Property:P9100

this activity were presented with two topics in the list and were instructed to pick *the most general term*, considering their respective domains. A mock-up of the interface is illustrated in Figure 4. The terms were also linked to the URLs of the Wikidata pages in case the annotators were not confident with a specific topic.

In case of doubt, the annotators could skip the pair. The *'Tie'* option was also available, in case the annotators believed that the two terms represent domains of the same hierarchical level. This option was also used to collect data to validate our results: since ASAP does not support *'Ties'*, we instructed the annotators to use it rarely. Instead, annotators were instructed to rather pick one of the options randomly, as having one term before the other in the continuous ranking does not affect the final discrete rank.

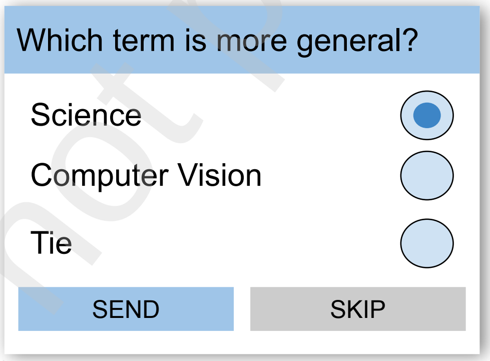

Figure 4: The user interface presented to the annotators. The skip bottom allows the annotators to obtain a new pair when they are not confident with the current.

Given the large amount of annotated pairs required to create a ranking, we used the ASAP algorithm to assist the work of the annotators: its main advantage is to reduce the number of pairwise annotations and still achieve a good performance. The ASAP paper [35] shows in fact how, on an example with 200 items, the performance is affected when one reduced the number of comparisons. In their case, a reduction in the number of comparisons by a factor of 3 shows a good balance in terms of performance, resulting in a total of $\frac{1}{3}\binom{n}{2}$ annotations needed.

The ASAP algorithm identifies the topic pairs and uses the previous annotation to find the best, most informative, new pairs to add to the list of annotations. The annotators are then presented with a random pair from this pool. The ASAP algorithm is a very memory-intensive task; we used an AWS instance with 64 GB of RAM and a 16 core CPU for our experiments.

In terms of effort required for our annotation, and given the 301 terms we collected and an estimated amount of 25 seconds per pair on average, we expected the overall time required to be $\frac{1}{3}\binom{301}{2} * 25/3,600 = 104$ hours. Given our pool of 8 annotators, this would translate to 13 hours for each annotator. However, we would also expect fewer samples required for our case, since this task is less challenging as compared to modelling more abstract values like a player skill in an online game.

## *4.5. Ranking*

The final ranking of the topics is computed by ASAP using TrueSkill. It uses the comparison matrix created with the annotations at the previous step, excluding the items marked with *'Tie'*. The comparison matrix is a square matrix, where every entry is the number of times the term in the corresponding row was selected over the term in the corresponding column. This results, for each topic, in a mean and a standard deviation value. The mean represents the position in the ranking of the topic; the standard deviation is the confidence that the model holds in its prediction.

For example, given the terms: '*Science*', '*Computer Science*', and '*Software Engineering*', and a comparison matrix, the final ranking returned by TrueSkill will be:

- *Science*: $\mathcal{N}(2, 0.3)$;
- *Computer Science*: $\mathcal{N}(1.5, 0.3)$;
- *Software Engineering*: $\mathcal{N}(0.9, 0.3)$.

### *4.6. Clustering*

Lastly, the final step of our pipeline is to create a discrete rank of the topics. Using the resulting ranking of ASAP, we feed the mean of the topics ranking as an input to the *KMeans* clustering algorithm. To find the optimal number of clusters, we used the Elbow method. The clustering is performed on the uni-dimensional data of the topics' mean score computed by TrueSkill.

Having a discrete set of ranks instead of a continuous one brings many possibilities: perform analysis to study developers' behaviour, and use the discrete values to train models to predict topics at specific levels, aiding with the annotation of the repository.

### *4.7. Ranking New Topics*

One important feature of a taxonomy is adapting to the domain's evolution. Our pipeline allows for such adaptation, with the addition of new terms. In this section we show how relatively easy it is to extend our taxonomy.

We evaluated the number of annotations required to rank newly added topics. The experiment uses the annotations collected from the experiments with annotators. We simulate a new topic addition by removing their annotations and incrementally adding them, one by one, and checking for convergence. We measure the average number of annotations required to reach convergence for all 301 topics by individually removing each one.

We compare three different strategies of simulating the newly inserted topic: **random**, **order**, and **informed**. The **random** strategy samples annotations of the topic in a random fashion; the **order** strategy selects the pairs in order as annotated and suggested by ASAP; the **informed** one uses only the last 20 pairs suggested by ASAP, making it a more efficient, but not optimal way, to simulate the new annotations.

Convergence is defined as being in proximity of the position held by the topic in the ranking created with all the annotations at our disposal. We use a max of 3 positions difference for proximity, and convergence is reached when the proximity is held for two consecutive annotations.

## 5. Results

In this section, we present the results of our approach and discuss them. We show the statistics of the filtering and annotation. We also show the ranking and samples from it. Lastly, we show the results of adding a new term to the ranking.

### *5.1. Filtering*

Starting from the 3,000 topics, that cover 60% of the total topics distribution, the final list analysed in this step by the three annotators contains 368 topics.

Table 1 shows the number of positively labelled topics from each annotator, with their positive rate. The three annotators had different ideas of what an application domain is. Annotator **C** was more strict, picking a minimal amount of terms, **A** was more relaxed picking many, with **B** in between. Furthermore, we measured the inter-rater reliability using Krippendorff's alpha [40], a general, reliable measure for inter-rater reliability [41] suited for any number of annotators. For the 3,000 topics and our three annotators, we obtained an agreement score of 0.68. If we measure it per pair of annotators, we have an agreement of 0.79 for pair **A-B**, 0.68 for the **B-C** one, and 0.52 for pair **A-C**.

The low amount of GitHub Topics that qualify as application domains suggests that using 'popularity' as a seed for a taxonomy (similarly to how previous research works have done in the past), results in low quality labels.

Table 1: Number of positive labelled topics for each annotator and their positive rate.

| **Annotator ID** | **Positive** | **Positive Rate** |
|---|---|---|
| A | 496 | 0.16 |
| B | 370 | 0.12 |
| C | 238 | 0.08 |

Takeaway 1

Using 60% of the most frequent topics as seed allows us to reduce bias in the representation of the less common topics. The filtering avoids the presence of terms that are not application domains (e.g., programming languages). Both issues are common in previous works.

## *5.2. Annotation*

From the pairwise comparison data annotation, we collected 5,281 annotated pairs from 8 annotators, including Professors, PostDocs, and PhDs in Computer Science with a mix of Software Engineering and Machine Learning backgrounds. The statistics about the annotators' contribution to the process are presented in Table 2.

As we expected, we were able to converge to a stable rank in less than the $\frac{1}{3}\binom{301}{2} = 15,050$ that we expected annotations, and similar to the case study in the ASAP paper. From Figure 5, which shows the average change in position at incremental amounts of annotations, we can see that the average change in positions in the ranking converged at around $\frac{1}{9}\binom{301}{2} = 5,000$ annotations.

The curve has a steep decrease in the first 1,000 comparisons, and later plateaus at an average of 10 positions changed by the terms in the ranking. After 5,000, it immediately falls to an average close to 0.

Table 2: List of the annotators' IDs and their contribution to the total annotations with the number of ties assigned.

| **Annotator ID** | **Annotations** | **Ties** |
|---|---|---|
| 1 | 1,207 | 117 |
| 2 | 454 | 43 |
| 3 | 1,520 | 11 |
| 4 | 94 | 0 |
| 7 | 561 | 82 |
| 8 | 246 | 27 |
| 9 | 764 | 89 |
| 10 | 190 | 13 |
| **Total** | 5,281 | 382 |

Takeaway 2

ASAP reduces the amount of annotation required to reach convergence in the ranking by a factor of 9, making it an effective way to aggregate domain expertise in qualitative ranking tasks.

## *5.3. Ranking*

The final ranking is presented in Figure 6, where we see the topics' mean score computed by TrueSkill, and their position in the final rank. We can notice that, while the extremity of the ranking is very well separated, the central area is not as much. This is caused by the higher difficulty of comparing

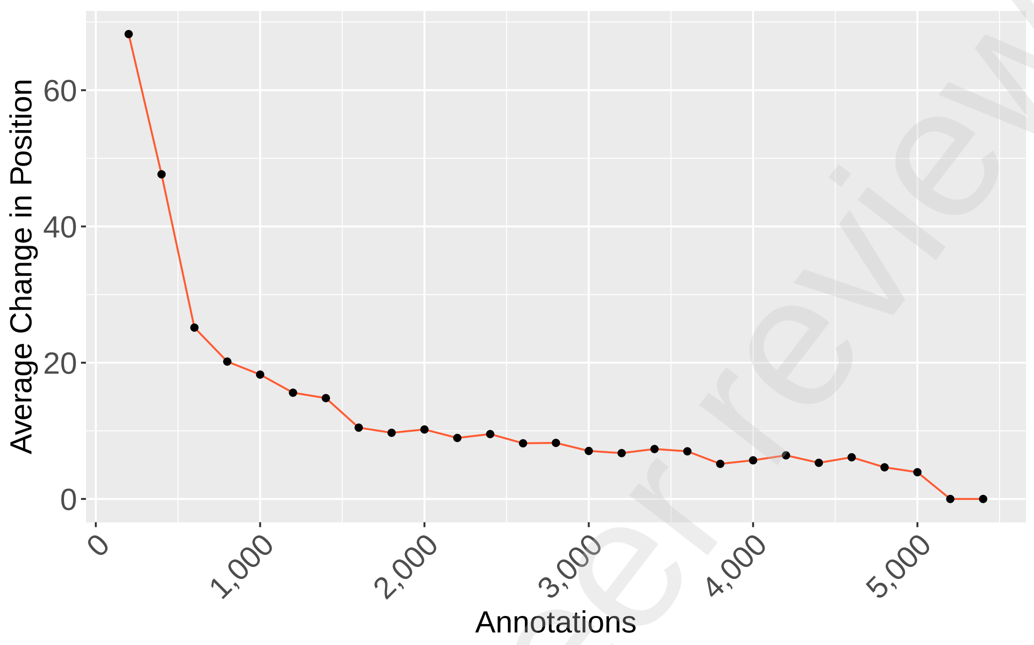

Figure 5: Average change in position of the terms every 200 new annotations.

topics belonging to the middle area of a taxonomy.

A more qualitatively view of the resulting ranking is presented as a sample of the topics at different levels in Table 3. The Table shows a vertical view of topics that would belong to the same branch in a taxonomy. In particular, it presents the Artificial Intelligence / Computer Vision branch, from the top most general term to middle terms, all the way to the last terms in such a branch. From Table 3, we notice that at the top, we have terms that we would come to expect with '*Science*' being the first one. As we go down the ranking, the terms get more specific, first '*Computer Science*' followed by '*Artificial Intelligence*'. After these, we find a limit case, where we have two terms '*Computer Vision*' and '*Machine Learning*' that might need to be reversed; for someone, they are correct, and for others, they should be at

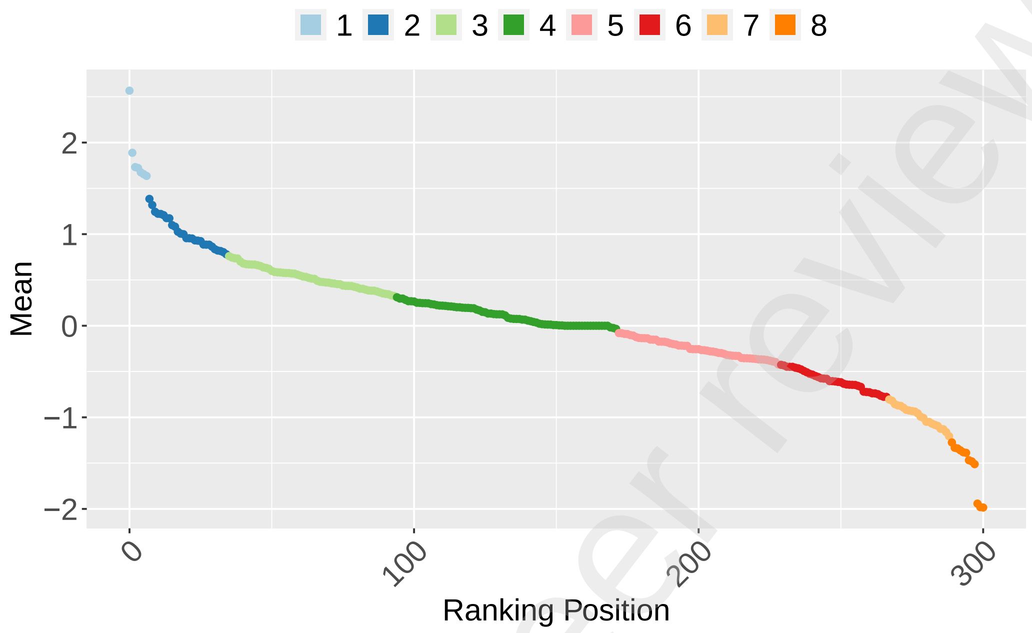

Figure 6: Ranking of the GitHub Topics. The $x$-axis represents the position, while the $y$-axis is the mean score extracted by TrueSkill. The color represents the cluster that each topic is assigned to.

the same level. As we get closer to the end, we find more concrete tasks like '*Image Segmentation*'. At the bottom, we see methods like '*Convolutional Neural Network*'.

The final list can be checked in the data replication package (see Section 1).

Takeaway 3

TrueSkill's ranking captures the hierarchical relations among terms. However, there is still room for improvement at the middle levels as the separation is not as strong.

Table 3: Rank of a subset of terms in the vertical of Computer Science, Machine Learning, and Computer Vision.

| **Rank** | **Topic** |
|---|---|
| 1 | Science |
| 2 | Mathematics |
| 3 | Physics |
| 4 | Engineering |
| ... | ... |
| 9 | Computer Science |
| ... | ... |
| 34 | Artificial Intelligence |
| ... | ... |
| 60 | Computer Vision |
| 61 | Machine Learning |
| ... | ... |
| 147 | Image Segmentation |
| ... | ... |
| 295 | Convolutional Neural Networks |
| ... | ... |

## *5.4. Clustering*

Using the Elbow method for KMeans, we found the optimal number of clusters for our ranking at $n = 8$. In Figure 6, we can see the topics' ranking distribution and their cluster.

We evaluate our clustering using the *'Tie'* labelled pairs from the annotation phase. We measure how many of the *'Tie'* annotated pairs end up in the same bucket in the cluster. Out of the 382 ties, 364 were unique, and almost evenly distributed among the eight categories. The results show that 30% of the tied pairs belong to the same cluster. When loosening the constraint of equality and setting a distance of 1 cluster, we reach 100% accuracy of our clustering. This result suggests that overall our method is effective; nevertheless, many cases are not precisely placed at the correct level. However, if we also take into account the results in Table 3, we can see that terms in the same vertical are correctly ordered. Still, across verticals, there might be less order, which is in line with the objective of the ranking: *create a sorting of terms that belong to the same domain.*

## *5.5. Topic Ranking Distribution*

After obtaining our ranking, we are able to answer:

> ***RQ1****: Are the topics used to annotate GitHub projects* ***evenly distributed*** *in the levels of a taxonomy?*

To answer this, we can use Figure 7, which shows the distribution of topics at different levels in the ranking. The top bar chart shows that the topics are normally distributed across the levels, with the mean right in the middle at level four. However, if we consider the topics' frequency (bottom plot), the mean moves towards a higher, and more general level, level three.

This suggests a tendency on the developers to use a general term that only describes the application area but without specific information. This negatively impacts the retrieval of projects and affects the time required to find the appropriate repository, as not many are labelled with a more specific term.

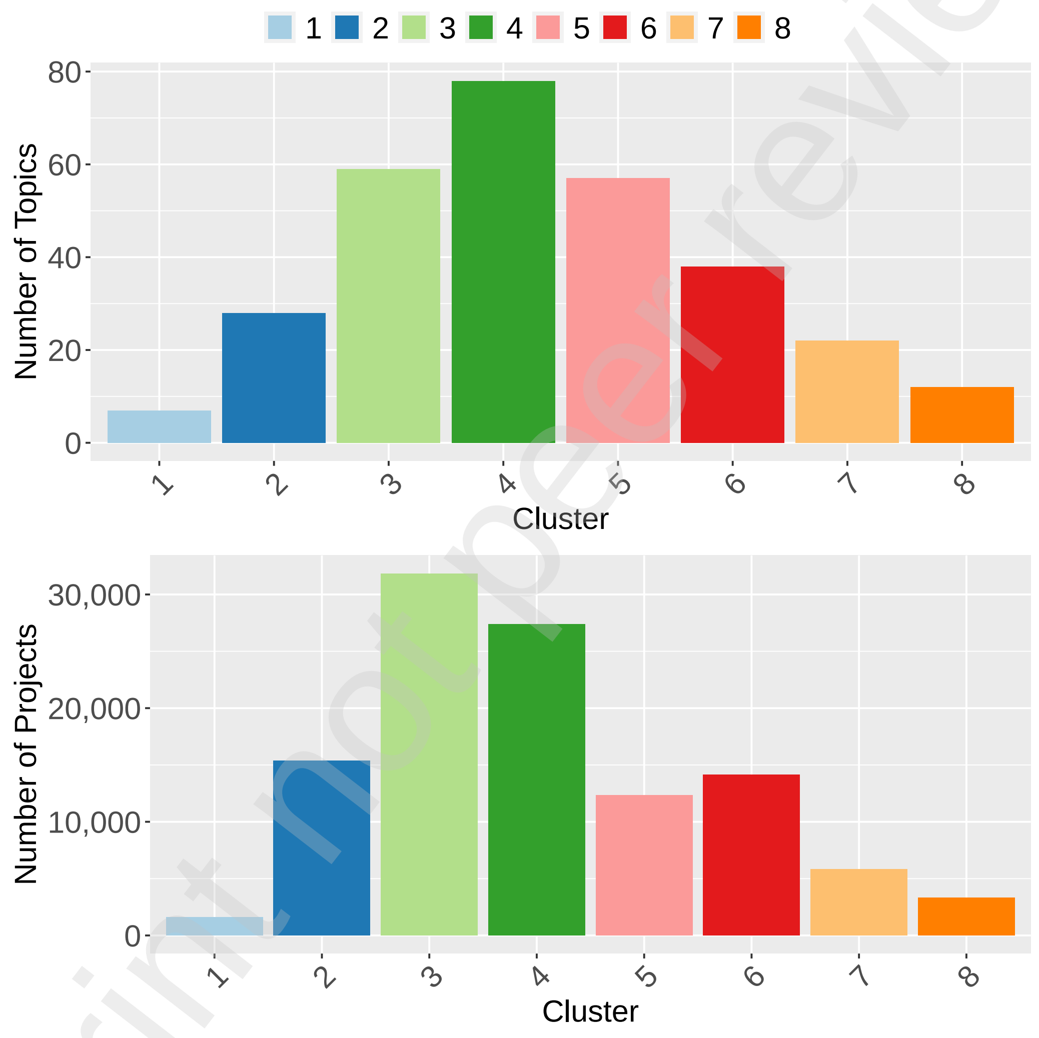

Figure 7: (Top) Distribution of the topics in the clusters. (Bottom) The number of projects (frequency of the topics) in each cluster. A low cluster number means more general, and a higher value means more specific.

Takeaway 4

There is a lack of specificity in the terms used by developers to annotate their projects. Future work in software classification needs to address this issue by suggesting topics at multiple levels.

## *5.6. Ranking New Topics*

Using the different approaches to simulate annotations for a newly added topic defined in Section 4.7 ('random', 'order' and 'informed'), we measured the average amount of annotations required to reach convergence.

The results are presented in Table 4. We can notice that, independently of the scenario, the amount of annotations required is minimal, with a range from 22, in a non-optimized scenario, to around 15 when using a better approach (e.g., 'informed'). These are more than the ones suggested by TrueSkill. However, the difference is negligible if we consider that the selection of the pairs was not online, as it would be for new topics.

This shows how easy it is to extend the classification to keep it up to date or adapt it to new subdomains. Furthermore, the required amount of annotation scales linearly with the number of new topics for batch additions, as the problem can be viewed as multiple single-term increments.

Takeaway 5

Our pipeline is flexible and allows for the insertion of new terms with minimum effort, making our taxonomy a good starting point to build upon.

Table 4: Average number of annotations required to reach convergence when adding a new term in the ranking.

| Method | Average |
|---|---|
| random | 22 |
| order | 22 |
| informed | 15 |

## 6. Classification

As a catalyst for further work on the task of software classification, we also release a new dataset. In this section, we will present the details of our dataset, from the statistics to the baseline performances using a state-of-the-art approach. The pipeline is presented in Figure 8.

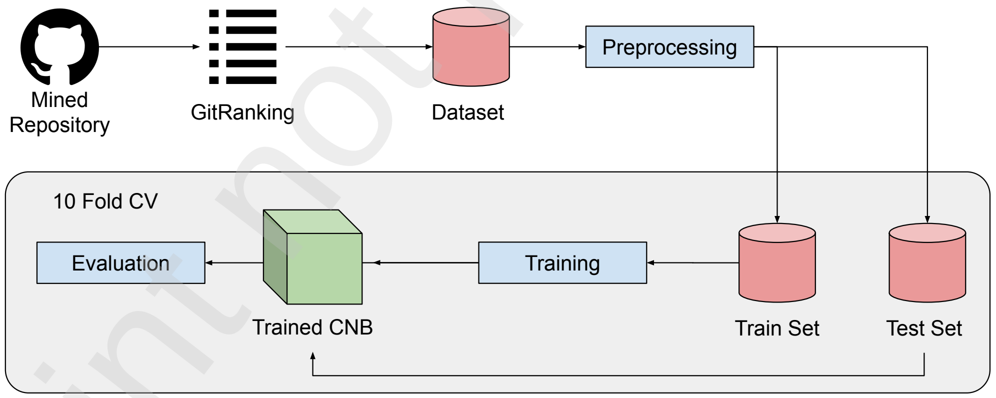

Figure 8: The classification process.

## *6.1. Dataset*

A clear advantage of using GitHub Topics as the source of our classification is the possibility of continuously having an access to a pool of projects that can be added to our dataset.

We start with the projects that have at least one topic in our curated list of 301. We perform the scraping of the `README` content from the GitHub repository of the projects. Furthermore, we expand the mapping list of topics by looking for alternate forms used for the same topic. As an example, the topic '*computer vision*' is expressed with the following alternative forms contained in the list of GitHub Topics: '*computervision*', and '*computer-vi*'. We perform this step for the topics that have not been considered in the selection process above as a way to increase the number of examples available for the classification model.

The expansion is performed using the Wikidata Reconciliation API. Using the same Topic Linking step in our pipeline, described in Section 4.3, for each topic we look for the first candidate having a Wikidata ID that matches one in our taxonomy. Furthermore, in order to reduce noise, we add a constraint regarding the similarity between the topic and the entity using an edit distance of 0.75 or higher. The mapping results in an increase of 679 extra GitHub topics being mapped in our dataset, with a total of 8013 extra annotations in the dataset.

The dataset is available as a JSON file in the replication package (Section 1), and has the following schema:

- **full_name**: the name of the repository in the format *author/repository*;
- **readme**: the `README` HTML section;

- **text_readme**: the text content of the the `README`;
- **topics**: the original GitHub Topics that the author(s) of the repository used to annotate the project;
- **labels**: the labels proposed from our taxonomy. These are annotated by mapping the original topics into our labels, while ignoring the topics that do not have a map into a label;
- **levels**: the levels for the labels in our ranking;
- **description**: the project description written by the authors in the project GitHub page.

The final dataset consists of 48,000 examples. In Figure 9, we can see the distribution of examples per topic. The 'deep learning' and 'machine learning' are by far the labels that appear most often in the examples of our dataset.

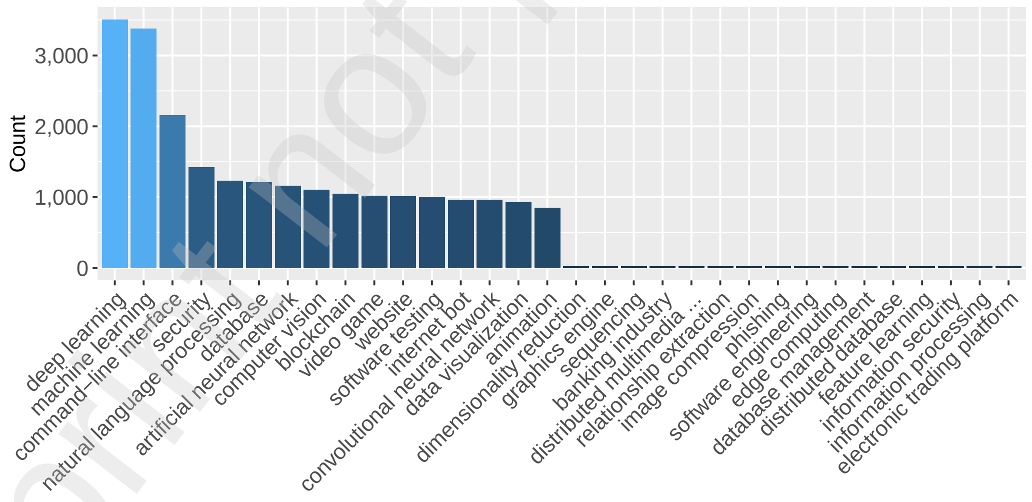

Figure 9: Distribution of examples for the 15 most and least frequent labels in the dataset. The least popular classes have around 20-30 examples.

Other statistics of the dataset are presented in Table 5.

| **Statistic** | **Value** |
|---|---|
| Examples | 48,000 |
| Annotations | 80,028 |
| Avg Num Label per Project | $1.65 \pm 1.07$ |
| Avg `README` Length (words) | $540 \pm 841$ |

Table 5: Various statistics for our dataset. The number of 'examples' is the total number of 'projects'; the number of annotations is how many labels are present in total - that is the sum of the number of labels for each project.

### *6.2. Experimental Setup*

To measure the benefits of adopting the proposed taxonomy in the classification task, we evaluate the results obtained by running an existing approach [4] on the dataset presented in Section 6.1 above. This state-of-the-art approach classifies Github repositories using a probabilistic network given the corresponding README file. We will provide further details in the next subsections.

#### *6.2.1. Pre-processing*

The `README`, used as an input for the model below, is pre-processed by performing the removal of stop words and the stemming of the remaining words. After the pre-processing phase, a TFIDF approach is used on the resulting bag of words: TFIDF is a feature representation technique for textual data, it is computed as the product of two statistics, the term frequency

(TF) in the document, and inverse document frequency (IDF) which is the total frequency of that term in all documents. The resulting representative vector is given as input to the model, which returns a probability score for each label.

### *6.2.2. Model*

In our prior work [4], we used the Multinomial Naïve Bayes (MNB) using the `README` file of the repository as input. The Naïve Bayesian network is a probabilistic model based on the Bayesian theorem that expresses the probability of a certain event given a set of preconditions. In this paper, we adopt a different implementation of the Bayesian network, called Complement Naive Bayesian (CNB) [42], capable of handling this issue by using a more even amount of training data per class. The rationale behind this choice is two-fold. First, the employed dataset described in Section 6.1 is unbalanced, that is, the `README`s are not equally distributed for the topics. Second, the CNB has been successfully employed to categorize textual documents in the existing literature [43, 44]. Formally, the CNB computes the predictions according to the following formula:

$$\begin{aligned}\hat{\theta}_{ci} &= \frac{\alpha_i + \sum_{j:y_j \neq c} d_{ij}}{\alpha + \sum_{j:y_j \neq c} \sum_k d_{kj}} \\ w_{ci} &= \log \hat{\theta}_{ci} \\ w_{ci} &= \frac{w_{ci}}{\sum_j |w_{cj}|}\end{aligned} \tag{1}$$

where the summations are over all documents $j$ (not present in class $\mathbf{c}$), $d_{ij}$ is the TFIDF value of term $i$ in document $j$ and $a_i$ is a smoothing parameter used to compute the final weight $w_{ci}$. The CNB eventually retrieves the list of recommended topics ranked by their probability.

### *6.3. Evaluation*

To assess the results, we performed a ten-fold cross-fold validation, a well-known strategy adopted to evaluate classifiers [45]. Starting from the mined repositories, we apply GitRaking to filter the original set of topics as described in Section 4. Afterwards, these repositories annotated with the new topics are split into training and testing sets. To preserve a distribution along with all the rounds, we used 90% and 10% of the files for training and testing, respectively. Afterwards, the CNB is fed with the training repositories and the corresponding topics and predicts the topics by analysing the `README` content of each project. The predicted tags are eventually compared with the testing ones in terms of the identified metrics.

Furthermore, we inspected the performance of the model at the various levels of the classification as identified GitRanking. This analysis is aimed to understand if the model has any performance loss or gain based on the levels, hence aiding with answering **RQ2**: *Does the classification models **performance** vary across different levels in a taxonomy?*

#### *6.3.1. Metrics*

Following the same evaluation conducted in [4], we compute the precision, recall, and F1-score by comparing the predicted labels for the project with the ground truth ones. To define these metrics we first need to define the following concepts:

- **True positive** (Tp): the topics that are correctly predicted;

- **False positive** (Fp): the predicted topics which actually do not belong to the ground-truth data;
- **False negative** (Fn): the topics that should be present in the predicted items, but they are not.

Using these, we can define our metrics as follows:

**Success rate:** Given a set of $R$ testing repositories, this metric measures the rate at which a recommendation engine can return at least 1 correct tag for each repository $r$.

$$Success\ rate = \frac{count_{r \in R}(Tp > 0)}{|R|} \times 100\% \tag{2}$$

where the function *count()* is a boolean function that returns 1 for each true positive.

**Precision:** the metric measures the rate of correct items over the entire set of recommended items:

$$precision = \frac{Tp}{Tp + Fp} \tag{3}$$

**Recall:** the ratio of the user's topics appearing in the N recommended items:

$$recall = \frac{Tp}{Tp + Fn} \tag{4}$$

**F1-score:** it represents the harmonic mean of the precision and the recall

$$F1\text{-}score = 2 * \frac{precison * recall}{precision + recall} \tag{5}$$

We show these results for the whole dataset, furthermore, we evaluate the model performance at the various levels of the classification to evaluate the quality of predicting specific labels that help with discoverability.

## *6.4. Results*

In this section we present the results of the classification task obtained from the performed experiments. The first experiment consists in training the CNB model on the full dataset to set a baseline for future research. The results are presented in Figure 10, where we can see the performance for the ten-fold cross-fold validation. In the scope of the evaluation, we set the number of recommended items N=3 since most of the considered repositories have a similar number of topics by using our GitRanking approach.

From running the CNB model, we noticed that CNB overall has a high *Success rate* (66%), meaning that for the most part it is effective at predicting a correct label. While the *Precision* is close to the *Success rate* (55%), the model suffers of low *Recall* (27%). This means that the model returns very few results, but most of its predicted labels are correct when compared to the real ones. The *F1-score* is 34%.

Lastly, we assess the accuracy of the employed classifier for the various levels as identified by GitRanking to answer **RQ2**. The results are presented in Figure 11. We can see that there is no obvious pattern regarding the performance of the model; however, if we measure the correlation with the number of labels in the level we get a correlation of -0.80, meaning that the performance is more influenced by the number of labels rather than the level itself. This trend is also found in the performance of models trained only on single-level data, as seen from Figure 12.

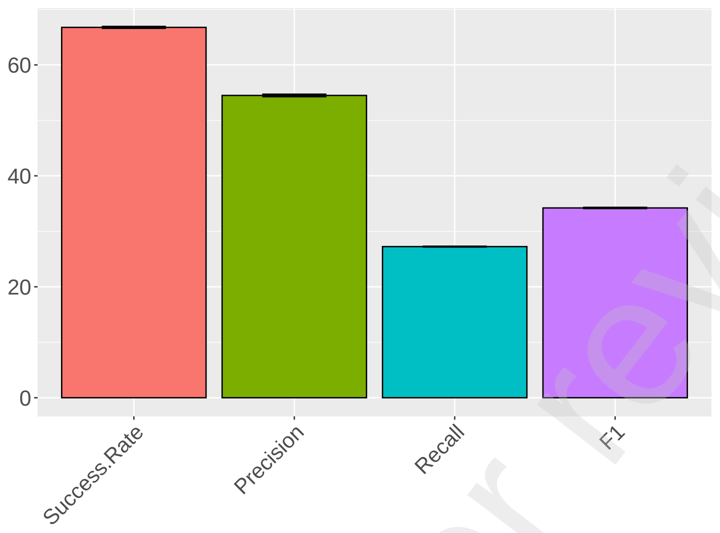

Figure 10: Overall performance of the models trained on all labels. The confidence bars show very little variance in the results.

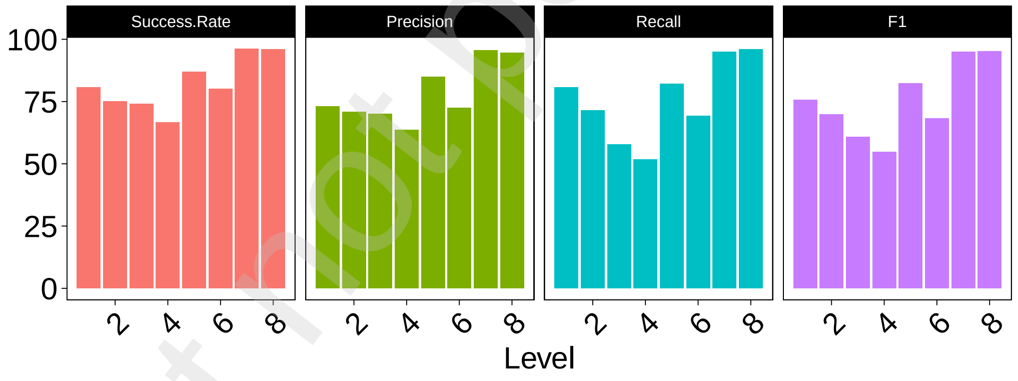

Figure 11: Results metrics for the model trained on all labels separated per level.

## 7. Discussion

In this section, we discuss what are the unique features, in our opinion, of the framework that we have presented. We also discuss the repercussions of the choices made and what implications should be expected. Lastly, we

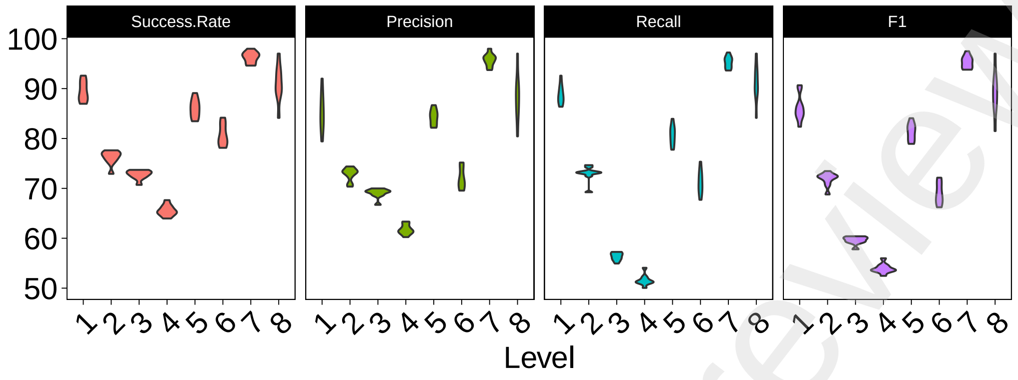

Figure 12: Results for the models trained only on the labels of the level.

discuss the findings of the classification.

*Unique features.* Our exploration of a bottom-up, data driven taxonomy has shown that the labels used by developers in their everyday work can form a solid starting point for a taxonomy. Although this has been attempted in the past, we believe that our work adds at least two unique features to this quest: first, the approach that we developed is based on a *seed* that was annotated by 8 experts, and whose provenance is directly rooted in the development of thousands of active GitHub projects. The annotation part, albeit time-consuming and process-intensive, is a necessary factor for the quality of the seed: this activity has been mostly absent in all the past works that we have analyzed for reference.

*Extensibility.* The second unique feature that this work offers to the research community is a flexible and dynamic approach to *expand* the taxonomy to further terms and labels. All the existing taxonomies can be considered as flexible, in the sense that they allow for further terms to be included. The added value in our work is that GitRanking allows to dynamically allocate a new label by means of further annotations by anyone proposing such a label. For example, if a researcher wanted to add a new label, not previously present in our taxonomy, they would be expected to run some 15 pairwise comparisons. That would be necessary and sufficient to locate the new term in the correct place of our taxonomy. The ability to add new terms is crucial, as the field evolves and new terms might become popular quickly, or someone wants to adapt a taxonomy with some terms of interest that are not currently present.

*Initial seed.* For our study we decided to use as a starting point the top 3,000 topics by frequency. This sample covers 60% of all the labels present in the overall set of 135K projects, and these labels do not include less popular or underrepresented topics. One might argue that this was the reason of our results regarding the distribution of the topics in the levels: the less popular are also the more specific terms. However, we perform the ranking and clustering on the topics available, hence the results would be even more skewed towards higher levels, if we considered the less frequent topics as well. Furthermore, the further down the list we go, the more noise and duplicates we find, making it more time consuming for the annotators.

*Practical applications of the taxonomy.* Taxonomies and classifications have an inherent utility in organizing the knowledge base around a specific area of expertise. In our case, we believe that having such a classification can be further used by GitHub to guide the developers in labelling their projects with at least one term for each level. This would be similar to the description of an academic paper using the ACM Computing Classification System, which allows researchers to choose from high-level and lower-level topics to describe their work.

*Improvements.* While our framework has shown its ability to create a good ranking of the selected terms, in Figure 6, as mentioned in the Results section, the separation at the middle layer is not as strong as we would have liked. This could be addressed by collecting more pairwise comparison data for only the middle area; however, contrary to the expectations, this might not be the case as the middle layer is also the hardest to separate for humans. One solution could be to perform linking among terms, resulting in a *tree-like* taxonomy. However, this is also not as trivial and requires more research.

*Classification.* The classification allowed us to have an overview of the performance of the state-of-the-art model over our new dataset. If we consider the results from **RQ1** (developers do not use specific terms to annotate their projects), with the findings of **RQ2** (our model performs independently of the level of ranking), we have a strong argument for the utility of classification models for software.

## 8. Threats to Validity

We use the classification of Runeson et al. [46] for analyzing the threats to validity in our work. We will present the *construct validity*, *external validity*, and *reliability*. Internal validity was not considered as we did not examine causal relations [46].

### *8.1. Construct Validity*

The first construction threat to our study is the initial filtering of the topics, which is highly subjective. However, we reduced this threat by having three annotators, and by only choosing the topics agreed on by at least two of them. For the ranking step, while we have a subjective input, the algorithm combines the input from all 8 annotators, reducing the weight of annotation errors. Furthermore, our ranking is open to evolution; by having a flexible pipeline, additions and changes can be done with minimal effort, which can be open to the community.

Regarding the analysis on the distribution of the ranking, and how developers label their projects, we mitigated this threat by using the validation data collected from the annotation phase.

Lastly, for the classification step, we reduce the construction threat by performing 10-fold cross-validation and have a large sample size in the dataset.

### *8.2. External Validity*

Our approach is independent of the domain, as it uses general methodologies from statistics and machine learning. Furthermore, we focus on words, making this approach applicable to all domains, not just software application domain ranking.

### *8.3. Reliability*

For the initial selection of projects, we collected a large amount of different projects, resulting in a large pool of terms, making the collected pool a representative sample of the population. Regarding the filtering, as discussed in the Construction Validity section, working with natural text is inherently subjective; to minimise the threat, we focused on having a robust filtering of the topics used in our study.

## 9. Conclusions and Future Work

Categorizing GitHub projects has recently attracted the attention of the research community, and leverages approaches and tools that cope with this task in an automatic way. Despite this, several challenges remain uncovered, such as the detection of antipatterns that are misleading during the definition of the topics. This paper presented GitRanking, a framework for creating a discrete ranking of software application domains. Our work aims at solving some of the common issues present in current datasets for software classification, including: *mixed taxonomies*, *mixed granularity*, and *ambiguity*.

Using GitRanking, we analyzed the most used topics of a large sample of GitHub Topics, covering 60% of the total frequency, and we selected a list of 301 application domains. We then disambiguated each topic by linking them to the Wikidata knowledge base. Furthermore, aided by the ASAP active sampling algorithm, 8 annotators compared more than 5,000 topic pairs: finally, the TrueSkill algorithm used those annotated pairs to create a ranking of the selected application domains. The ranking orders the topics from the most general to the most specific, allowing the creation of a hierarchy between topics. GitRanking's pipeline allows the creation of a taxonomy that is free of most classification antipatterns found in previous works.

To evaluate the proposed taxonomy, we analysed the developers' behaviour during the manual annotation task. By performing clustering of our ranking, we were able to find that developers tend to assign more high-level labels to their projects, making it harder to find specific projects.

Furthermore, we employed GitRanking to study how a state-of-the-art classifier performance are affected by the level of the labels. The results show that the models are not affected by the level, but only by the number of labels. This shows the utility of these models in aiding with the classification at all levels of a taxonomy.

We plan to improve our work in different ways: first, we aim to create a hierarchical taxonomy and link the terms in our ranking. Moreover, we are interested in focusing more on the topics that have not been taken into account by our current solution: this will focus both on adding new terms and also on improving the synonyms of terms that we already have in our taxonomy. Lastly, we would like to evaluate the classification using the source code as the main artefact for the project: this will allow to expand the ability to classify projects also to those where the `README` is not available, or is not very descriptive.

## References

[1] A. Sharma, F. Thung, P. S. Kochhar, A. Sulistya, D. Lo, Cataloging github repositories, in: Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering, EASE'17, Association for Computing Machinery, New York, NY, USA, 2017, p. 314–319. doi:10.1145/3084226.3084287.

[2] S. Vargas-Baldrich, M. L. Vásquez, D. Poshyvanyk, Automated tagging of software projects using bytecode and dependencies (N), in: M. B. Cohen, L. Grunske, M. Whalen (Eds.), 30th IEEE/ACM International Conference on

Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015, IEEE Computer Society, 2015, pp. 289–294. doi:10.1109/ASE.2015.38.
URL https://doi.org/10.1109/ASE.2015.38

[3] Y. Zhang, F. F. Xu, S. Li, Y. Meng, X. Wang, Q. Li, J. Han, Higitclass: Keyword-driven hierarchical classification of github repositories, in: J. Wang, K. Shim, X. Wu (Eds.), 2019 IEEE International Conference on Data Mining, ICDM 2019, Beijing, China, November 8-11, 2019, IEEE, 2019, pp. 876–885. doi:10.1109/ICDM.2019.00098.
URL https://doi.org/10.1109/ICDM.2019.00098

[4] C. Di Sipio, R. Rubei, D. Di Ruscio, P. T. Nguyen, A multinomial naïve bayesian (MNB) network to automatically recommend topics for github repositories, in: Proceedings of the Evaluation and Assessment in Software Engineering, EASE '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 71–80. doi:10.1145/3383219.3383227.
URL https://doi.org/10.1145/3383219.3383227

[5] Y. Zhou, J. Wu, Y. Sun, Ghtrec: A personalized service to recommend github trending repositories for developers, in: C. K. Chang, E. Daminai, J. Fan, P. Ghodous, M. Maximilien, Z. Wang, R. Ward, J. Zhang (Eds.), 2021 IEEE International Conference on Web Services, ICWS 2021, Chicago, IL, USA, September 5-10, 2021, IEEE, 2021, pp. 314–323. doi:10.1109/ICWS53863.2021.00049.
URL https://doi.org/10.1109/ICWS53863.2021.00049

[6] C. Sas, A. Capiluppi, Antipatterns in software classification taxonomies, Journal of Systems and Software 190 (2022) 111343.

doi:https://doi.org/10.1016/j.jss.2022.111343.
URL https://www.sciencedirect.com/science/article/pii/S0164121222000826

[7] R. Caruana, Multitask learning, Machine Learning 28 (1) (1997) 41–75. doi:10.1023/A:1007379606734.
URL https://doi.org/10.1023/A:1007379606734

[8] C. Yu, J. Han, P. Wang, Y. Song, H. Zhang, W. Ng, S. Shi, When hearst is not enough: Improving hypernymy detection from corpus with distributional models, in: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), Association for Computational Linguistics, Online, 2020, pp. 6208–6217. doi:10.18653/v1/2020.emnlp-main.502.
URL https://aclanthology.org/2020.emnlp-main.502

[9] C. Shang, S. Dash, M. F. M. Chowdhury, N. Mihindukulasooriya, A. Gliozzo, Taxonomy construction of unseen domains via graph-based cross-domain knowledge transfer, in: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Online, 2020, pp. 2198–2208. doi:10.18653/v1/2020.acl-main.199.
URL https://aclanthology.org/2020.acl-main.199

[10] C. Zhang, F. Tao, X. Chen, J. Shen, M. Jiang, B. M. Sadler, M. Vanni, J. Han, Taxogen: Unsupervised topic taxonomy construction by adaptive term embedding and clustering, in: Y. Guo, F. Farooq (Eds.), Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018, ACM, 2018, pp. 2701–2709. doi:10.1145/3219819.3220064.
URL https://doi.org/10.1145/3219819.3220064

[11] J. Shang, X. Zhang, L. Liu, S. Li, J. Han, Nettaxo: Automated topic taxonomy construction from text-rich network, in: Y. Huang, I. King, T. Liu, M. van Steen (Eds.), WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020, ACM / IW3C2, 2020, pp. 1908–1919. doi:10.1145/3366423.3380259.
URL https://doi.org/10.1145/3366423.3380259

[12] A. A. Salatino, T. Thanapalasingam, A. Mannocci, F. Osborne, E. Motta, The computer science ontology: A large-scale taxonomy of research areas, in: D. Vrandecic, K. Bontcheva, M. C. Suárez-Figueroa, V. Presutti, I. Celino, M. Sabou, L. Kaffee, E. Simperl (Eds.), The Semantic Web - ISWC 2018 - 17th International Semantic Web Conference, Monterey, CA, USA, October 8-12, 2018, Proceedings, Part II, Vol. 11137 of Lecture Notes in Computer Science, Springer, 2018, pp. 187–205. doi:10.1007/978-3-030-00668-6_12.
URL https://doi.org/10.1007/978-3-030-00668-6_12

[13] C. Wang, X. He, A. Zhou, A short survey on taxonomy learning from text corpora: Issues, resources and recent advances, in: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Copenhagen, Denmark, 2017, pp. 1190–1203. doi:10.18653/v1/D17-1123.
URL https://aclanthology.org/D17-1123

[14] M. Linares-Vásquez, C. Mcmillan, D. Poshyvanyk, M. Grechanik, On using machine learning to automatically classify software applications into domain categories, Empirical Softw. Engg. 19 (3) (2014) 582–618. doi:10.1007/s10664-012-9230-z.

[15] S. Moustafa, M. Y. ElNainay, N. El Makky, M. S. Abougabal, Software bug

prediction using weighted majority voting techniques, Alexandria engineering journal 57 (4) (2018) 2763–2774.

[16] A. Sabetta, M. Bezzi, A practical approach to the automatic classification of security-relevant commits, in: 2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, Madrid, Spain, September 23-29, 2018, IEEE Computer Society, 2018, pp. 579–582. doi:10.1109/ICSME.2018.00058.
URL https://doi.org/10.1109/ICSME.2018.00058

[17] S. Kawaguchi, P. K. Garg, M. Matsushita, K. Inoue, Mudablue: An automatic categorization system for open source repositories, in: 11th Asia-Pacific Software Engineering Conference (APSEC 2004), 30 November - 3 December 2004, Busan, Korea, IEEE Computer Society, 2004, pp. 184–193. doi:10.1109/APSEC.2004.69.
URL https://doi.org/10.1109/APSEC.2004.69

[18] K. Tian, M. Revelle, D. Poshyvanyk, Using latent dirichlet allocation for automatic categorization of software, in: M. W. Godfrey, J. Whitehead (Eds.), Proceedings of the 6th International Working Conference on Mining Software Repositories, MSR 2009 (Co-located with ICSE), Vancouver, BC, Canada, May 16-17, 2009, Proceedings, IEEE Computer Society, 2009, pp. 163–166. doi:10.1109/MSR.2009.5069496.
URL https://doi.org/10.1109/MSR.2009.5069496

[19] S. Ugurel, R. Krovetz, C. L. Giles, What's the code? automatic classification of source code archives, in: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD

'02, Association for Computing Machinery, New York, NY, USA, 2002, p. 632–638. doi:10.1145/775047.775141.

[20] A. LeClair, Z. Eberhart, C. McMillan, Adapting neural text classification for improved software categorization, in: 2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, Madrid, Spain, September 23-29, 2018, IEEE Computer Society, 2018, pp. 461–472. doi:10.1109/ICSME.2018.00056.
URL https://doi.org/10.1109/ICSME.2018.00056

[21] C. Zhou, C. Sun, Z. Liu, F. C. M. Lau, A C-LSTM neural network for text classification, CoRR abs/1511.08630 (2015). arXiv:1511.08630.
URL http://arxiv.org/abs/1511.08630

[22] A. Panichella, B. Dit, R. Oliveto, M. D. Penta, D. Poshyvanyk, A. D. Lucia, How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms, in: D. Notkin, B. H. C. Cheng, K. Pohl (Eds.), 35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013, IEEE Computer Society, 2013, pp. 522–531. doi:10.1109/ICSE.2013.6606598.
URL https://doi.org/10.1109/ICSE.2013.6606598

[23] M. Soll, M. Vosgerau, Classifyhub: An algorithm to classify github repositories, in: G. Kern-Isberner, J. Fürnkranz, M. Thimm (Eds.), KI 2017: Advances in Artificial Intelligence, Springer International Publishing, Cham, 2017, pp. 373–379.

[24] C. McMillan, M. Grechanik, D. Poshyvanyk, Detecting similar software applications, in: Proceedings of the 34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland, ICSE '12, IEEE

Computer Society, 2012, p. 364–374.
URL `https://doi.org/10.1109/ICSE.2012.6227178`

[25] M. L. Vásquez, A. Holtzhauer, D. Poshyvanyk, On automatically detecting similar android apps, in: 24th IEEE International Conference on Program Comprehension, ICPC 2016, Austin, TX, USA, May 16-17, 2016, IEEE Computer Society, 2016, pp. 1–10. `doi:10.1109/ICPC.2016.7503721`.
URL `https://doi.org/10.1109/ICPC.2016.7503721`

[26] D. Altarawy, H. Shahin, A. Mohammed, N. Meng, Lascad : Language-agnostic software categorization and similar application detection, Journal of Systems and Software 142 (2018) 21–34. `doi:https://doi.org/10.1016/j.jss.2018.04.018`.
URL `https://doi.org/10.1016/j.jss.2018.04.018`

[27] J. Di Rocco, D. Di Ruscio, C. Di Sipio, P. Nguyen, R. Rubei, Topfilter: An approach to recommend relevant github topics, in: Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), ESEM '20, Association for Computing Machinery, New York, NY, USA, 2020. `doi:10.1145/3382494.3410690`.
URL `https://doi.org/10.1145/3382494.3410690`

[28] M. Izadi, A. Heydarnoori, G. Gousios, Topic recommendation for software repositories using multi-label classification algorithms, Empirical Software Engineering 26 (5) (2021) 93. `doi:10.1007/s10664-021-09976-2`.
URL `https://doi.org/10.1007/s10664-021-09976-2`

[29] J. Devlin, M. Chang, K. Lee, K. Toutanova, BERT: pre-training of deep bidirectional transformers for language understanding, in: J. Burstein, C. Doran, T. Solorio (Eds.), Proceedings of the 2019 Conference of the North American

Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers), Association for Computational Linguistics, 2019, pp. 4171–4186. `doi:10.18653/v1/n19-1423`.
URL `https://doi.org/10.18653/v1/n19-1423`

[30] C. Chen, K. Lin, D. Klein, Constructing taxonomies from pretrained language models, in: Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Association for Computational Linguistics, Online, 2021, pp. 4687–4700. `doi:10.18653/v1/2021.naacl-main.373`.
URL `https://aclanthology.org/2021.naacl-main.373`

[31] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE Transactions on Neural Networks 20 (1) (2009) 61–80. `doi:10.1109/TNN.2008.2005605`.

[32] J. Huang, Y. Xie, Y. Meng, Y. Zhang, J. Han, Corel: Seed-guided topical taxonomy construction by concept learning and relation transferring, in: R. Gupta, Y. Liu, J. Tang, B. A. Prakash (Eds.), KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020, ACM, 2020, pp. 1928–1936. `doi:10.1145/3394486.3403244`.
URL `https://doi.org/10.1145/3394486.3403244`

[33] F. Osborne, E. Motta, Klink-2: Integrating multiple web sources to generate semantic topic networks, in: M. Arenas, Ó. Corcho, E. Simperl, M. Strohmaier, M. d'Aquin, K. Srinivas, P. Groth, M. Dumontier, J. Heflin, K. Thirunarayan, S. Staab (Eds.), The Semantic Web - ISWC 2015 - 14th In-

ternational Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part I, Vol. 9366 of Lecture Notes in Computer Science, Springer, 2015, pp. 408–424. `doi:10.1007/978-3-319-25007-6\_24`.
URL `https://doi.org/10.1007/978-3-319-25007-6_24`

[34] P. Ye, D. S. Doermann, Active sampling for subjective image quality assessment, in: 2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014, IEEE Computer Society, 2014, pp. 4249–4256. `doi:10.1109/CVPR.2014.541`.
URL `https://doi.org/10.1109/CVPR.2014.541`

[35] A. Mikhailiuk, C. Wilmot, M. Pérez-Ortiz, D. Yue, R. K. Mantiuk, Active sampling for pairwise comparisons via approximate message passing and information gain maximization, in: 25th International Conference on Pattern Recognition, ICPR 2020, Virtual Event / Milan, Italy, January 10-15, 2021, IEEE, 2020, pp. 2559–2566. `doi:10.1109/ICPR48806.2021.9412676`.
URL `https://doi.org/10.1109/ICPR48806.2021.9412676`

[36] R. Herbrich, T. Minka, T. Graepel, Trueskill(tm): A bayesian skill rating system, in: Advances in Neural Information Processing Systems 20, advances in neural information processing systems 20 Edition, MIT Press, 2007, pp. 569–576.
URL `https://www.microsoft.com/en-us/research/publication/trueskilltm-a-bayesian-skill-rating-system/`

[37] M. E. Glickman, Parameter estimation in large dynamic paired comparison experiments, Journal of the Royal Statistical Society: Series C (Applied Statistics) 48 (3) (1999) 377–394.

[38] M. Izadi, A. Heydarnoori, G. Gousios, Topic recommendation for software

repositories using multi-label classification algorithms, Empirical Software Engineering 26 (5) (2021) 93. doi:10.1007/s10664-021-09976-2.
URL https://doi.org/10.1007/s10664-021-09976-2

[39] D. Vrandečić, Wikidata: A new platform for collaborative data collection, in: Proceedings of the 21st International Conference on World Wide Web, WWW '12 Companion, Association for Computing Machinery, New York, NY, USA, 2012, p. 1063–1064. doi:10.1145/2187980.2188242.
URL https://doi-org.proxy-ub.rug.nl/10.1145/2187980.2188242

[40] K. Krippendorff, Estimating the reliability, systematic error and random error of interval data, Educational and Psychological Measurement 30 (1) (1970) 61–70.

[41] K. Krippendorff, Reliability in content analysis: Some common misconceptions and recommendations, Human communication research 30 (3) (2004) 411–433.

[42] J. D. M. Rennie, L. Shih, J. Teevan, D. R. Karger, Tackling the poor assumptions of naive bayes text classifiers, in: T. Fawcett, N. Mishra (Eds.), Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA, AAAI Press, 2003, pp. 616–623.
URL http://www.aaai.org/Library/ICML/2003/icml03-081.php

[43] A. M. Kibriya, E. Frank, B. Pfahringer, G. Holmes, Multinomial naive bayes for text categorization revisited, in: Australasian Joint Conference on Artificial Intelligence, Springer, 2004, pp. 488–499.

[44] R. Rubei, C. D. Sipio, AURYGA: A recommender system for game tagging,

in: V. W. Anelli, T. D. Noia, N. Ferro, F. Narducci (Eds.), Proceedings of the 11th Italian Information Retrieval Workshop 2021, Bari, Italy, September 13-15, 2021, Vol. 2947 of CEUR Workshop Proceedings, CEUR-WS.org, 2021. URL `http://ceur-ws.org/Vol-2947/paper10.pdf`

[45] R. Kohavi, A study of cross-validation and bootstrap for accuracy estimation and model selection, in: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes, Morgan Kaufmann, 1995, pp. 1137–1145. URL `http://ijcai.org/Proceedings/95-2/Papers/016.pdf`

[46] P. Runeson, M. Höst, A. Rainer, B. Regnell, Case Study Research in Software Engineering - Guidelines and Examples, Wiley, 2012. URL `http://eu.wiley.com/WileyCDA/WileyTitle/productCd-1118104358.html`