#ML #NLP: Autonomous Tagging of Stack Overflow Questions

Mihail Eric, Ana Klimovic, Victor Zhong
{meric, anakli, vzhong}@stanford.edu

December 8, 2014

Abstract

Online question and answer forums such as Stack Exchange and Quora are becoming an increasingly popular resource for education. Central to the functionality of many of these forums is the notion of tagging, whereby a user labels his/her post with an appropriate set of topics that describe the post, such that it is more easily retrieved and organized. We propose a multi-label classification system that automatically tags users' questions to enhance user experience. We implement a one-vs-rest classifier for a Stack Overflow dataset, using a linear SVM and a carefully chosen subset of the entire feature set explored. Our classification system achieves an optimal F1 score of 62.35% on a subsampled portion of the original data restricted to 100 tags and a minimum of 500 instances of each tag across the data.

1 Introduction

With the advent of online education, question and answer forums are becoming an increasingly popular resource for information. Examples include Stack Exchange, Quora, and forums for Massive Open Online Courses (MOOCs) like Coursera and OpenEdX. While the quantity of information available on these forums is steadily increasing, there is currently no efficient and automatic way of grouping and classifying the information so that it can be displayed to users in an intuitive way. It would be useful to automatically infer and tag the topic of a question posted on a forum. A system that automatically infers the topic of a question can improve user-experience on online forums by 1) grouping questions about common topics together for users to browse and 2) showing users' posts related to a question they are inputting, since their question may already have been answered on the forum. In order to allow the grouping of common posts, some forums such as Quora require users to manually enter tags associated with their questions. However, manually tagging a post is a burden for users which degrades the overall user-experience. We envision a platform that can infer the tags of posts automatically. To this end, we propose a multi-label classification system that automatically assigns tags for questions posted on a forum. We implement and test our classifier on a dataset of Stack Overflow questions.

The remainder of this paper is organized as follows. Section 2 outlines previous work on text classification, Section 3 describes our methodology and provides system architecture details, including feature extraction and classifier tuning. We present results in Section 4 and discuss key insights in Section 5, concluding in Section 6 with opportunities for future work.

2 Background

Text classification, the process of classifying text documents into classes based on their topic, is a common application of natural language processing and machine learning. In general, text classification is both a multi-class and multi-label classification problem, meaning there are multiple classes to choose from when labelling an input and each input might correspond to more than one class.

There are two common approaches for tackling multi-label classification [11]. One common approach, called one-vs-rest, decomposes the classification problem into k independent binary classification problems, training a separate classifier for each of the k possible output labels [1, 13]. An alternative approach for multi-label classification, collectively known as the set of adaptive algorithms, tries to predict all labels at once along with a confidence ranking associated with each label [12, 6]. Examples of adaptive algorithms include boosting and random forests [10, 3].

In our classification system, we take the one-vs-rest approach using support vector machines with a linear kernel. We also investigate lexical and syntactic features extracted from the data to enhance our model.

3 Methodology

3.1 Experimental Setup

To conduct our experiments, we use the scikit-learn machine learning suite [8] and the Natural Language Processing Toolkit [4]. All source code from the project can be found at the following Github repository: `https://github.com/vzhong/229-224N-Project`

The dataset we use to develop our classifier is from a past Kaggle competition [2]. The dataset consists of approximately 6,000,000 user questions spanning roughly 40,000 unique tags from the Stack Overflow site. Each post from the data contains a unique identifier, the title of the question, the body of the question, and the associated tags. The large quantity of data in our dataset caused memory to become a bottleneck for computing. Developing models for the full dataset was computationally intractable. Hence before testing any model or set of features, we subsampled the data, typically restricting it to only those posts with one of the 20 or 100 most common tags seen in the entire data. We also ensured that our subsampled dataset contains at least 100 training examples for each label. This ensured that we have sufficiently diverse training data so that we can learn statistics for the relevant tags.

We chose F1 score as the metric for evaluating our multi-label classification system's performance. F1 score is the harmonic mean of precision (the fraction of returned results that are correct) and recall (the fraction of correct results that are returned). F1 score is a common metric for multi-label classification and was also the chosen metric for the Kaggle competition from which our dataset originates.

3.2 Feature Extraction

The majority of our feature set corresponds to ngram features and label counts. Figure 1 depicts our complete feature extraction pipeline. In this section, we describe our process for selecting and tuning feature extractors and classifiers in the pipeline.

We find that starting with a feature set corresponding to the number of occurrences of a label in the title and body of the question (we refer to this as "label counts"), yields the highest precision of all features we tried stand-alone. Thus, to construct our complete feature set, we start with label counts, then introduce and greedily tune feature extractors (through 3-fold cross validation) one at a time.

We add body and title ngrams to capture terms that co-occur with certain tags. Hyperparameter tuning for ngrams and label counts consists of choosing binarization settings (Bernoulli vs. Multinomial counts), and IDF and norm settings for the proceeding TFIDF transformer. We use TFIDF [5], or Term Frequency Inverse Document Frequency, to reweight the count of each term in accordance with how relevant it is to each label. Unigram tuning additionally entails choosing binarization and cut-off settings for the counter.

We find that bigram features do not significantly improve performance and only hinder our ability to use a larger dataset. We also find that terms in the title are more indicative of labels because a user typically attempts to summarize and categorize the question in the title. Empirically, we discover that extracting ngrams for the body and the title separately to allow for independent weightings of terms improves performance. Extracting raw ngrams for the code portion of the questions (if such sections exist) decreases performance due to large variations in the code content. Thus, we separate the text and code portions for each question and only extract ngrams for the text portions.

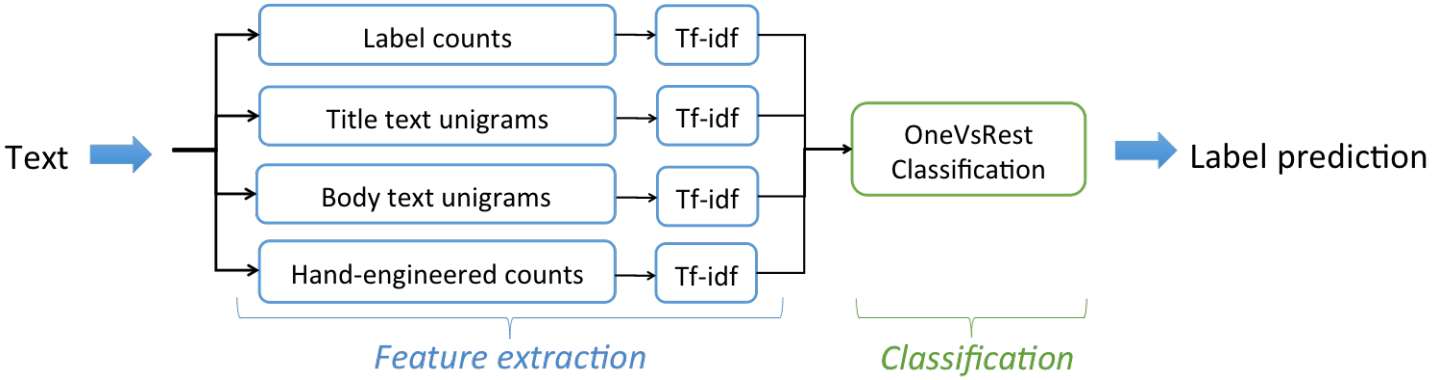

Figure 1: Feature Extraction Pipeline

Next, we hand-engineer features to make use of code portions of questions and improve performance on tags our classifier found most difficult to predict. We aim to enhance the performance on the 20 most common tags, as most examples have one of these tags (over 5 million of the questions had at least one of the top 20 tags). After tuning our system for the top 20 tags, we examined the worst performing tags and manually created simple string matches for commonly occurring syntax related to these tags and the various programming languages frequently occurring in the questions (e.g. `@property`, `#include`).

We also explored collecting unigrams over only noun terms (after POS tagging the question body) and over only named entities (after POS tagging and NER). The noun features were introduced to capture the intuition that the set of nouns in a question are a less noisy means of identifying the main entities in a question, which should be good indicators of the central topics of the question and thereby the associated tags. NER was considered because often a set of organizations such as *Microsoft* or *Google* may be indicative of certain tags like *windows*, *windows-7*, or *chrome*. The system with NER unigram counts achieved marginally better F1 scores, and the system with POS unigram counts achieved our absolutely highest F1 score. Collectively our best-performing system was trained on a training set with 176,681 questions consisting of 430,870 features and tested on a data set of 22,642 questions with the same number of features.

Finally we explored using lexer guesses from the Pygment Python package over the code portions. The lexer features were an attempt at capturing the abundance of information contained in the code portions of a question, i.e. the occurrence of certain programming syntax such as *def* or *__init__* may be strong indicators of certain tags such as *python*.

3.3 Classifiers

We use a one-vs-rest paradigm for multi-label classification. For a collection of labels $\{a_1, a_2, , a_k\}$, a one-vs-rest classifier is as a collection of k binary classifications, where for every label a_i, we treat a training example that has been labelled with a_i as a positive example and a training example that is not labelled with a_i as a negative example. Hence the output for a given example can be viewed as a binary vector of length k, $(c_1, c_2, ..., c_k)$, where $c_j = 1$ if label j has been chosen for the example and 0 otherwise.

We use a discriminative support vector machine with a linear kernel, for which we tune the loss parameter C and the loss function used (l_1 or l_2). For every different subsampled dataset and feature set we considered while developing our classification system, we tuned the classifier hyperparameters through 3-fold cross-validation with F1 score as the performance metric. In all cases, l_1 loss provided better results, with the C value varying in accordance with the number of training examples. As we increase the number of training examples, variance decreases and thus less regularization is required.

The objective function we use for our SVM linear classifier is as follows, where w, b, ξ, respectively denote the weight, bias, and soft margin variables. $x^{(i)}$ denotes the feature values (eg. unigram counts and string matches) for a given example and y^i denotes the classification output (whether a given label is assigned to a question):

$$\begin{aligned} &\min_{\gamma,w,b} \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i \\ &\quad \text{s.t. } y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i,\ i = 1, ..., m \\ &\quad\quad \xi_i \geq 0,\ i = 1, ..., m \end{aligned}$$

We also experimented with alternative classifiers. Within the one-vs-rest classification paradigm, we tried training the individual binary classifiers for each label with Logistic Regression, Multinomial Naive Bayes and Bernoulli Naive Bayes models. However, the linear SVM performance was superior. We chose to use a support vector machine for our system because we intended to make use of the model's ability to learn complex separating boundaries and its well-known robustness in learning regularities across diverse data sets. Using a Gaussian kernel for the SVM was computationally intractable for our large dataset. We also considered random forests as an alternative to the one-vs-rest family of classifiers. However, since random forests classifiers randomly sample examples from the dataset to construct each decision tree and require each tree's sampled dataset to contain positive examples for each tag, the number of examples the classifier required was intractable for our computing resources.

4 Results

We evaluate our multi-label classification system using the F1 score metric and report results for predictions of the 100 most common tags, trained on a dataset with a minimum of 500 questions for each label. We construct our test set by randomly sampling the original dataset

while ensuring a minimum of 75 questions for each label. Our multi-label classification system achieves an optimal F1 score of 62.35% on the test set.

The contribution of each type of extracted feature is summarized in Table 1 which presents the F1 scores for our classifier, starting with label counts features and gradually introducing and tuning additional feature sets discussed in Section 3.2. Label counts allowed us to achieve a 54.92% F1 score. Adding ngrams from the title and body of the question further improved performance and our hand-engineered features provided a small additional improvement. Adding NER ngrams also achieve marginal improvements, though not as large as we would hope. We found the implementation of the NLTK NER system to be unreliable, as the regularities it required (eg. proper capitalization, spelling) were often not met by the StackOverflow questions. Adding noun ngrams also produced small improvements and allowed us to achieve our best-performing system with the associated F1 in bold below. Due to time constraints of hyperparameter tuning and feature extraction, we were not able to attempt testing a system with both NER and noun ngrams.

	Features	Precision	Recall	F1
1)	Label Counts	45.36	73.62	54.92
2)	1)+ Title Ngrams	54.79	68.52	60.17
3)	2) + Body Ngrams	58.37	67.81	62.05
4)	3) + Hand-Engineered	58.56	67.62	62.10
5)	4) + NER Ngrams	58.19	68.88	62.31
6)	4) + Noun Ngrams	57.98	69.31	**62.35**

Table 1: Feature Tuning

Table 2 shows the system's F1 scores when adding additional features. We report the scores associated with these features in a separate table since none of these feature extraction techniques resulted in significant performance improvements. The Pygment lexer package relied heavily on rules such as looking for the standard Unix header `#!/usr/bin/env python` and file extensions to distinguish between languages, neither of which is common in short Stack Overflow postings.

	Features	Precision	Recall	F1
5)	4) + Lexer Guess	57.89	68.50	62.04

Table 2: Additional Features

Using the final feature set in Table 1, we show the training and cross-validation learning curves in Figure 2. As expected, the addition of more examples causes training F1 score to decrease while cross-validation F1 score increases. The relationship between the training and cross-validation learning curves suggests that our classification system has high variance. Specifically, cross-validation F1 score approaches 62% while training asymptotically approaches the significantly higher F1 score of 82%.

Finally, in Table 3, we report the precision, recall and F1 score of our system on the 5 most common tags.

	Label	Precision	Recall	F1
1	c#	52.02	61.45	56.34
2	php	68.12	73.26	70.6
3	java	68.85	69.03	68.94
4	javascript	60.30	65.36	62.73
5	android	90.90	88.49	89.68

Table 3: Performance on most common labels

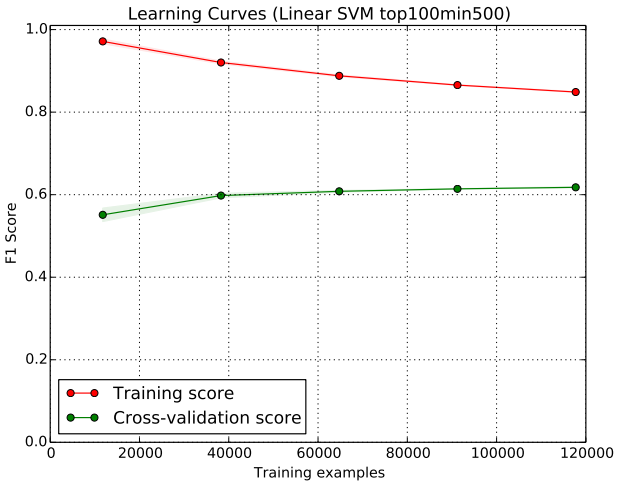

Figure 2: Learning curve

5 Discussion

Memory constraints were a major challenge we continued to encounter while developing our multi-label classification system. Our dataset was too large to fit in RAM so we were forced to subsample the dataset and deal with sparse matrices in our pipeline. As a result, we were not able to benefit from the use of certain machine learning libraries in sklearn such as PCA, Z normalization, and Gaussian-kernel SVMs which required dense matrices.

We found that extracting relevant and useful natural language features was quite difficult given the nature of Stack Overflow questions, which tend to vary greatly both in length and in content. Moreover, the

uniqueness of source code makes it challenging to construct features more complex than string matching of common syntax. While techniques such as parsing and named entity resolution improved performance, they did not provide significant gains due to the noisy nature of the data. Relevant tags are often very domain-specific and will, more often not, be directly mentioned by name in a question. Hence, it is often quite effective to just do very precise keyword matching.

To debug our multi-label classification system, we examined which of the 20 most common tags had the worst performing classifiers (lowest F1 scores). We identified *html*, *windows* and *.net* as problematic tags for prediction. Hand-engineering features specifically associated with these tags, such as counting the number of occurrences of `&lt`, `&gt` and other HTML syntax to improve prediction of the *html* tag, marginally helped overall performance. We hypothesize that tags such as *windows* and *.net* are difficult to predict since they are closely related to other tags such as *asp.net* and *windows-7*, respectively. In fact, the tags exhibit a hierarchical relationship, meaning a question tagged as *windows-7* is also *windows*. A fundamental limitation of our one-vs-rest classification system is the assumption that all tags are independent.

6 Conclusion and Future Work

We implement a one-vs-rest classification system to automatically tag online forum question topics. Our classification system, implemented using a linear SVM model and carefully selected features, achieves an optimal F1 score of 62.35% for a sampled portion of the dataset with 100 of the most popular tags and a training set consisting of at least 500 examples of each tag.

As future work, we plan to address the high variance our classifier exhibits through feature selection techniques such as principal component analysis (PCA). We also plan on exploring additional features through the use of distributed representations of words. For example, we can train a neural language model over a corpus and consider as the ngrams vocabulary the set of labels and the set of k words that are most semantically similar to each label. We began exploring this approach using word2vec [7] on the StackOverflow dataset itself, but could not produce word vectors that yielded meaningful nearest neighbors. This approach can probably be improved with further tuning and the use of pre-trained vectors [9, 7].

Considering the difficulty in classifying general tags such as *windows*, we believe that taking into account the hierarchical nature of the tags by modeling tag correlation could benefit performance. We can also explore alternative, non-linear classifiers such as Gaussian kernel SVMs and neural networks, since non-linearity might allow for a better separation of the data.

References

[1] Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features.

[2] Kaggle (2013). Facebook recruiting III - keyword extraction. `https://www.kaggle.com/c/facebook-recruiting-iii-keyword-extraction`.

[3] Klassen, M. and Paturi, N. (2010). Web document classification by keywords using random forests. In *Networked Digital Technologies*, volume 88 of *Communications in Computer and Information Science*, pages 256–261. Springer Berlin Heidelberg.

[4] Loper, E. and Bird, S. (2002). Nltk: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*, ETMTNLP '02, pages 63–70, Stroudsburg, PA, USA. Association for Computational Linguistics.

[5] Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.

[6] McCallum, A. K. (1999). Multi-label text classification with a mixture model trained by em. In *AAAI 99 Workshop on Text Learning*.

[7] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

[8] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[9] Pennington, J., Socher, R., and Manning, C. D. . (2014). Glove: Global vectors for word representation. Conference on Empirical Methods in Natural Language Processing.

[10] Schapire, R. E. and Singer, Y. (2000). Boostexter: A boosting-based system for text categorization.

[11] Tsoumakas, G. and Katakis, I. (2007). Multi-label classification: An overview. *Int J Data Warehousing and Mining*, 2007:1–13.

[12] Ueda, N. and Saito, K. (2003). Parametric mixture models for multi-labeled text. In *In Advances in Neural Information Processing Systems 15*, pages 721–728. MIT Press.

[13] Zhang, M. and Zhou, Z. (2007). Ml-knn: A lazy learning approach to multi-label learning. *Pattern Recognition*, 40:2007.