Building a Large-scale Software Programming Taxonomy from Stackoverflow

Jiangang Zhu
School of Software
Shanghai Jiao Tong University
jszjgtws@sjtu.edu.cn

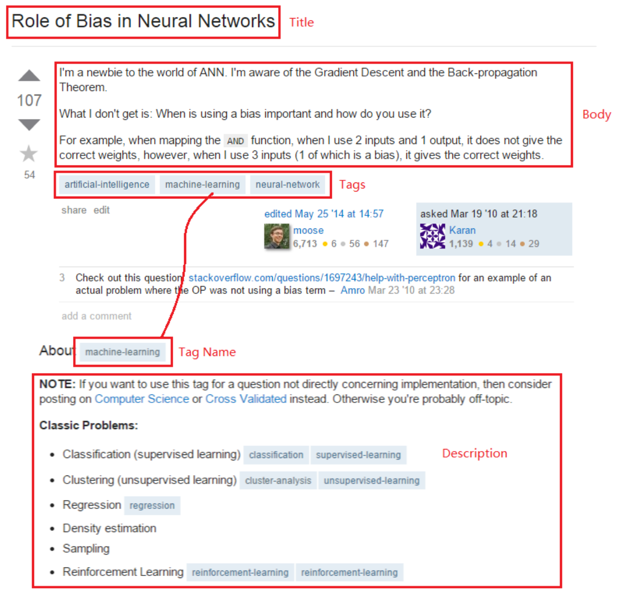
Beijun Shen*
School of Software
Shanghai Jiao Tong University
bjshen@sjtu.edu.cn

Xuyang Cai
School of Software
Shanghai Jiao Tong University
bakercxy@hotmail.com

Haofen Wang*
East China University of
Science & Technology
whfcarter@ecust.edu.cn

***Abstract*—Taxonomy is becoming indispensable to a growing number of applications in software engineering such as *software repository mining* and *defect prediction*. However, the existing related taxonomies are always manually constructed. The sizes of these taxonomies are small and their depths are limited. In order to show the full potential of taxonomies in software engineering applications, in this paper, we present the first large-scale software programming taxonomy which is more comprehensive than any existing ones. It contains 38,205 concepts and 68,098 subsumption relations. Instead of learning from a open domain, we focus on taxonomy construction from Stackoverflow which is one of the largest QA websites about software programming. We propose a machine learning based method with novel features to create a taxonomy that captures the hierarchical semantic structure of tags in Stackoverflow. This method executes iteratively to find as many relations as possible. Experimental results show that our approach achieves much better accuracy than baselines. Compared with taxonomies related to software programming which are extracted from the general-purpose taxonomies such as WikiTaxonomy, Yago Taxonomy and Schema.org, our taxonomy has the widest coverage of concepts, contains the largest number of subsumption relations, and runs up to the deepest semantic hierarchy.**

***Keywords*—Taxonomy Construction, Stackoverflow, Software Engineering**



Role of Bias in Neural Networks

107

I'm a newbie to the world of ANN. I'm aware of the Gradient Descent and the Back-propagation Theorem.

What I don't get is: When is using a bias important and how do you use it?

For example, when mapping the `AND` function, when I use 2 inputs and 1 output, it does not give the correct weights, however, when I use 3 inputs (1 of which is a bias), it gives the correct weights.

54

artificial-intelligence machine-learning neural-network

share edit

edited May 25 '14 at 14:57
moose
6,713 ● 6 ● 56 ● 147

asked Mar 19 '10 at 21:18
Karan
1,139 ● 4 ● 14 ● 29

3 Check out this question: stackoverflow.com/questions/1697243/help-with-perceptron for an example of an actual problem where the OP was not using a bias term – Amro Mar 23 '10 at 23:28

add a comment

About machine-learning

NOTE: If you want to use this tag for a question not directly concerning implementation, then consider posting on Computer Science or Cross Validated instead. Otherwise you're probably off-topic.

Classic Problems:

- Classification (supervised learning) classification supervised-learning
- Clustering (unsupervised learning) cluster-analysis unsupervised-learning
- Regression regression
- Density estimation
- Sampling
- Reinforcement Learning reinforcement-learning reinforcement-learning

Fig. 1: An example from Stackoverflow

I. Introduction

Taxonomy plays an important role in software engineering. For example, in software maintenance such as measuring quality and predicting defects, taxonomies are used to measure the relatedness between documents and create links between bugs and committed changes [1]. In program comprehension, taxonomies provide an effective way to compute the semantic similarities between words from the comments and identifiers in software [2].

However, most existing taxonomies used in these applications are often manually created according to application specific requirements and their sizes are not large enough. A recent literature [3] argued that the quality and the scale of taxonomy would significantly benefit the performance when applied in software engineering. On the other hand, there have been a considerable amount of research works on taxonomy construction [4], [5], [6], [7], [8]. The value of automatic taxonomy construction is two folds. Automatic taxonomy construction can achieve large scale taxonomies while manual construction is a laborious process. Moreover, compared with automatic approaches that are data-driven, taxonomies built manually are often highly subjective. Unfortunately, the resulting taxonomies of these existing automatic approaches would probably lead to poor results when applied in software engineering for several reasons. First is *timeliness*. The techniques in software engineering are fast changing, while the general web pages and encyclopedic sites are insensitive to this change and always fail to update in time. So it is not suitable to select text corpora such as general web pages or Wikipedia as its input. Second is *granularity*. Since the input of traditional taxonomy construction approaches is from a open domain, some fine-grained terms about software programming cannot be found in these taxonomies. For example, "hashmap" about a well-known data structure is not included in either Yago Taxonomy [9] or WikiTaxonomy [4] which are the largest existing public available taxonomies.

Recently, Stackoverflow has becoming one of the largest QA websites about software programming. Specifically, *questions* are the key elements in Stackoverflow. Besides the question description, as shown in Fig. 1, a question is also associated with tags and authors. Formally, a question q is a triple in form of (t_q, b_q, TS_q), where t_q is the title of the question, b_q is the body while TS_q is the tag set which annotate the question. A *tag* a in Stackoverflow can be represented as a

*Corresponding author

DOI reference number: 10.18293/SEKE2015-135

four-tuple (t_a, d_a, c_a, Q_a), where t_a is the name of the tag, d_a is the description of a, c_a is the number of questions that it has annotated, Q_a is the set of questions annotated by a. These tags represent vocabularies about software programming. They can also reflect the fast changing nature of technique terms because they are created on the fly by Web users. So the large amount of tags provide a promising way to build the taxonomy. Therefore, in this paper, we try to construct a software programming taxonomy from tags in Stackoverflow.

The problem is non-trivial and poses unique technical challenges. First, tags in Stackoverflow are always composed of domain-specific terms. While natural language processing techniques like segmentation or pos tagging are basis of some Web-based approaches for taxonomy construction, directly applying these approaches to our scenario will lead to poor results. Second, most tag-based taxonomy construction approaches only use tag co-occurrences and annotated documents to help the subsumption detection between tags [8], [7]. However, Stackoverflow contains unique information such as wiki descriptions of tags. We argue that these information will significantly increase the performance of taxonomy construction. How to design an algorithm to incorporate these information when detecting subsumptions between tags has not been studied yet. Moreover, there are tens of thousands of tags in Stackoverflow, it is time-consuming and sometimes impractical to enumerate all tag pairs for subsumption relation detection. How to perform subsumption relation detection in a large scale scenario is also a challenge problem.

In order to solve the above challenges, we propose a machine learning based approach. Specifically, our approach leverages several features from different aspects to measure the semantic relatedness between tags. Then a semi-supervised learning method is used to predict subsumption relations. To the best of our knowledge, we are the first to focus on building a software programming taxonomy from Stackoverflow. Our contributions mainly include:

- To overcome the informality of tags, we leverage different information from Stackoverflow to represent these tags. Specifically, we design the co-occurrence-based features to measure the semantic relatedness. Moreover, we also use the wiki description and annotated questions of each tag to learn a topic representation by applying Latent Dirichlet Allocation (LDA) [10]. Together with a lexical feature, our proposed approach can combine evidences from the co-occurrence relevance, the implicit topic relevance and the lexical relevance.
- We propose a unified model that incorporates these features to automatically construct a software programming taxonomy. Specifically, we design a blocking mechanism to reduce the number of tag pairs to be calculated which ensure our approach can be applied to a large scale scenario. Then, we treat subsumption relation detection as a binary-class classification problem to solve. A semi-supervised learner is applied which executes iteratively to find as many relations as possible. Note that the blocking mechanism and semi-supervised learning make our approach quite general and can be applied to other domains.
- We present a taxonomy about software programming. Experimental results show that our taxonomy not only contains a large number of concepts and subsumption relations, but also have a deep semantic hierarchy. That is to say, concepts and subsumption relations in our taxonomy are more fine-grained and can be applied in many software engineering applications.

II. Related Work

There have been a considerable amount of research works on taxonomy construction. Approaches for automatic taxonomy construction can be encyclopedic-based or Web-based.. For the encyclopedic-based approaches, they mainly focus on extracting concept hierarchies from Wikipedia. WikiTaxonomy [4] builds a taxonomy from the Wikipedia category system. It contains 105,000 subsumption relations with the accuracy of 88%. Kylin Ontology Generator (KOG) [5] uses Markov Logic Network (MLN) to predict subsumption relations between Wikipedia infobox classes. Yago [9] interlinks Wikipedia categories to WordNet synsets. There are over 200,000 classes and 400,000 subsumption relations in Yago and the accuracy is estimated to be 96%. Our research is quite different from the encyclopedic-based approaches because there has been no structure information between tags in Stackoverflow.

Regarding Web-based approaches, it can be free text based or social tag based. For the free text based approaches, Hearst patterns [11] are widely used. The most recent effort is Probase [12]. It builds the largest taxonomy which contains over 2.7 million classes from 1.7 billion web pages. For the social tag-based approaches, Mianwei Zhou et al. [6] introduced an unsupervised model to automatically derive hierarchical semantics from social annotations. Jie Tang et al. [7] proposed a learning approach to capture the hierarchical semantic structure of tags. Xiance Si et al. [8] proposed three methods to estimate the conditional probability between tags and used a greedy algorithm to eliminate the redundant relations. Huairen Lin et al. [13] described an integrated method for extracting ontological structure from tags that exploits the power of low support association rule mining supplemented by an upper ontology such as WordNet. Zhishi.schema [14] is the first effort to publish a general taxonomy from tags and categories in popular Chinese Web sites. These traditional tag-based approaches for a general domain only use the annotated documents to help the subsumption detection. However, Stackoverflow is more domain specific which contains other information such as wiki descriptions. So traditional tag-based approaches may not be the best because the additional information in Stackoverflow will probably increase the performance of taxonomy construction.

Regarding taxonomy construction in software programming, the most recent research is *Lexical Views* [3]. It applied some natural language processing techniques to automatically extract and organize concepts from software identifiers in a WordNet-like structure. But more software programming terms would not be included in this taxonomy since *Lexical Views* only use the software identifiers as its input. In our research, we first focus on automatically constructing a software programming taxonomy from Stackoverflow. Specifically, we design several novel features which can capture similarities between tags from several aspects. Also, we use a more sophisticated semi-supervised learning approach by generating labeled examples semi-automatically.

III. Approach

In this section, we start with a brief introduction of our proposed approach, and then describe it in details.

A. Approach Overview

We now provide a workflow to explain the whole process and how different components interact with each other. As shown in Fig. 2, we have four main components, namely *Candidate Selection*, *Labeled Data Generation*, *Feature Engineering*, and *Semi-supervised Learning*. The input of *Candidate Selection* is tags collected from Stackoverflow. *Candidate Selection* tries to divide all tags into blocks. Each block includes similar tags which can form candidate tag pairs for further processing. *Candidate Selection* leads to a significant reduction of the number of tag pairs to be further processed for subsumption relation detection, which guarantees the scalability and efficiency of our approach. We generated labeled data (both positive and negative) semi-automatically using a rule-based method. All the tag pairs are fed to *Feature Engineering* to extract features like co-occurrence-based features and the topic-based features. A semi-supervised learning algorithm is adapted to discover hypernym-hyponym relations. The learned classifier can be updated iteratively by adding new labeled data of high confidence. Finally we build a software programming taxonomy composing subsumption relations between tags.

B. Candidate Selection

Since Stackoverflow contains tens of thousands of tags and the number is still increasing, it is time-consuming and sometimes impractical to enumerate all tag pairs as candidates for subsumption relation detection. To avoid brute-force comparison, we leverage the co-occurrence information to limit the number of candidates. Previous research [8], [15] shown the effectiveness of the co-occurrence information in subsumption relation detection. So, only if two tags have once co-occurred, they can be divided into the same block and can be selected as candidate pairs. Note that given a candidate tag pair (a, b), both a subsumes b and b subsumes a will be checked in the next learning process. Given we collected 38,205 tags from Stackoverflow, there would be over one billion pairs without blocking. Only less than 3 million candidates will be retained after using the above candidate selection mechanism. It is obvious that the blocking mechanism reduces the number of candidate pairs significantly.

C. Feature Engineering

The purpose of feature engineering is to quantitatively characterize the similarities or relatedness between tags. We define six features to characterize the tag relations. The details of these features are as follows:

Lexical Feature:

1) Lexical Feature: Given a tag pair "asp.net" and "asp.net mvc", it is intuitive that they hold the subsumption relation in term of the lexical pattern. So we define a Token-based Longest Common Sub-string Asymmetric Similarity as lexical feature by considering the length information. Then the lexical similarity between two tags a and b is computed by

$$s(a,b) = \frac{|LCS(seq(t_a), seq(t_b))|}{|seq(t_a)|} \quad (1)$$

Where $seq(t_a)$ is the word sequence of the name of the tag a, $|.|$ returns the length of a word sequence, and LCS is a function to calculate the longest common sub-string sequence between two tag labels. This similarity measure captures the lexical similarity between two tags. Since we treat version number as a seperated token, this metric can also capture subsumption relations between those tags and their instance versions. For example, "c++" and "c++11".

Co-occurrence-based Features: Although the lexical feature works well, its limitation is obvious. Many tag pairs which actually hold the subsumption relations have very low lexical similarities. Thus, we also leverage the co-occurrence information to measure the semantic relatedness between tags. For example, "word2vec" and "deep-learning" do not share any lexical tokens. However, by considering the co-occurrences of them in questions, we can find that they always co-occur and may be semantically closed.

2) Question Divergence Feature: We define this feature to measure the co-occurrence of tags in questions based on the Normalized Google Distance [16].

$$d(Q_a, Q_b, Q) = \frac{log(max(|Q_a|, |Q_b|)) - log(|Q_a \cap Q_b|)}{log(|Q|) - log(min(|Q_a|, |Q_b|))} \quad (2)$$

where Q_a and Q_b are the sets of questions annotated with a and b, respectively; and Q is the set of all questions in Stackoverflow.

3) Sentence Divergence Feature: Stackoverflow additionally provides wiki description for each tag. The wiki description provides a more precise explanation for each tag. If two tags have co-occurred in the same sentence such as "java" and "programming language", they may be semantically closed even if the question divergence feature of them is rather low. Inspired by this idea, we define the sentence divergence feature which aims to measure the co-occurrence of tags in sentences. These sentences are extracted from wiki descriptions of all tags. The computation of this feature $d(S_a, S_b, S)$ is similar as Equation 2, where S_a and S_b are the sets of sentences which contain a and b, respectively; and S is the set of all sentences in all wiki descriptions from Stackoverflow.

4) Tag Divergence Feature: Given two tags a and b, if both of them have co-occurred with tag c, they tend to hold a semantic relation. Inspired by this basic idea, we also design a tag divergence feature to measure the relatedness between tags. We compute this feature $d(T_a, T_b, T)$ as Equation 2, where T_a and T_b are the sets of tags of Q_a and Q_b, Q_a and Q_b are the sets of questions annotated with a and b, respectively; and T is the set of all tags in Stackoverflow.

Topic-based Features: The lexical feature and the co-occurrence features only capture the semantic relation in an explicit way which can not detect the implicit relations between tags. For example, most people tend not to annotate "machine learning" together with "artificial intelligence" because "artificial intelligence" is too high-level. But it is obvious that "artificial intelligence" is semantically closed to "machine

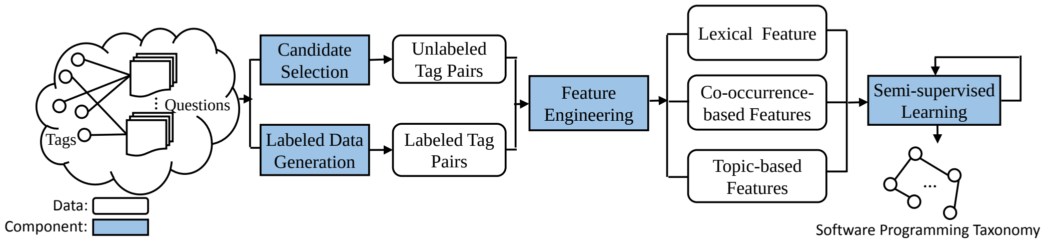


Fig. 2: The workflow to generate taxonomy for Stackoverflow

learning". So we also try to represent each tag in a sematic level and design some topic-based features. We leverage a topic modeling method named LDA [10] to generate a topic-based representation for each tag. In our proposed approach, we model a tag using both of its wiki description and the bodies of its annotated questions, and learn its topic representation by LDA. In our experiments, the number of topics was empirically set as 150.

5) Wiki Topic Divergence Feature: This feature is the measure of the difference or dissimilarity between two topic distributions of wiki descriptions of tag a and tag b. We define this feature based on KL-divergence [17], a standard measure of the difference between two probability distributions. Note that it is an asymmetric metric, i.e. $d_{KL}(p_{wa}, p_{wb}) \neq d_{KL}(p_{wb}, p_{wa})$.

$$d_{KL}(p_{wa}||p_{wb}) = \sum_{i=1}^{K} p_{wa}(i) log \frac{p_{wa}(i)}{p_{wb}(i)} \tag{3}$$

where $p_{wa}(i)$ and $p_{wb}(i)$ denote the probability of i-th topic in the topic distribution of a and b respectively, using wiki descriptions as document.

6) Question Topic Divergence Feature: In Stackoverflow, some tags only contain few contents in their wiki descriptions, especially when the tags are newly created or have fewer editors. In this circumstance, wiki topic divergence cannot accurately assess the dissimilarity between tags. However, the topic of questions can better represent their tags, since the same tag in different questions is annotated by different editors, which can avoid the subjectivity. Therefore, for a given tag, we also use questions annotated by it to represent the tag. Similarly, we compute this feature as Equation 3.

D. Labeled Data Generation

We treat subsumption relation detection as a binary-class classification problem to solve. Classification (binary or multi-class) is supervised learning, which requires labeled data for training. The classification performance depends on whether the labeled data is adequate and whether training data and test data have the similar distributions. In order to ease the burden of manual labeling and to avoid distribution bias, we propose an effective rule-based method to create labeled data. Both positive and negative examples are checked manually. These examples are not only used to boost the learning process but also treated as ground truth to evaluate our approach in Section IV.

For positive examples, we apply some lexical-syntactic patterns on descriptions of tags. These patterns are extended from the Hearst patterns [11] (e.g. NP1 is a/an NP2). Finally, we have 12,608 hypernym-hyponym candidate relations between tags and 2,870 of them are manually checked as positive examples.

For negative examples, we define a conditional probability metric to measure the probability of a as the hypernym given b. This metric relies on an implication, that if a user has annotated a document d with b, he also tend to annotate tags that subsumes b. Specifically, we use the following formula:

$$p(a|b) = \frac{N_d(a, b)}{N_d(b)} \tag{4}$$

where $N_d(a, b)$ is the number of documents that are annotated by both a and b, and $N_d(b)$ is the number of documents that are annotated by b. Given a tag b, we select the tag a as its hypernym with its probability less than 0.01 and treat the pair $(a,\ b)$ as negative examples. Besides, we also manually select those relations, which do not hold subsumption relations but their probabilities $p(a|b)$ are more than 0.5 to enrich the set of negative examples. Finally, among these enriched sets, 3,000 negative examples are manually checked and selected.

E. Semi-supervised Learning

While we generate labeled data by applying some lexical-syntactic patterns and heuristic rules semi-automatically, the number of positive and negative examples is very small compared with that of the candidate tag pairs. So a natural idea is to use some kind of semi-supervised learning algorithm to predict new semantic relations between these candidate pairs.

We select the simplest and the most efficient one - self-training. In each iteration, self-training accepts the labeled data as training data and learns a classifier. Then the classifier is applied to the unlabeled data and adds tag pairs of high confidence to the labeled data to train a new classifier for the next iteration. The whole process will terminate if the difference between the predicted labels of these candidates (whether they satisfy subsumption relations or not) given by classifiers in the two consecutive iterations is smaller than a threshold or we have achieved the maximal number of iterations. Note that we use the Support Vector Machine (SVM) algorithm to train the binary classifier, which is known as one of the best single classifiers [18].

IV. Experiments

A. Experiment Setup

1) Data Statistics: In order to evaluate our approach, we use the Stackexchange dump from https://archive.org/details/stackexchange. Note that before further process, we first singularize the names of all tags and then replace underscores and hyphens by spaces as preprocess. In total, there are 38,205 tags and 7,990,787 questions. Among these tags, 25,798 tags have descriptions. The number of questions annotated by the tag ranges from 1 to 708,533. On the average, each tag annotated 617 questions.

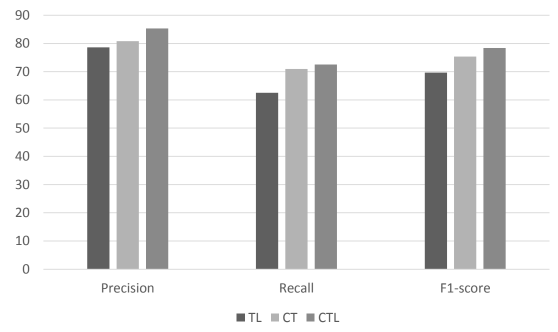


Fig. 3: Three models with different feature sets

2) Comparison Methods: We select several state of the art methods as comparison methods, namely Tag-Tag Co-occurrences (TTC) method and Tag-Word Co-occurrences (TWC) method.

- **Tag-Tag Co-occurrences (TTC).** The TTC method [15] uses Equation 4 to estimate $p(a|b)$. One of its benefits is that it does not rely on the content of the annotated document, so it can be applied to tags for non-text objects.
- **Tag-Word Co-occurrences (TWC).** This method [8] uses the content of the annotated document to estimate $p(a|b)$. We use the following formula to estimate $p(a|b)$ by tag-word co-occurrences:

$$\begin{aligned} p(a|b) &= \sum_{w \in W} p(a|w)p(w|b) \\ &= \sum_{w \in W} \frac{N_d(a,w)}{N_d(w)} \frac{N_d(b,w)}{N_d(b)} \end{aligned} \tag{5}$$

  where $N_d(a,w)$ is the number of documents that contains both tag a and word w, and $N_d(w)$ is the number of documents that contains the word w. Instead of computing tag-tag co-occurrences directly, TWC uses words in the document as a bridge to estimate $p(a|b)$.

For these two comparison methods, we first sort the discovered relations by their probabilities in descending order. Then, we take the top-n relations, discarding the others. Here we evaluate these methods with $n = 1$ and $n = 5$.

B. Result Analysis

1) Feature Contribution Analysis: We discuss the effect of different features for predicting subsumption relations. We use the labeled data generated in Section III-D as the ground truth. Then we train three SVM classifiers based on different combinations of features. The first classifier (denoted as *TL*) only uses topic-based features and the lexical feature. The second classifier (*CT*) combines the co-occurrence-based features and the topic-based features. The third one (*CTL*) includes all features. We apply 5-fold cross validation to train the three classifiers. Precision, recall, and F-measure are used for effectiveness study. As shown in Fig. 3, the classifier with all features performs best. That is to say, all these features are useful in predicting new subsumption relations. We can also find all classifiers use topic-based features, this is mainly because the subsumption relation is asymmetric, so only asymmetric metrics can capture this relation semantically. In our feature set, only topic-based features can measure the subsumption relations while the co-occurrence-based features are symmetry and the lexical feature can only capture the surface patterns.

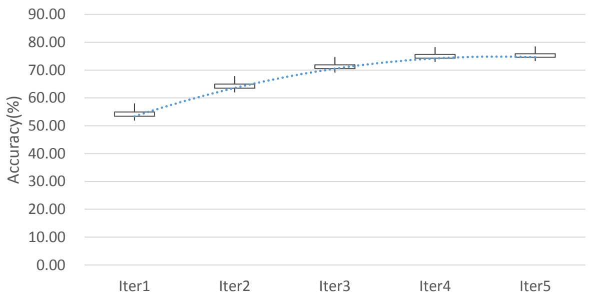


Fig. 4: Accuracy in each iteration

2) Accuracy Evaluation of Iterations: We applied the iterative semi-supervised learning approach with all features to build a taxonomy containing 68,098 subsumption relations. Since there are no ground truths available for the whole taxonomy, the qualities of these relations have to be verified manually. Due to the large number of relations, it is impossible to evaluate all of them by hand. Therefore, we design an evaluation theme including a *sampling* strategy and a *labeling* process. *Sampling* aims to extract a subset of relations (called samples) which can represent the distribution of the whole result set. Then we can perform manual labeling to evaluate the correctness of samples. The accuracy assessment on the subset can further be used to approximate the correctness of the resulting taxonomy.

We first evaluate the performance improvements in each iteration compared with the estimated accuracy produced by the previous iteration if existed. We randomly select 1,000 subsumption relations from the resulting taxonomy. We record their predicted labels after each iteration. Four students from our laboratory are invited to participant in the labeling process. We provide them three choices namely *agree*, *disagree* and *unknown* to label each sample. Then we can compute the average accuracy. Finally, the *Wilson interval* [19] at α = 5% is used to generalize our findings on the samples to the whole taxonomy. Fig. 4 shows the accuracy of each iteration. According to the results, the accuracy increases consistently when we perform more iterations. In particular, after the fifth iteration, our approach achieves the best accuracy of $75.90\% \pm 2.64\%$.

C. Performance Comparision

We further compare the two baseline approaches with ours. Accuracy and scale are used as evaluation metrics. According to the result, the TTC and TWC method get a similar scale of 184,818 and 184,816 subsumption relations respectively with $n = 5$. It is obvious because they all apply a ranking mechanism by probability. For accuracy, the TTC method achieves accuracy of $53.29\% \pm 3.09\%$, while the TWC method only achieves accuracy of $37.05\% \pm 3.0\%$. Even the threshold n is set to 1, the accuracy of TTC and TWC method is only $70.42\% \pm 2.82\%$ and $43.03\% \pm 3.06\%$ respectively with the trade-off of scale which is only 38,166. Compared with these two baseline methods, our method can find 68,098 subsumption relations and the accuracy of resulting taxonomy using our proposed method is $75.90\% \pm 2.64\%$. Therefore, our approach can not

TABLE I: Comparison with other datasets

	Ours	**Yago**	**WikiTaxonomy**	**Schema.org**
Concept Number	**38,205**	898	711	10
Concept Overlap	/	29	27	2
Subsumption Number	**68,098**	870	630	0
Subsumption Overlap	/	0	1	0
Maximum Depth	**28**	3	6	1
Minimum Depth	1	2	1	1
Average Depth	**6.99**	2.24	1.39	1.00

only discover more subsumption relations but also achieve a better accuracy.

D. Comparison with Other Datasets

Since there is no public software programming taxonomy, we only compare our taxonomy with the subsets about software programming extracted from other well-known general-purpose datasets namely Yago Taxonomy, WikiTaxonomy and Schema.org[1] in terms of concepts and subsumption relations. Table I not only shows the concept and subsumption information of each dataset, but also lists the concept overlaps and subsumption overlaps between our taxonomy and the other datasets. Moreover, we present the maximum, minimum and average depth of each dataset to illustrate the granularity and richness. As for the concept and subsumption number, our taxonomy is much larger than any other datasets. The overlaps of both concept and subsumption with these datasets are not so high. The reason mainly comes from two aspects. First, concepts in our taxonomy are fine-grained while those in Yago Taxonomy, WikiTaxonomy and Schema.org are more high-level. Second, the overlaps between our taxonomy and any of the other three (i.e., Yago Taxonomy, WikiTaxonomy, and Schema.org) are the lower bounds and can actually be larger due to the fact that we compute the overlaps using the exact string matching and tag from Stackoverflow are not as formal as those in the three compared datasets.

Regarding to the granularity and richness of concepts, our taxonomy has the largest average depth and maximum depth. That is to say, our taxonomy has a more fine-grained concept hierarchy compared with other existing datasets. As a representative example, our taxonomy contains a hypernymy path like "machine learning"$\rightarrow$"bayesian"$\rightarrow$"bayesian network"$\rightarrow$"belief propagation". Moreover, some newly terms can also be found in our taxonomy such as "neural network"$\rightarrow$"deep learning"$\rightarrow$"word2vec". So our taxonomy contains not only many newly-added and fine-grained concepts, but also a richer semantic hierarchy.

V. Conclusion And Future Work

In this work, we proposed a machine learning based approach with some novel features to automatically create hypernym-hyponym relations between tags in Stackoverflow, which results in a taxonomy about software programming containing 38,205 concepts and 68,098 relations. The experiments show the high-quality of this taxonomy.

As for future work, we will try to extract more concepts about computer programming from Wikipedia, Github and other Web sites to enrich our taxonomy. Moreover, it would be interesting to explore some more potential applications based on this taxonomy such as linked data based recommendation, semantic relatedness measuring between terms about software programming and so on.

[1]https://schema.org/

VI. Acknowledgment

This research is supported by 973 Program in China (Grant No. 2015CB352203) and National Natural Science Foundation of China (Grant No. 61472242).

References

[1] Rongxin Wu, Hongyu Zhang, Sunghun Kim, and Shing-Chi Cheung. Relink: recovering links between bugs and changes. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 15–25. ACM, 2011.

[2] Giriprasad Sridhara, Emily Hill, Lori Pollock, and K Vijay-Shanker. Identifying word relations in software: A comparative study of semantic similarity tools. In *ICPC 2008*, pages 123–132. IEEE, 2008.

[3] J-R Falleri, Marianne Huchard, Mathieu Lafourcade, Clementine Nebut, Violaine Prince, and Michel Dao. Automatic extraction of a wordnet-like identifier network from software. In *Program Comprehension (ICPC), 2010 IEEE 18th International Conference on*, pages 4–13. IEEE, 2010.

[4] Simone Paolo Ponzetto and Michael Strube. Wikitaxonomy: A large scale knowledge resource. In *ECAI*, volume 178, pages 751–752, 2008.

[5] Fei Wu and Daniel S Weld. Automatically refining the wikipedia infobox ontology. In *WWW*, pages 635–644. ACM, 2008.

[6] Mianwei Zhou, Shenghua Bao, Xian Wu, and Yong Yu. *An unsupervised model for exploring hierarchical semantics from social annotations*. Springer, 2007.

[7] Jie Tang, Ho-fung Leung, Qiong Luo, Dewei Chen, and Jibin Gong. Towards ontology learning from folksonomies. In *IJCAI*, volume 9, pages 2089–2094, 2009.

[8] Xiance Si, Zhiyuan Liu, and Maosong Sun. Explore the structure of social tags by subsumption relations. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 1011–1019, 2010.

[9] Johannes Hoffart, Fabian M Suchanek, Klaus Berberich, and Gerhard Weikum. Yago2: a spatially and temporally enhanced knowledge base from wikipedia. *Artificial Intelligence*, 194:28–61, 2013.

[10] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[11] Marti A Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics-Volume 2*, pages 539–545. Association for Computational Linguistics, 1992.

[12] Wentao Wu, Hongsong Li, Haixun Wang, and Kenny Q Zhu. Probase: A probabilistic taxonomy for text understanding. In *SIGMOD 2012*, pages 481–492. ACM, 2012.

[13] Huairen Lin, Joseph Davis, and Ying Zhou. An integrated approach to extracting ontological structures from folksonomies. In *The semantic web: research and applications*, pages 654–668. Springer, 2009.

[14] Haofen Wang, Tianxing Wu, Guilin Qi, and Tong Ruan. On publishing chinese linked open schema. In *ISWC 2014*, pages 293–308. Springer, 2014.

[15] Patrick Schmitz. Inducing ontology from flickr tags. In *Collaborative Web Tagging Workshop at WWW2006*, volume 50, 2006.

[16] Rudi L Cilibrasi and Paul MB Vitanyi. The google similarity distance. *IEEE Transactions on Knowledge and Data Engineering*, 19(3):370–383, 2007.

[17] Solomon Kullback. *Information theory and statistics*. Courier Corporation, 1997.

[18] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15:3133–3181, 2014.

[19] Lawrence D Brown, T Tony Cai, and Anirban DasGupta. Interval estimation for a binomial proportion. *Statistical Science*, pages 101–117, 2001.