Identify Keywords for Stack Exchange Questions

CS299 Final Project
Guoxing Li {guoxing@stanford.edu}**, Chi Zheng {zheng29@stanford.edu}**

I. ABSTRACTION

The purpose of this project is to build a keyword identification system for questions in major Stack Exchange websites to improve the efficiency of question-answer cycle.

We leveraged a wide range of knowledge in machine learning and information retrieval to develop a robust keywords identification system. In particular, we adapted the Tf-Idf[1] heuristic for feature selection, investigated linear regression, logistic regression, and SVM for predicting number of keywords (tags), and also implemented Naive-Bayes, Labeled Latent Dirichlet Allocation(L-LDA)[2] algorithms for identifying keywords. After varies experiments, we analysed the results and derived conclusions.

II. INTRODUCTION

I. Stack Exchange Websites

Stack Exchange is an emerging network of over 100 question and answer sites that cover a wide spectrum of topics ranging from cooking to traveling to software programming. Aiming to establish a valuable knowledge network, it offers high quality answers to critical questions in diverse fields.

II. Motivation

The question volume of stack exchange websites increases drastically with the popularity of internet. An automatic key words identification system will benefit both the people who asked the questions and those who answered them. The system can generate automatic tags for questions to better categorized questions. This process simplifies the process to match people with right knowledge to specific questions.

III. EXPERIMENTAL METHODS

In this section we will discuss the key challenges we face and the corresponding machine learning methods we applied to solve those challenges. For each methods, we will introduce its mathematical logic and in detail explain the way it has been applied to our task.

I. Learning techniques

1) Number of tags prediction

To predict tags for a certain question, we first need to predict the number of tags. We select feature by constructing a vocabulary from training set with each word as an entry. For a training example, we then convert raw text to an array that contains the number of occurrence for each word in the vocabulary, which is used as input in our predicting model. We investigate a number of commonly used approaches to build our predictor, including both regression (linear regression) and classification methods (logistic regression, SVM). For classification method, the range of number of tags is the range seen in the training set. We also tried regression method since there's no hard limit on the number of tags for each question.

We make a further improvement on the features used. Intuitively, a word occurs multiple times in a question doesn't mean the question should have multiple tags. So instead of using the number of word occurrence as features, we clip the number to a binary value indicating whether a word occurs or not.

2) Naive Bayes for tags prediction

We choose Naive Bayes as our first attempt to solve the tags prediction problem, because of its simplicity and wide use case in text classification. Specifically, we train a collection of multinomial Naive Bayes classifiers with Laplace smoothing using training set provided.

Similar to the number of tags prediction, we construct a vocabulary and convert input correspondingly. The vocabulary is selected using the approach described in Feature selection and noise reduction section. To predict

different tags using Naive Bayes, we construct the following algorithm.

- Construct a tag set by collecting all tags in the training set. The tags predicted will be within this tag set.
- Build a vocabulary by selectively taking words from training set (details in feature selection and noise reduction section).
- Train a Naive Bayes classifier for each tag in the tag set using the vocabulary. The classifier for each tag tells the probability of a question having this tag. Before feeding our training data into classifiers, we pre-process input text using the same filtering method used as feature selection so that words in input can be best matched with corresponding entries in the vocabulary.
- We also train a model discussed in previous section to predict the number of tags for each question.
- To predict, given a testing example, we run all classifiers on the example, and choose k tags with the highest probability, where k is determined by our number of tags predictor

The time complexity of our approach is $O(M \times N \times V)$, where M is the size of training set, N is the size of tag set, and V is the size of vocabulary. Given that there could be millions of questions, thousands of tags and thousands of entries in the vocabulary, the Naïve Bayes approach we used here is rather time-consuming.

3) Labeled Latent Dirichlet Allocation (L-LDA)

Labeled LDA is a supervised topic model generated from LDA[3] for credit attribution in multi-labeled corpora. We believed L-LDA is a good fit as our task is very similar to the one in the paper that introduced L-LDA.

In this model, a document is viewed as a combination of topics, each of which consists of a word distribution. In our case, the topics are labels in the training set. Through the training process, the algorithm build the word distribution of all the topics provided in the training set. Each document is broke down into a distribution of all topics. By selecting the topics with highest probabilities, we are able to select the top tags.

II. Feature selection and noise reduction

Applying machine learning techniques to text based data is challenging partially due to the large noise within the data. Factors such as typos, inconsistency usage of capitalization and grammar mistakes drastically increase the data noise. Specifically, text in questions at Stack Exchange can be even noisier as it may contain a large amount of code snippets, which includes unwanted words like variable names, different syntax, etc. However it is undesirable to simply take out code snippets from question body as it may drop some language/technique specific keywords. As a result, we adapted a series of approaches to remove noise by standardizing words and filtering out undesired words. The techniques we used here not only sanitize the input data, but also dramatically decreases computation time by shrinking the size of vocabulary (especially for Naïve Bayes approach).

Word Purification We try to reduce noise originates from different forms of a word by lowercasing all characters, ripping out certain punctuations, and using word stemming. Note that we don't rip out all punctuations since that will make it harder to differentiate words like C++, C, and C#.

Tf-Idf To filter out undesired words, a common way is to filter out a list of stop words (e.g. "the", "and", "you", "what"). The simple approach is not good enough in our case, since it doesn't remove those words with really low frequency (e.g. variable names). Therefore, in addition to that, we utilize the state of the art Tf-Idf weighting heuristic: Okapi BM25[4] to discover meaningful words in each training. Note that a good vocabulary needs to be small while contain as many meaningful words as possible. The heuristic takes both word frequency in particular training example and its appearance in corpus into account to assign a score that reflects its overall expressing power.

$$IDF(qi) = \log\frac{N - n(qi) + 0.5}{n(qi) + 0.5}$$

$$score(D, qi) = IDF(qi) \times \frac{f(qi, D) \times (k1 + 1)}{f(qi, D) + k1 \times (1 - b + b \times \frac{|D|}{avgdl})}$$

In the equation above, D is the document, $n(qi)$ is the number of times qi appear in the whole corpus. $f(qi, D)$ computes the word frequency of qi in document D. $k1$ and b are tuning parameters.

We control the size of vocabulary by specifying the percentage (p) of words selected for each question. For a question with n number of words, we select $n \times p$ words with the highest score to construct vocabulary.

Title/body Separation When think about the essence underlying question title and question body, we realize that title serves as a summarization of the question people want to ask, and body describes the question in more details. So given the same amount of words, it's generally true that title is more informative than body. Therefore, for the Naïve Bayes approach, we tried to leverage this property by prepending "Title_" with title words, and "Body_" for body words to separate title and body words when constructing the vocabulary. However, that didn't work as we expected (the result is worse than the one without separation), which we think is because that the approach leads to over-fitting problem. We then tried another method, which is to select more words in title than those in bodies by assigning a higher Tf-Idf score to title words. This method gives us slight performance improvement under certain circumstances (especially when training set is small), but is still outperformed by the model without separation in general.

IV. RESULTS

I. Data Set

The data set is directly taken from the open challenge posted by Facebook, Inc at kaggle.com [5]. Each entry in the training set corresponds to a question posted by users in stack exchange websites. It consists of four fields: question identification, title, content and actual tags. In contrast, each entry in the test set contains all fields except for the ideal tags. The tags, ranging from one to five depends on the question, are the keywords or summary of the content. The training set is of size 7.26 GB; the testing set 2.36 GB.

Due to the large size of the dataset and high computation complexity of our algorithms, it's impractical to train and test with all of the data. As a result, we only conduct our experiments with a relatively small data set.

II. Evaluation

Our experiments are designed to tackle the following three major challenges we faced in developing the keywords identification system. 1). Find the optimal method to predict the ideal number of tags for a specific question. 2). Find an optimal learning techniques to predict tags. 3). Discover feature selection schemes to improve the learning method's accuracy and efficiency. In the following sections, we will discuss our approaches that address these challenges in detail. By analysing the results of these experiments, we are able to derive a best model.

1) Predict Number of Tags

To find the optimal methods for predicting number of tags given a specific question, we experimented different methods including Linear Regression, SVM and logistic regression. We developed two versions of each method – one with clipping, where we clip number of words occurrence to binary value, and one without clipping. For each method, we varies the number of words used as features and recorded the corresponding mean square error (MSE). Both our training and testing set have 1000 examples. The experiment results are shown in Figure 1.

By analysing the figure, we drew the following conclusions. We discovered logistic is the most effective method with small number of features while SVM being the most effective one overall (least MSE). This result aligns with the fact that SVM tend to have better performance compared to logistic regression when the data dimension is high. As the number of features increases, the mean square error of our experiments decreases. The error drops drastically when the number of features used is smaller than 3800, but much slower after that.

Finally, our experiments indicates that methods with clipping outperforms their counterparts without clipping. This suggest that duplications of words in a single question didn't carry additional information. In fact, the duplication will increase error of prediction. Besides, the minimum MSE we have is around 2, which is not quite satisfying.

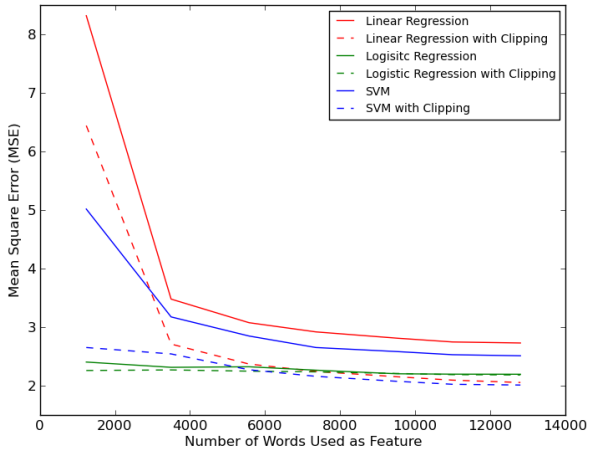

Figure 1: Number of Tags Prediction

2) Find Optimal Learning Method

To find an optimal learning method, we compared Naïve Bayes and L-LDA using precision, recall and F1 score as our error metrics. We compared these methods to our baseline method in which we assign each tag a probability of occurrence that is proportionally to its number of occurrence in the training set. Given a question, we randomly predict the tags based on their assigned probability. Our experiments are conducted with 1000 training questions. To reduce computation time, we only consider 50 most frequently appeared tags. Note that since there's no way for our algorithms to predict a tag that was never seen before, we only tested on 412 examples whose actual tags are within the 50 tags we selected. We always choose 2 tags for each testing example, and pre-process input using the same technique to eliminate other factors.

As shown in Table 1, both L-LDA and Naïve Bayes significantly outperform the baseline method. In this experiment, Naïve Bayes outperforms L-LDA. We think this is because the limited number of words, especially after selection, mitigates L-LDA's advantage in topic discovering. In most cases, each topic contains less than ten words. However, L-LDA outperforms Naïve Bayes in computational efficiency by roughly 5 times. As number of training tags increase, we expect L-LDA to be a more optimal algorithm from a computation efficiency perspective.

	Baseline	L-LDA	Naïve Bayes
Precision	0.09	0.25	0.45
Recall	0.12	0.39	0.61
F1-score	0.10	0.30	0.52

Table 1: Different Learning Methods

3) Discover feature selection mechanisms

To improve the accuracy and efficiency of the learning method we discovered in the previous section, we conducted experiments to discover the most optimal feature selection mechanisms. To test the power of Tf-Idf, we implemented a Naïve Bayes model that constructs vocabulary by selecting words with top frequency while filtering out stop words. We trained our Naïve Bayes model with 2000 questions on 50 most frequent tags. We test the classifier with 924 examples whose actual tags are within the training tag set. To eliminate other factors, instead of using our model to predict number of tags, we always choose k tags where k is the actual number of tags for a specific testing example. We used F1 score as our measure and plot it against the number of words used as features with different feature selection techniques.

From Figure 2, we observed that basically for all classifiers, the score increases as the number of features increases before it reaches 2000, and decreases thereafter. When the feature number is relatively small (less than 2000), as the number of feature increases, our classifiers get more information (under-fitting). However, once the feature number surpass 2000, noises in the additional feature outweighs the information they provide (over-fitting). Therefore we concluded that 2000 is the optimal feature number for this experiment setup. The title and body context separation methods doesn't live up to our expectation, failing to outperform the one without separation. This might due to

the fact that Tf-Idf heuristic already selects the meaningful words for the classifier. Selecting more words from title by manually assign a higher value for title words may inadvertently lose some information from body given the same amount of features used. Model without Tf-Idf and purification has a much lower score compared with the full model, which shows the effectiveness of these two techniques. Furthermore, though not shown in the figure, the classifier with Tf-Idf heuristic significantly reduces the computational complexity as it reduce feature size 10 to 20 times.

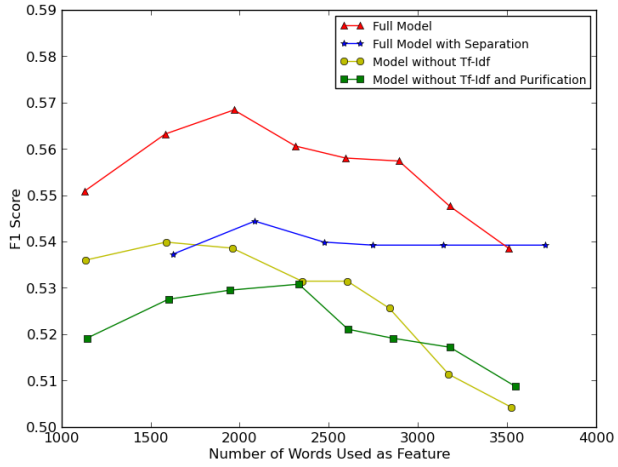

Figure 2: Feature Selection Comparison

III. Final Result

By combining the best number-of-tags predictor (SVM) and the best tags classifier (Naïve Bayes), we derive the final result on the same training and testing set.

	K-tags-predictor	5-tags-predictor
Precision	0.47	0.27
Recall	0.57	0.80
F1-score	0.52	0.34

Table 2. Testing result of best model

In Table 2, K-tags-predictor is utilizing SVM with clipping to predict number of tags, where 5-tags-predictor is always predicting 5 tags. It should be not surprising that 5-tags-predictor has a higher recall and a lower precision since most of the questions have less than 5 tags. We test and show the 5-tags-predictor mainly because its extraordinarily high recall suits fine in a real world use case. Note that this project can provide a great way to recommend tags for questions. It's important that the recommended tags contain the ideal tags, which can greatly improve user experience by saving user's effort on typing in those tags manually. As a result, high recall is desired in this specific use case.

V. CONCLUSION

In this project, we explored a number of techniques to best identify keywords in Stack Exchange questions. The project is basically divided into two parts, namely predicting number of tags, and identifying ideal tags. We conducted various tests, and finally found that the combination of SVM number-of-tags predictor and Naïve Bayes classifier gives us the best result.

We realize that our method is by no means perfect. There are two major constraints, which are its high time complexity and its limit on not able to predict tags never seen before. However, if it's released to real production, the algorithm can be run using a cluster of powerful computers, which will make it usable in real-time (training can be done constantly without influencing user experience). Also, by training on a larger tag set, the model will be able to capture most of the tags, since, though the tag set is not fixed, it's not actually changing frequently.

REFERENCES

[1] Jones KS (1972). "A statistical interpretation of term specificity and its application in retrieval". Journal of Documentation 28 (1): 11–21. doi:10.1108/eb026526.

[2] Blei, David M.; Ng, Andrew Y.; Jordan, Michael I (January 2003). "Latent Dirichlet allocation". In Lafferty, John. Journal of Machine Learning Research 3 (4–5): pp.993–1022. doi:10.1162/jmlr.2003.3.4-5.993.

[3] Ramage, D; Hall D; Nallapati R; and Manning, C. Labeled LDA: A supervised topic model for credit attribution in multi-labeled corpora.

[4] http://en.wikipedia.org/wiki/Okapi_BM25

[5] https://www.kaggle.com/c/facebook-recruiting-iii-keyword-extraction