Final Report: Keyword extraction from text

Gowtham Rangarajan R and Sainyam Galhotra

December 2015

Abstract

We have devised a model for tagging stack overflow questions with keywords which help classify them into meaningful categories. We have explored various features like bigram, code based, length of text and models like naive bayes, logistic regression and search based approach (tf-idf) . Topic models were also explored and used as another feature to boost the discriminatory power of the model. Multilabel logistic regression along with these features is the technique that came out to be the best for this task. We dealt with challenges of cleaning the large dataset, running online LDA over it and efficiently implementing the models that scales to large dataset.

1 Introduction

There is a huge amount of text available over the social media and stack exchange websites. Administering such large corpus automatically is an important task for efficient management of the documents. In this project we confine ourselves to automatically tagging stackoverflow questions with keywords. Such keywords help in broadly identifying the language being talked about, the problem type which is discussed and some more fine grained categories to which this question may belong to. The keywords have been annotated by users to help categorize the text for ease of management and we wish to reproduce the same keywords automatically using machine learning techniques. These keywords are supposed to be fine

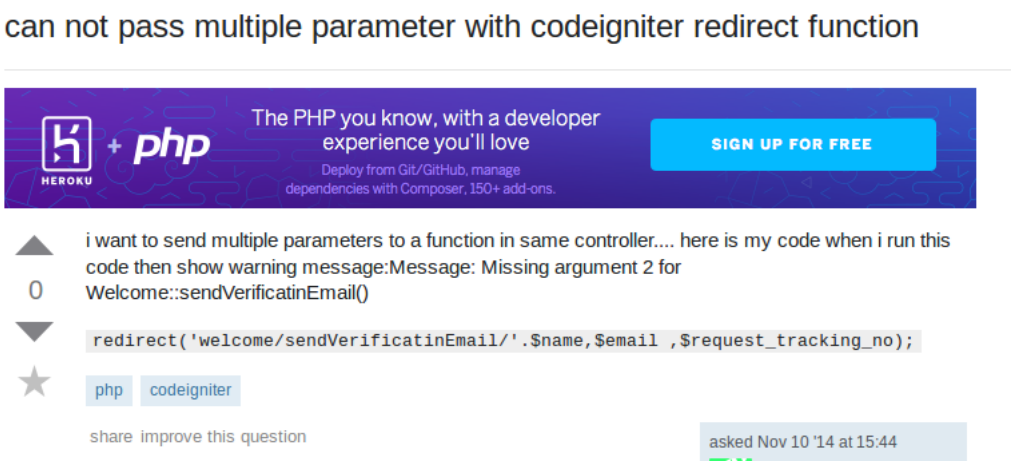

Figure 1: Sample data

grained as compared to the topics extracted by topic models as they help us categorize the documents efficiently. User queries (on the database) mostly fall into these categories and such categorization come handy while retrieving information. Prediction of such keywords with high accuracy will help automate this categorization.

To this end, we start with an example of the problem which motivates the problem and the difficulties involved in this task. Consider an example question from stack overflow [9] website which contains question title and the body of the question as represented in Fig 1. It can be seen that the title present along with body relates to the keywords (php and codeigniter) decently well. The challenge here is also evident as the keywords (php) is not present in the text of the question and is quite difficult for a human to predict that!

The task of tagging each question with these keywords automatically has various challenges to be addressed in order to form an automatic system. Some of the challenges are :

- Diverse categories : The number of keywords which can represent are huge and of different nature eg. some are inferred from the programming language, some from tool and some from the nature of the problem.
- If we look at it as a multi class classification problem, then the number of classes are

huge, making it difficult to model it as a multi class classification problem.

- The context available form the question text and body is not huge in many documents as the average document size in our corpus is roughly 150 words.
- Each document can be represented by multiple keywords and predicting how many keywords would represent the document is itself challenging.

These are some challenges which needs to be addressed to devise a model which can predict keywords for a given document. In addition to these, we had some more difficulties in devising a model for the stack overflow dataset. Some of them are :

- The dataset is quite huge as we discus it later in Sec. 3. This poses scalability issues during the training phase itself.
- The set of keywords is quite sparse as there are some keywords which is not present in a large corpus, making the prediction difficult for these keywords.
- The presence of some redundant information in the data and efficient cleaning strategy was needed to remove stop words and html keywords which act as a noise.
- The feature vector generated for a document is huge and sparse too as the vocabulary size is itself quite large.
- There are a number of keywords present in the data which are confusing eg. c and c++. These keywords are difficult to differentiate for a human too, making it impossible for the model to predict correctly.
- The bosy of question consists of code along with text and the kind of features, we need to extract from code should be different form that of text. There are many things in code which act as noise and hamper the prediction task.

Even thought there are so many challenges, there are some efficient techniques which helped us deal with most of them. We will discus those in later sections of the report. There is one

feature of the data which needs a mention that there are some keywords which are highly corelated eg. C# and Android and exploiting this aspect help us boost the prediction task. We discus this in later sections in more detail.

The key contributions of our work are the following :

1. We did an exploratory analysis of the huge data, cleaned it by removing stop words, special symbols and extracting the code snippets separate.

2. Devised diverse features which take into account the bigram, code features, length of text, etc.

3. Devised models and pruning strategies for efficient prediction.

4. Explored topic models and perceptron to boost the prediction.

2 Related Work

As this problem was proposed as a part of kaggle competition [1], there have been a number of techniques which have been applied for comparison. Heymann et al. [6] published a work in which they devised models to predict keywords for a social benchmarking system del.icio.us based on page text, anchor text, surrounding hosts, and other keywords applied to the URL. This is one of the state of the art techniques being applied for tagging text documents with keywords. [9] is a work which is the same as that our problem. In this paper, the authors have devised ACT-R inspired Bayesian probabilistic model to predict the keywords. The ACT model (declarative memory retrival model) uses co-occurrences and can predict link traversal for search queries. [11] is another work on similar lines which suggests users to expand queries leveraging the keywords associated with the queries. They used SVM along with their application specific tweaks which were not useful for our problem. In our techniques, we have explored tf-idf [2] approach along with logistic regression and naive bayes models. tf-idf approach calculates the relevance of a word with a document by

calculating the inverse frequency of occurrence of a word in a document with respect to the corpus. This approach can be used to calculate the similarity between different documents.

We explored topic models to extract topic based features and used these topics as a feature to improve the discriminatory power of the model. In order to exploit this, we used LDA [4] to extract useful topics from the documents. LDA is a generative model which generates each word per topic and then produces a distribution of topics per document. The major challenge with this technique is that it takes a lot of time to run on a large dataset for a large number of topics. This is an unsupervised technique and models have been devised over it to form a supervised topic model [7] which tries to learn the topic model where we have well defined classes with us. [3] is a work which is highly related to the problem we solve as they use just topic models to predict the keywords over stack overflow. Their observation was that they were able to classify related keywords into the same category like css and css-3 into the same category. [12] is a generic approach which predicts keywords for a set of documents by replacing the unigram word distributions with a factored representation conditioned on both the topic and its structure. The reason why we donot use these techniques is that they are not scalable to large corpus of the size of millions as normal LDA with 50 topics took around a week for our data.

We looked into various techniques which have been devised using neural networks. We went through these techniques and found them to be apt for solving our problem but could not try these due the constraint of server and lack of time. [5] is the first work in the area of neural networks where they devise a technique named Ranknet which tries to model the underlying ranking function. There have been plenty of work which have built upon this work. [10] builds the ranknet technique further on the probabilistic ranking framework, and propose a novel loss function named Fidelity to measure loss of ranking. This new loss function is shown to be helpful in ranking. We felt that these techniques and improvement of ranking techniques in information retrieval would be useful for our problem. [8] build on the above discussed works and propose using query-level loss functions in learning of ranking functions. This is one of the state of the art algorithms known for ranking in the field of

information retrieval.

In the papers which we discussed, we found them very related and useful to solve our technique and build a holisitic model but due to efficiency constraints, we were not able to try out all of these.

3 Exploratory Data Analysis

We obtained an 8GigaByte data dump consisting of questions which users asked on the stackoverflow website. Each question had a title, body and few keywords associated with it.

The 8GB data contained **6,034,195** questions. Each question had maximum of **5** keywords and a minimum of **1** keyword with the average being **2.88** keywords per question. The data set consisted of **42,000** number of unique keywords and more than **200** keywords were used in atleast **10,000** questions.

We also observed that the questions had **8.34** words while the answers had **184.077** respectively. This shows that the feature vector being considered would be highly sparse as we have a total of more than 33,000,000 unique words present in the data. This means that we need to clean the data efficiently to reduce the feature space as we cannot run any technique on such a huge feature space.

Some of the keywords which were highly frequent in the data are shown in the table.

Tag	C#	Java	php	javascript	android	jquery	c++	python	iphone	asp.net
Percentage	7.72	6.79	6.51	6.06	5.33	5.03	3.3	3.1	3.06	2.9

It can be observed that the keywords which are present in large numbers are languages itself. The distribution of keyword frequency has been shown in table below for the top ten keywords which shows that there are some language keywords which are present in large numbers in the data. Since the number of keywords are huge, we try to analyse the co-occurrence of keywords which can help us figure out keywords which are somewhat related. Fig. 2. As expected keyword correlation is sparse. Interestingly, people compare c# and objective-c !

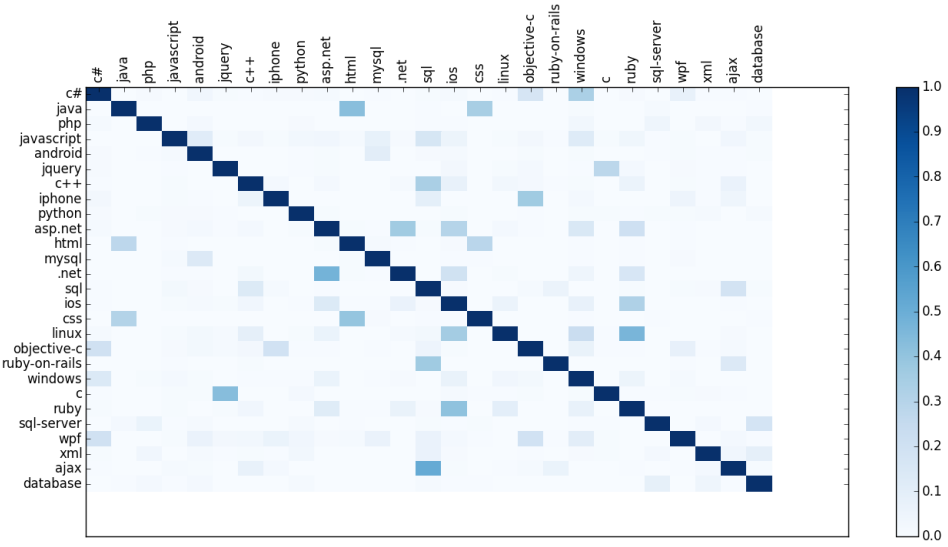

Figure 2: Corelation between top keywords

In simple terms, given the question is about C# tells a lot more information (regarding other possible keywords) than just the presence of C# keyword. This important information can be imagined to be captured well through a 'decision tree of classifiers' based model.

Few of these annotated keywords were semantically very similar hence hard to evaluate, eg., python2, python2.6. Some strongly correlated keywords were fused together to form a new keyword. eg, Questions in which facebook and ios keywords were identified were associated with a keyword called facebook-ios-sdk.

We sub-sampled 100k examples from the data set to analyse our technique. We use this subsampled dataset to weed out bad techniques and plan to run 'good' ones over the complete dataset.

4 Models

In this section, we attempt to rephrase the problem as problems that have well formed solution. We also describe the features we used to aggregate the results and the rationale

behind using the methods and features.

4.1 Baseline

One of the baseline classifier we used predicts the most frequent keyword that occurs in the training set for all questions in the test set. The other baseline is that we used, collected a set of keywords from train and predicted those keywords which are present 'exactly' in the body or title of the question. eg.,

```
find LaTeX3 manual
month ago saw big pdf manual LaTeX3 packages
new syntax think bigger 300 pages find web anyone link
```

The above question will have **latex3** as a predicted keyword since **latex3** is a keyword occurring in the train set. We refer to this as the 'Keyword baseline' in the experimental evaluation. (The above example was obtained after cleaning the data-set)

4.2 Search Based Technique

Term frequency -inverse document frequency is a statistic that is used to reflect the importance of a word is a document. The tfidf value of a word is proportional to the number of times a word occurs in a document and inversely weighted based on the number of times it occurs across documents. An example is represented as bag of words and the the tfidf value for each word is obtained using the train-set. Tf-idf based vector is constructed from the bag of words representation for a given test example. This tfidf vector is a common representation used for searching and retrieval.

We run through the training data and collect all possible keywords that can occur. We assume those keywords can be the only plausible classes and train a classifier for each keyword. We run each trained keyword classifier on the question to predict the presence of the keyword. To prune the number of possible keyword classes to look for in a test example, we compute TFIDF vectors for training data and the test example and identify '10' train

examples which are close to the test tfidf vector using cosine similarity. We collect the assigned keywords for those 10 examples and predict the presence of those keywords in the test example using our trained classifier. We experimented with NaieveBayes and Logistic regression based classifiers. We worked with features host of features on this method.

4.3 Logistic Regression Based Technique

We run through the training data and collect all possible keywords that can occur. We assume those keywords can be the only plausible classes. We then train a multi-label logistic regression classifier and squash them into a d-dimensional vector where dth dimension measures the probability of the dth keyword. We predict the top dimensions from the d-dimensional vector and assign the corresponding keyword to the test example.

5 Features

We experimented the models described above with basic BOW model features, bigram features , code based features, features that were dependant on titles only. The code extraction features included rules to identify languages like C, python, javascript, jquery.

5.1 Latent Dirichlet allocation

Latent Dirichlet allocation (LDA) is a generative model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. In LDA, each example, which is represented as bag of words, is viewed as a mixture of topics.

We introduced topic models as features. We assumed that the examples are drawn from 100 different topics. (100 was a hyperparamter. it was not cross validated) Thus using LDA an example was collapsed into a 100 dimensional vector. We ran an LDA on the data set with an intention to represent each question on a reduced dimension. This vector is then fed into the Logistic regression network as features. This feature helps in capturing correlations between output keywords.

6 Evaluation Metric

In order to compare the effectiveness of the techniques, we calculate precision, recall and f-score for k keywords where k is the number of keywords which we track. In order to define precision and recall, we take an example where we had k keywords to track say $\{t_1, t_2, \ldots, t_k\}$ and the question had keywords $\{t_1, t_2\}$ in ground truth and suppose our model predicted keywords $\{t_2, t_3, t_4\}$. Using this example, we define the following terms which are used to define precision and recall :

- tp : True positive (tp) is defined as the number of keywords which are predicted for a question and also present in the ground truth[1]. In our example we have a single true positive i.e. t_2.
- fp : False positive (fp) is defined as the number of keywords which are predicted for a question and but are not present in the ground truth. In out example there are two false positives i.e. t_3 and t_4.
- tn : True negative (tn) is defined as the number of keywords which are not predicted for a question and are also not present predicted for a question and but are not present in the ground truth. In above case, all keywords which are not predicted i.e. $\{t_5, t_6, \ldots, t_k\}$ are true negatives.
- fn : False negative (fn) is the number of keywords which are predicted negative but are actually positive eg. t_1 is predicted negative but is actually positive.

Precision is defined as the fraction of keywords, predicted positively which are actually positive. Recall is the fraction of keywords predicted positive out of the relevant keywords. Mathematically, $precision = \frac{tp}{tp+fp}$ and $recall = \frac{tp}{tp+fn}$.

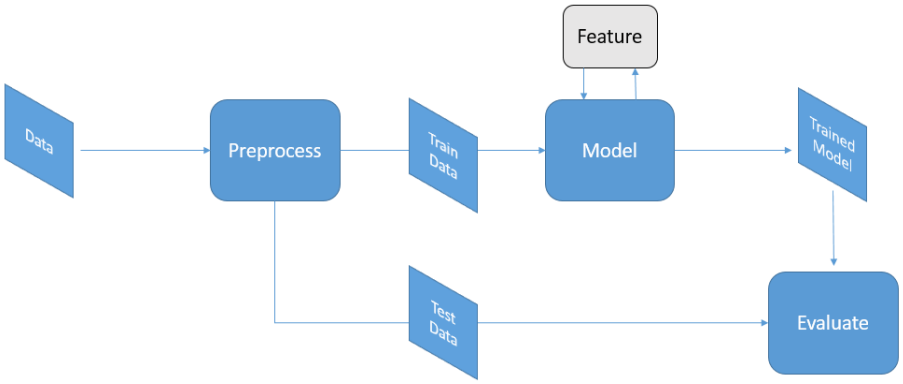

Figure 3: Implementation pipeline

7 Implementation

We implemented a system as shown in 3. Their implementation details are covered in this section.

7.1 Preprocessing Phase

Any given data sample is passed through a cleaning phase where stopwords, slangs, urls are removed and semantic information in markup keywords like 'code' for these questions, if it exists, is extracted. This stage takes in title and description and produces cleaned title, description and code. This is the cleaned form of the data which is used for learning the model.

The data passed through the cleaning phase is split into 3 sets train,test in 1:1 ratio. The train set is collected for features (a sparse matrix).

7.2 Training Phase

We implemented 2 different training and classification models. The first one is a tfidf based model that was presented in the progress report the next one is a logistic regression based one. We used both models with the features described in the previous section.

[1]These keywords are subset of the k keywords which we are tracking

7.2.1 Tfidf based model

A binary classifiers for a keyword is trained on a subset containing equal positive and negative samples. The equalized training set is extracted for features and passed through a learning method. For this report we used bow/ngram model with scikit learns's l2 regularized Logistic regression/naive bayes.

The train set is used to learn tf-idf feature for each train document. tf-idf, the acronym for short for term frequency–inverse document, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus[2].

7.3 Logistic regression based model

We extracted the top 10 keywords from the train set and trained a logistic regression classifier by replicating the examples such that we have 1 keyword associated with an example. The output of the Linear regression is a k-dimensional vector indicating the probability of the k-keywords in the example. We used the bfgs optimization algorithm to train the logistic regression.

On seeing a test example we ran the trained network and predicted the top/ top2 keywords obtained from the probability of the presence of those keywords.

We used the server provided by umass CICS to perform our experiments.

8 Result

We cleaned the whole dataset by removing the stop words and slang, extracting code along with some more cleaning. Out of this cleaned dataset we extracted a small set of 100K questions and split into two equal halves to train and test our techniques and features.

8.0.1 Feature Engineering

First we try to fix a feature which is the most useful out of all the variations which we tried. In order to check which features actually helped, we compare the precision recall values for

the same. The different set of features are :

- Title: This used a bag of words of the title of the question asked by the user
- Bow_complete: This refers to unigram feature over the title, body of the question and the code.
- Bow_split: This feature gives different weights to the the title and body words in the question.
- Bow_bigram: This consists of bigram in the bosy of the question and unigram of the title.
- Bigram+rules : We ran some simple set of rules over the code to identify the language from it ans used this as a feature.

There are some more features which we tried like the number of capital letter words, number of integers, number of special characters, etc but as they did not help much, we donot report them here. Also there are some more sophisticated features which we need to devise and is the plan of the next part of our experimentation

We present the confusion matrix along with precision, recall and the f-score to compare with. The table compares the logistic regression with all these set of features.

Feature	TP	TN	FP	FN	precision	recall	f-score
Title	553	27726	1452	269	0.28	0.67	0.39
Bow_split	588	28047	1131	234	0.34	0.72	0.46
Bow_complete	592	28127	1051	230	0.36	0.72	0.48
Bow_bigram	593	28238	940	229	0.39	0.72	0.50
Bigram+rules	593	28232	946	229	0.39	0.72	0.50

It can be observed that the performance of the classifier with bi-grams performed the best which is as expected because there are many keywords like ‘windows 7’ which are captured through presence of both words together. Also another thing to observe is that the rules

applied over the code help us improve the classification. We plan to improve the rules as we look more into the data which will help us improve the precision and recall.

Now since we have fixed the feature, we compare the results of logistic regression with that of the two baselines which we implemented.

Feature	TP	TN	FP	FN	precision	recall	f-score
Keyword_baseline	505	28055	1123	317	0.31	0.61	0.41
Bow_bigram	593	28238	940	229	0.39	0.72	0.50

This shows that the logistic regression helps us produce decent improvements over the baseline which predicts just the tag which is present in the text of the question.

Moving forward we tried the multilabel logistic regression model with the features which we discussed above and found the results below. Along with the above mentioned features, we incorporated the topics generated by LDA along with bigram and found the best results.

Feature	TP	TN	FP	FN	precision	recall	f-score
BOW_Multi label LR	635	28055	923	393	0.41	0.79	0.53
Bow_bigram_Multi label LR	855	28240	835	208	0.51	0.80	0.62
Bow_bigram_Rules_Multi label LR	868	28242	823	206	0.52	0.80	0.62
Bow_bigram_LDA_Multi label LR	921	28246	769	200	0.55	0.82	0.66

9 Observation and Analysis

From the experiments, we made some observations which are in accordance to what was expected. Some of them are :

- Bigrams are better features than bag of words and are needed to get decent results because of presence of things like eg. windows 7, python 2.7, objective c, etc.
- Code specific features are important to extract programming language eg. the figure 4 below has keywords which are impossible to be predicted by looking at the text. Also the programming language is one tag which can be predicted with decent correctness from the code.

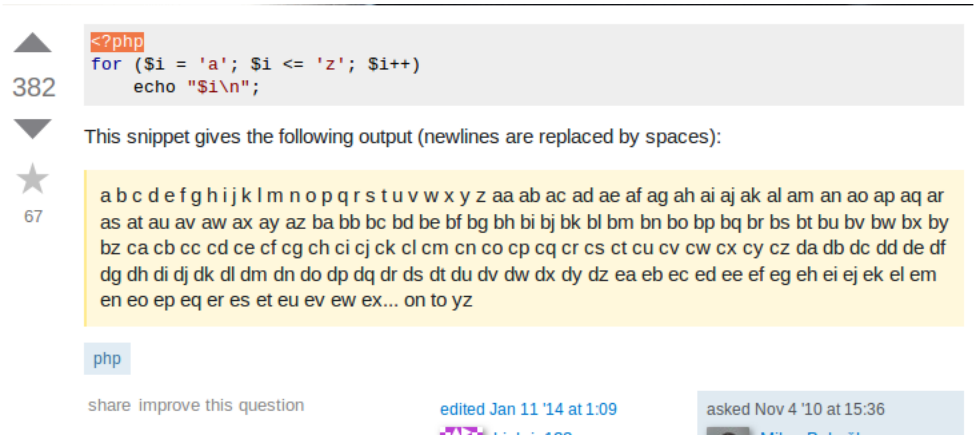

Figure 4: Sample data

- Logisitc regression is better than naive bayes becasue it is hard to balance the samples in naive bayes. This is because the train data is highly skewed (very less number of positive questions and large number of negative questions)

- tf-idf has been used for pruning and is highly efficient to reduce the number of classifiers being asked to predict the tag. Also we find the quality of prediction is not hampered much even on using this approach.

- Corelation of keywords helps us in prediction and lda has been used by us to capture this corelation. The improvement which is visible by incorporating LDA explains the benefit of using a topic model. We did not exploit the benefit of LDA completely as we our experiments for top 20 keywords and there are not many corelations in these keywords.

- Since multinomial logistic regression is better than training individual models per tag, we feel multilayer perceptron would be a better model to try the technique on. Also an ensemble of these techniques is one thing which we feel could be a good idea to get the best results.

9.0.2 Mis-classified examples

We analysed some of the mis classified examples to see if we could improve features to improve classification. Some of them were found to be really hard for humans to annotate too. The examples are :

- **Text**: CFInput autosuggest value', 'used simple CFInput autosuggest code copied Ben Forta blog works ok need one additional feature user used autosuggest field choose something would like populate second form field result doesn work like Javascript using onChange value property field seems value property contains original user input chooses autosuggest list life find copy chosen autosuggest value another field using Javascript Anyone **Tag**: 'ajax' **Predicted** : 'Javascript'

- 'C using E mathematical equation', 'trying solve aX 2 bX p 0 seem make work using math header supposed use printf' **Tag** : 'C++' **Predicted** : 'C'

- **Text**: 'Error Handling Linux Daemon', 'writing server Linux daemon want know protocol UNIX Linux community daemon encounters fatal error eg server failing listen segmentation fault etc already done whole thing system log want know fatal error log keep running infinite nothing loop log exit standard thing daemon written C using custom exception system wrap POSIX error codes know things fatal' **Tag** : 'c++' **Predicted** : 'C'

10 Future work

Since we feel that multi layer perceptron would have helped us in improving the prediction, this is one technique which is definitely a followup on this work. Also we feel some more features, increasing the number of topics in LDA would have improved the discriminatory power of the model. Last of all, ensemble of all the technqiues would have definitely helped us boost our performance.

11 Conclusion

Tagging stack overflow questions with meaningful keywords is a challenging task and needs proper feature engineering and model selection to get decently good precision. We observed that bigrams are definitely needed to improve the prediction and topic modelling helps in improving the discriminatory power of the model. Overall we feel that the prediction task can be improved further with proper feature engineering and using an ensemble of the techniques.

References

[1] Identify keywords and tags from millions of text questions. https://www.kaggle.com/c/facebook-recruiting-iii-keyword-extraction.

[2] Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning. 2003.*

[3] Miltiadis Allamanis and Charles Sutton. Why, when, and what: analyzing stack overflow questions by topic, type, and code. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 53–56. IEEE Press, 2013.

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

[5] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pages 89–96. ACM, 2005.

[6] Paul Heymann, Daniel Ramage, and Hector Garcia-Molina. Social tag prediction. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 531–538. ACM, 2008.

[7] Jon D. Mcauliffe and David M. Blei. Supervised topic models. In J.C. Platt, D. Koller,

Y. Singer, and S.T. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 121–128. Curran Associates, Inc., 2008.

[8] Tao Qin, Xu-Dong Zhang, Ming-Feng Tsai, De-Sheng Wang, Tie-Yan Liu, and Hang Li. Query-level loss functions for information retrieval. *Information Processing & Management*, 44(2):838–855, 2008.

[9] Clayton Stanley and Michael D Byrne. Predicting tags for stackoverflow posts. *Proceedings of ICCM*, 13.

[10] Ming-Feng Tsai, Tie-Yan Liu, Tao Qin, Hsin-Hsi Chen, and Wei-Ying Ma. Frank: a ranking method with fidelity loss. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 383–390. ACM, 2007.

[11] Jian Wang and Brian D Davison. Explorations in tag suggestion and query expansion. In *Proceedings of the 2008 ACM workshop on Search in social media*, pages 43–50. ACM, 2008.

[12] Xiaojin Zhu and David Blei. Taglda: Bringing document structure knowledge into topic models. Technical report, 2006.