Stack-Exchange Tag Recommendation

Bhavya Goyal
cs1120222@iitd.ac.in
Indian Institute of Technology, Delhi

Abstract

Most of the Q&A websites have an option to assign appropriate labels/tags to each question. The task of automatically predicting such tags or labels for each question is of great importance. It helps such websites in categorizing all the questions to make it more user-friendly and at the same time helps the experts, who answer the questions, to quickly find the questions related to their area of experitize. Many times a new user doesn't know how to tag a question appropriately or even a past user may find it difficult to assign an appropriate tag so that right audience can find that question. In such cases, automatic prediction of tags will ensure that the question is answered quickly as right experts are more likely to find it. In this paper, we use the data of Stack-Exchange websites like Stack Overflow, Math Overflow etc. to create an automatic tag recommendation system. Our Experimenation on the dataset provided by the Kaggle competition shows that our system has an average accuracy of 75% for most frequent tags which is an improvement from 68% as proposed by Avigit K. Saha et. al. ([2]). Our system shows an improvement of recall@5 from 52.4% of Avigit et. al. to 56.1%
Index Terms automated tagging, tag prediction, Stack Exchange Q&A

Introduction

Tagging is a very popular and efficient way to categorize the data and search related information on software information sites. Many newpapers, journals, blogs etc. use tags to categorize multiple articles, news, documents etc. Q&A websites like StackOverflow, MathOverflow etc. also uses tags for each question for similar purpose. Tags on StackOverflow are used to describe the important features of the posted question. While adding a new question to the website, a user has the option to add upto 5 tags for a question in order to better categorize it. Relevant tagging of a question is important as better tags increase the chances of question being solved by others. This is largely because the experts who usually write answers to such questions mostly browse the questions of his/her own area of expertize.

Related Work

Text categorization and tagging has been the focus of many researchers in natural language processing (NLP) community. Recent work on automatic tag recommendation in this specific domain includes the work by Avigit et. al.[2] published in 2013. They provide a discriminative model approach for suggesting tags automatically for stack overflow Q&A which has an average accuracy of 68% for each tag. Work by Clayton Stanley et. al. [5] provides an ACT-R inspired Bayesian probabilistic model which is 65% accurate but only when it predicts one tag per question. This is usually not the case as user needs to be prompted with atleast 4-5 tags for each question. Work by Xin Xia et. al.[3] proposes a tag recommendation method, named TagCombine, which has three main components, multi-label ranking, similarity-based ranking and tag-term based ranking. All of these are combined to get a recall@5 of 0.596 on stackoverflow data.

Dataset

Dataset used for our experimentation was from the Kaggle Competition [1]. It was a huge dataset of questions posted on various Q&A websites like Stack Overflow, Math Overflow, Ask Ubuntu etc. Most of the questions come from the domain of programming and mathematics and large number of such questions are from Stack Overflow. The dataset consisted of 6,034,196 (6 million) questions with size being around 8GB. Each row in the dataset is a question which has four fields, namely, ID, title, text and tags.
On the primary analysis of the dataset, we noticed that there are around 40,000 unique tags in the dataset. The most common tags among all were C#, Java, php, javascript, android, jquery, C++, python, iPhone, asp.net, mySQL, html, .net, ios, Objective-C. Thus, the most common tags were of programming languages only. But the usage of these tags is very skewed, i.e. a small set of these frequest tags are used most of the times and most of the other tags are used very few times. So, for our training dataset, we picked up only the questions with most frequent tags (800 tags) and only those questions which had atleast 500 characters present in it.

For testing, we separated out 10,000 questions from the dataset and made sure that these questions were not part of the training set. Note that these questions may have tags

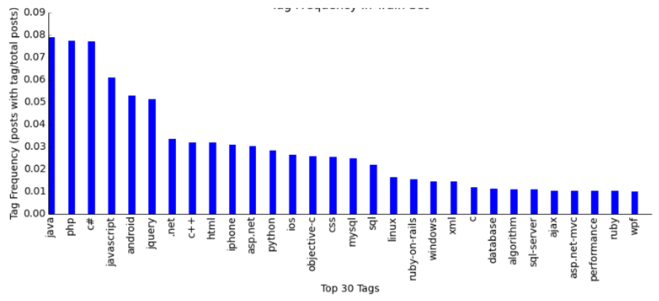


Figure 1: Distribution of most frequent tags
Source: Project report by Rajiv Tarigopula [6]

other than the frequent ones we selected for the training set.

We also observed that the number of tags per question is not close to 5 (the upper bound). Most of the question have been given only two tags. This shows that if more tags are predicted for such questions, it is more likely that they will be answered.

Technical Challenges

The text of the question body is not just plain text. More than 73% of the questions contain code snippets which contain very valuable information. This is because many tags can be decided from the code itself like programming language etc.
As explained in the previous section, the number of tags is also not constant which makes it difficult to decide how many tags to predict for each question. This means that if a question has low number of original tags and we suggest 5 (or so) tags, our accuracy will go down even if the tags were good. Because of this we used recall@5 (Recall when 5 tags are predicted) as our evaluation metric rather than average accuracy.
Another major challenge arises because of presence of tag synonyms - tags that are syntactically different, but semantically the same. For instance, in the case of StackOverflow, we can see tags such as 'zombie' and 'zombies' being used which both describe the same concept of zombie process in Unix. Similar example of tags like 'xmlparser', 'xml-parser', and 'xmlparsing' being used but they all refer to the same concept. This poses a problem as similar questions gets distributed into multiple tags which confuses the user and recommendation system as well.

Experimental Evaluation

Baseline and scope for improvement

We used the method proposed by Avigit et. al.[2] as our baseline. It trains an svm classifier for each tag by collecting enough positive and negative examples corresponding to that tag. Then for every new question, it computes the likelihood probabilities of each tag and then selects the top k tags with highest likelihood. We analysed the method proposed and identified few areas which had scope for improvement. For example, the method does not account for synonymous tags. Other improvements like meta-features, term affinity, use of word embeddings etc. are described in the next sections.

Meta-features, classifiers, weight of title

Many meta features were added to the feature matrix such as number of code segments in the question text, punctuations used in the code segement (as these are important features for identifying the programming language which is usually a tag), number of 'href' tags or links etc. Further, we experimented on different types for classifiers for the problem like LinearSVC, logistic regression etc. Logistic regression gave us the best results among all.
We also realized that many tags come from the title itself, which is because of the fact that users are careful in defining the title of the question at the time of posting it. So, we increased the weight of title by repeating it multiple times. All these techniques combined gave us a recall@5 of 54% which is an increase of 1.6%

Doc2Vec

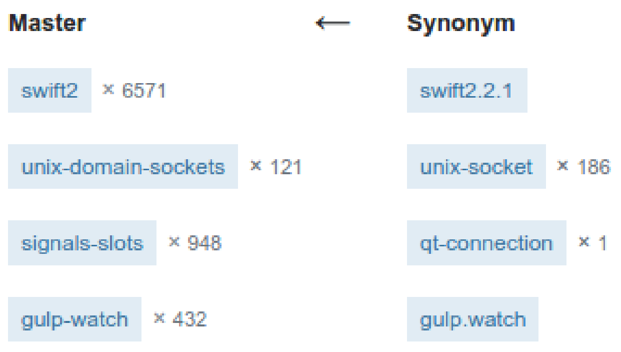
Doc2Vec modifies the word2vec algorithm to learn continuous vector space embeddings for large pieces of text, like documents, sentences and paragraphs. We tried to use doc2vec to learn and predict vector space embeddings for the given questions. To train the model, we used around 50,000 questions since it was taking a long time to train the model on large corpus. The title and the body of the questions were combined and the model trained 10 times with random shuffling of data-points and with a vector size of 100, window size of 10 and mincount of 1. The representation obtained from doc2vec model was used as feature for the main classification algorithm. As can be seen in the summary table, our recall dropped by 1%. A probable reason could be that the text of question was not simply plain text in english but it consisted of many code snippets which is not handled well by doc2vec. Another reason could be that the feature added by the doc2vec were not that important and were just overfitting as they might be highly correlated with previous features. Yet another reason could be that our classifier was linear (logistic regression) and hence could not make proper use of the representations learned by doc2Vec.

KMeans with Word2Vec

We used Google's pre-trained Word2Vec model for this purpose. First all the words that were of length atleast 3, occuring atleast 4 times in the corpus and also present in the Google's Word2Vec model were collected to make a vocabulary. Then the representations of these words were used to train a mini-batch kMeans model. With this kMeans model, for a given question title and body, the cluster-id of the words contained in it was added to the text. The classification algorithm was then run on this modified text data. We observed only a marginal improvement (around 0.5%) in the recall using this.

Tag synonymy

As already explained in the previous sections, tag-synonyms present a technical challenge for our problem. We observed that Stack-Overflow has a page where users can add requests for synonym tags and expert users can then approve it. We decided to scrape this information from the Stack-Overflow website and use it in our system. The figure 2 shows an exmaple of such a web-page. In total, after scrapping through all the pages available, we had 3365 pairs of synonym tags. We replaced all occurences of synonyms with its master synonym tag and trained our system on this corpus. Intuitively, since the system now does not confuse between tags like Java5.0, Java-5.0 and Java5, it performs better than before. With this technique, we were able to raise our recall to around 55.5%. Looking carefully, we did not actually learn the tag-synonym pairs, but only used the available data from online resources. This presents a future scope for improvement, by learning a model that can automatically recognize new synonym tag pairs.



Master		←	Synonym	
swift2	× 6571		swift2.2.1	
unix-domain-sockets	× 121		unix-socket	× 186
signals-slots	× 948		qt-connection	× 1
gulp-watch	× 432		gulp.watch	

Figure 2: Example of StackOverflow synonym-tag pairs

Term Affinity

Many multi-label classification problems use this technique for their problem. It is a measure of the co-occurence of the tags and the term. Counting number of such co-occurences accounts for the importance of the term with respect to the tag. We define affinity as

$$Aff(tag, term) = n_{term,tag}/n_{tag}$$

where $n_{term,tag}$ denotes the number of questions where both $term$ and tag both appears together and n_{tag} is the number of questions which has tag. For every tag, we will add a new feature to the feature matrix which will be a score defined as:

$$TagTermScore_q(tag) = 1 - \prod_{term \in q} (1 - Aff(tag, term))$$

For every question, we will iterate over all the terms to compute the TagTermScore for that question and tag. As we can see, higher the TagTermScore, higher are the chances of that tag. We saw an improvement of around 0.5% in recall@5 with the Term Affinity.

Results

The following table provides the summary of the results obtained on experimentation on kaggle dataset.

Technique	**Recall**	**Accuracy**
Baseline	~52.4 %	68 %
Meta-features/Classifiers/Title-weight	~54 %	-
Doc2Vec	~52.9 %	-
KMeans with Word2Vec	~54.5 %	-
Tag Synonymy	~55.5 %	-
Term Affinity	~56.1 %	75 %

We can see an increase in the average accuracy from 68% to 75%. We calculated the recall@5 for Avigit et. al. on the kaggle dataset which turned out to be 52.4%. Our recommendation system provides the recall@5 of 56.1%. It is still less than the recall@5 of 59.6% of Xin Xia et. al. but they use a different dataset and direct comparison can not be made from this dataset.

Further Work:

There are many cases of co-occurence of some tags like 'android' and 'java', 'flask' and 'python' etc. This information can also be used somehow to predict the other tag if one tag has been predict. Also, there should be a way to automatically identify synonyms of tags instead of mining from stack overflow etc. by using techniques like stemming etc. Work can also be done to predict tags other than the frequent ones.

References

1. A. Goldbloom, Kaggle, http://www.kaggle.com/c/facebook-recruiting-iii-keyword-extraction, 2013, [Online; accessed13-November-2013]
2. K. Saha, R. K. Saha, and K. A. Schneider. A discriminative model approach for suggesting tags automatically for stack overow questions. In Proceedings of the Tenth International Workshop on Mining Software Repositories, pages 7376. IEEE Press, 2013
3. Xin Xia, David Lo, Xinyu Wang, Bo Zhou, Tag Recommendation in Software Information Sites , Proceedings of the 10th Working Conference on Mining Software Repositories, IEEE Press, 2013
4. Report on Predicting Tags for Stack-Overflow Questions by Schuster et. al.: http://cs229.stanford.edu/proj2013/SchusterZhuCheng-PredictingTagsforStackOverflowQuestions.pdf
5. Clayton Stanley and Michael D Byrne. 2013. Predicting tags for stackoverflow posts. In Proceedings of ICCM 2013
6. Harvard Project Resport by Rajiv Tarigopula: http://scholar.harvard.edu/rajiv/classes/materials/data-science-final-project