Tag Prediction from Stack Overflow Questions

Jalal Buckley (jalalb91), Kevin Fuhs (kfuhs), Reid M. Whitaker (reidw)

June 12, 2019

1 Introduction

Keyword assignment is a central task in informational retrieval and natural language processing. Automated tagging is one subtype of keyword assignment, and is itself an important type of text classification, with applications for online fora and for the automated filtering of documents.

In order to test the performance of different machine learning models for keyword assignment, we predicted the tags for questions posted to the online forum Stack Exchange. Specifically, we investigated the use of supervised learning techniques with a dataset of questions posted on the online question and answer forum Stack Exchange. We compare using Logistic Regression, Support Vector Machines, Multinomial Bayes, and Neural Networks in order to predict tags. [2]

2 Related Work

Assigning keywords to short text snippets, such as questions, is a well established research area both because of the direct utility of keyword assignment and because keyword tagging often enables further processing. Marina Litvak and Mark Last [7] and Tanya Gupta and Gurukul Vidyapeeth [5] provide recent surveys of techniques for keyword extraction.

Eibe Frank et al. designed an early keyphrase extraction algorithm called KEA, based on KFIDF and distance from the beginning of the document [4]. Later scholars have improved on these base methods by incorporating additional semantic information into the models [6, 10]. Much of this research has focused on identifying keywords for large documents, including published articles, rather than identifying tags for short quantities of text [9]. Kuo Zhang et al. applied SVMs to this task [11].

Recently, focus on keyword extraction has been largely focused on unsupervised methods [3]. In their review piece Litvak and Last pay special attention to unsurprised graphical models [7], of which TextRank is the most prominent [8].

3 Dataset

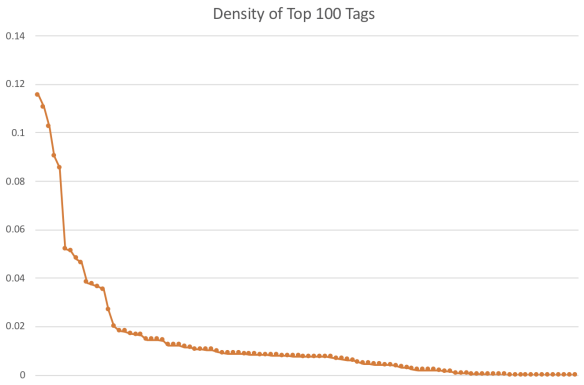

Figure 1: Cardinality of tag occurrence in the dataset, normalized by the total size of the dataset after removing duplicate entries

3.1 Data Distribution

Our data consists of 6,034,196 questions, titles, and tags from the site Stack Overflow, a question and answer site for programming questions. Each question has between one and five tags; tags represent the themes and keywords that appear in a question. In order to avoid computation constraints with training and evaluation of the data, and in order to properly bound the problem space, we decided to limit our tag prediction to the top 1,000 most commonly occurring tags, and conducted training

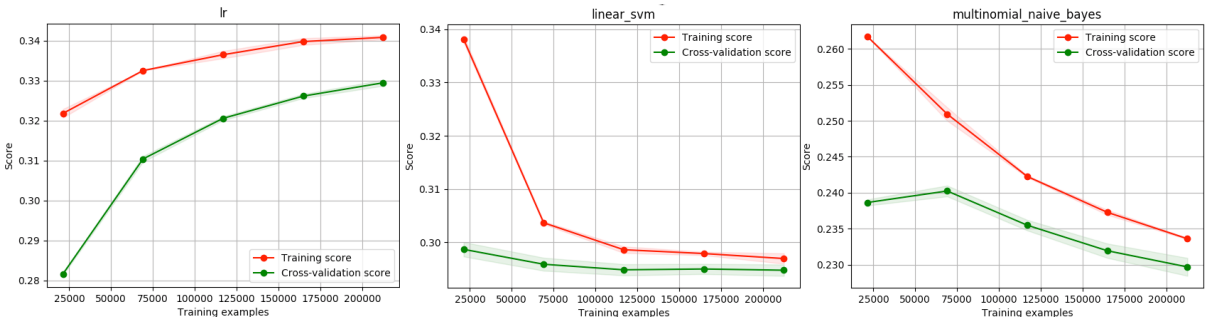

Figure 2: Learning Curves [1] for Models Learning Linear Decision Boundaries. Score represents subset accuracy, which requires that all labels are correctly predicted.

and testing on a subset of 700,000 rows. We removed all questions that did not have a tag from the 1000 most common tags or which were duplicates. Duplicate questions accounted for around 10% of the original dataset.

3.2 Feature Selection

In order to create features from the question and title text, we first cleaned question text by removing all html code and markup tags, removing stop words, and replacing all words with their stems. We then created features from text by creating vectors for which a single column represents the presence of a word in the text vocabulary. We then selected the top 1000 features using Univariate feature selection. We did this for both computational and ideological reasons. The size of the complete vocabulary is very large, and we specifically wanted our algorithms to learn the connections between programming-related tags and the programming-related vocabulary which commonly occurs in online forums. We then weighted these vectors using TFIDF in order to avoid giving precedence to common English words in our classifiers.

Initially, we experimented with concatenating title and question text, and then creating features using the above methodology. However, we found that creating features using question text and title text separately, and then concatenating the resulting feature vectors was more effective. This is likely because words in the title are more important, and thus separating title and question words allows the model to weight title words more effectively. In order to create features from title text, we created vectors such that a column represented the presence of a word in the vocabulary. We used a vocabulary size of 1000 words. Note that we did not use TFIDF to weight title feature vectors. This strategy improved our models' performance, as discussed later in the report.

4 Methodology

4.1 Set Up and Evaluation

There are several unique challenges associated with multi-label classification. Because we needed to predict multiple values (i.e. a probability indicating the likelihood of a given tag) per question, this in turn meant that we needed to either have multiple classifiers (one for each label), or we need to have a model architecture suited to a multilabel setting (such as multitask learning). We opted for the former, using the one-vs-rest paradigm in which a separate classifier is trained for each label.

Since this is a multi-label classification task, we needed to choose an appropriate set of evaluation metrics that properly weigh precision and recall. The most common is F1 score, which can further be split into micro-averaged and macro-averaged. Macro computes an F1 score for each label and averages across all the labels. Micro does so using a weighted average, where common labels receive more weight. We use both in this report, since both are useful in order to understand classifier performance.

4.2 Model Architecture

We experimented with Logistic Regression, Linear SVM, and Multinomial Bayes using the One vs Rest strategy, whereby a separate classifier is created for each tag. These models fit linear decision boundaries to the feature space, and assign probabilities for each tag. We also experimented with a simple Neural Network. Our Neural Network consisted of an input and hidden layer, both using the ReLU activation function, and an output layer using the sigmoid activation function. We use the sigmoid activation function instead of Softmax, because unlike in Multi-class classification, where each example can only have one result class, in Multi-label classification each row can potentially have multiple result classes associated with it.

5 Results

5.1 Initial Impressions

We began by training a baseline version of our models, and improved them through failure analysis. Learning curves can be found in Figure 2, and metrics can be found in Table 1. In these graphs, score represents subset accuracy, wherein we require all labels to be correctly predicted.

Note that although the cross validation score improves with more training examples for Logistic Regression, it actually seems to decrease with Multinomial Bayes. We investigated why this would be the case, since the model should improve with more training examples. Because we are using subset accuracy, note that it is possible that the performance of individual tag classifiers improved, but other tag classifiers also worsened, causing overall subset accuracy to decline. Our analysis showed that in general Multinomial Bayes seemed to predict more tags than Logistic Regression, causing it to perform poorly on certain examples.

5.2 Prediction Thresholds

Analysis of performance on specific test rows revealed that for some particularly vague questions, our linear models failed to predict any tags at all. This is problematic; unlike spam prediction, where precision is critical and false negatives are merely unfortunate, if a tag prediction system fails to suggest at least one relevant tag to a user, this would likely be a bad user experience.

Each tag classifier outputs a probability between 0 to 1 indicating how likely it is that an example belongs to that tag. The standard threshold choice is 0.5. However, since predicting nothing is a more severe error than predicting an incorrect tag, we experimented with alternative thresholds. First, we tried choosing the five tags with the highest probability of belonging to that test row. We then set a restriction that the probabilities output by these tag classifiers needed to be greater than a certain threshold. After experimenting with different thresholds, we found that requiring that all tags have at least a 0.2 probability improved overall recall and F1 Score for our One Vs Rest Logistic Regression (see Figure 3). Curiously, although this improved our Logistic Regression, it did not improve the performance of our Multinomial Bayes model.

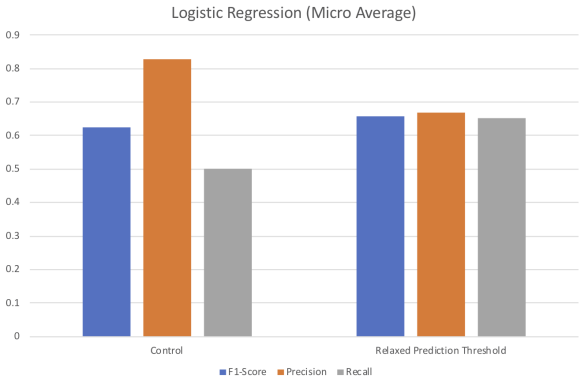

Figure 3: Model performance before and after relaxing classifier confidence thresholds

5.3 Featurizing Title and Question Text Separately

In order to further understand the performance of our classifiers, we sorted the trained tag classifiers by F1-score, and examined their performance on specific test examples. We noted that classification of tags which represented programming languages did not perform as we would have expected, despite good performance associated with tags for APIs and frameworks, such as "facebook," "git,"

	Micro Avg			Macro Avg		
	F1-Score	Precision	Recall	F1-Score	Precision	Recall
Logistic Regression	0.623	0.828	0.500	0.530	0.748	0.428
Logistic Regression (relaxed thresholds)	0.659	0.667	0.650	0.592	0.639	0.565
Multinomial Bayes	0.562	0.553	0.571	0.509	0.501	0.539
Multinomial Bayes (relaxed thresholds)	0.541	0.450	0.678	0.498	0.430	0.620
Linear SVM	0.563	0.866	0.418	0.443	0.715	0.351
Simple Neural Network	0.460	0.780	0.320	0.210	0.450	0.160

Table 1: Model Performance

or "android." After examining some of these failure cases, we realized that there was valuable information contained in titles that was not fully being utilized by the classifiers. Titles are typically shorter than questions, and are a more succinct summary of what the user is trying to convey. In order to take advantage of this, we tried creating features separately from question and title text, and concatenating these into a combined feature vector, which was used in training. This improved the overall performance of our models, and successfully fixed the issues our models were having with prediction of tags which were programming languages.

5.4 Class Imbalance with Tags

Additionally, while analyzing the performance of our tag classifiers, we noted that there were some poorly performing classifiers which were both extremely infrequent, occurring in less than 1% of the training rows of our dataset, and which also represented academic subjects, languages or frameworks which were no longer popular, or very general terms; some examples include, "design," "sdk," "apache," and "algebra." Unfortunately, because of the general nature of these tags, using titles as features did not greatly improve performance here. Therefore, we tried to improve our performance on these tags by instead addressing the sparsity of these tags in the training data. We did this by using class weighting in our modeling. We experimented with using a "balanced" heuristic, in which class weights are given weights inversely proportional to class frequencies in the input data. While this greatly improved recall, it also greatly lowered precision, and analysis of the resulting models revealed that it led our models to bias in favor of blindly predicting tags in order to maximize recall.

Tag	Associated Features
android	asynctask, spinner, apk, edittext, intent, webview, bitmap, adt
django	queryset, tastypie, celery, manytomanyfield, modelform
facebook	fql, fb, graph, wall, likes, friends, timeline, social
git	gitignore, github, commit, branch, commits, pull, egit, control
ruby	rails, nokogiri, ror, rspec, sinatra, activerecord, mongoid

Table 2: Feature Importance for Tag Classifiers with High F1 Score

Therefore, we did not use this technique in our final models.

5.5 Neural Network

Our simple neural network performed reasonably well. We noticed that its recall is much lower than its precision, and that it was too cautious when predicting the presence of a given tag. In order to rectify this, we tried increasing the number of hidden layers, and changing the dimensionality of the hidden layers. However, this was largely unsuccessful in moving the metrics meaningfully.

6 Discussion

6.1 Feature Importance

In order to understand which features helped predict certain tags, we extracted the most important title features to specific tags from our Logistic Regression. This information was obtained from

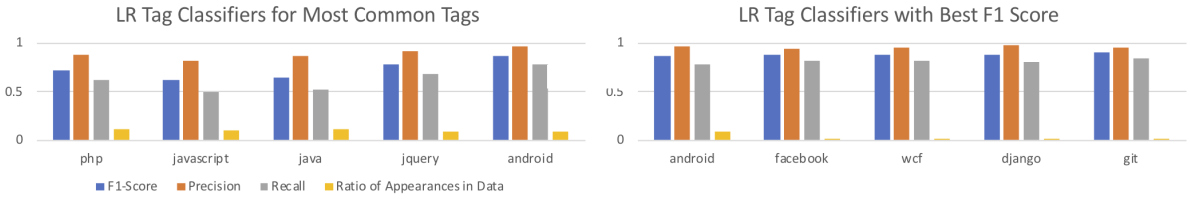

Figure 4: Tag Classifier Performance compared with Tag Prevalence in the Dataset

Tag	Associated Features
code	apex, 404, coverage, qr, entity
touch	sencha, uiimageview, coca nspredicate, uitextfield
query	jquery, jquerymobile, jqgrid jqm, getjson, sortable, accordion
management	cms, autorelease, dalloc, memory, ndb, software, exc_bad_access
sdk	drive, oauthexception, nib, nsmutablearray, uitextfield, fb

Table 3: Feature Importance for Tag Classifiers with Low F1 Score

model weights. Note that model weights cannot always be used in Logistic Regression as a measure of feature importance (for example, one might have a very large weight which corresponds to a feature with a tiny value). However, in this case, we have binary title features, so we can do so.

When examining important features associated with a given tag, several trends stand out. Tools, programming languages, and IDE's are normally predicted by related libraries or tools; for example, "django," which is a python web framework, is associated with "celery" and "tastypie," which are related open source tools. (Examples of feature importance for tag classifiers which perform well can be found in Table 2). While tags that are easy to predict correspond to some very specific keywords, tags which are difficult to predict cover a larger breadth of keywords. For example, "code" is associated with "entity," "404," and "coverage," among other features, but these words are not specific to the tag "code," and may be used in a variety of contexts. Therefore, predicting tags which refer to esoteric or broad topics is likely to be difficult, even with more training rows. (Examples of feature importance for tag classifiers which perform poorly can be found in Table 3).

6.2 Tag Classifier Performance Compared with Tag Prevalence

Because of the sparsity of the tags present in our dataset, we were curious as to whether tag prevalence in the dataset was strongly correlated with the performance of that tags corresponding classifier. In order to investigate this further, we found the tag classifiers which performed the best on the test data by finding the tag classifiers with the best F1-score. We then measured the prevalence of their tags in the dataset. Likewise, we also calculated the performance of tag classifiers corresponding to the tags which occurred most frequently in the data.

What we found was interesting—although we would have expected the most prevalent tags to perform much better than other tags, this was not always the case. As can be seen in Figure 4, it seems like a better criterion for success of a classifier is how unique its tag is, and if there are other words in the vocabulary which uniquely identify it.

7 Conclusion

Overall, our approach worked fairly well. Although our classifiers did not always succeed in predicting every single tag correctly, they were able to successfully predict at least one tag for many examples. If we had more time, we would follow up with improving the Multinomial Bayes classifier, and understanding why it predicted more tags than it should. We would also try to make use of a more complex neural network which could understand sequential data, in order to better model the question and title text in our data.

8 Contributions

Jalal contributed training of the models and failure analysis on misclassified examples. Kevin Fuhs and Reid M. Whitaker both contributed failure analysis on the linear decision boundary models, analysis of the distribution of the data, and a review of the literature.

References

[1] Plotting learning curves. `https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html`.

[2] Project code on github. `https://github.com/greenmintleaf/cs229project`.

[3] Slobodan Beliga, Ana Meštrović, and Sanda Martinčić-Ipšić. Toward selectivity based keyword extraction for croatian news. *arXiv preprint arXiv:1407.4723*, 2014.

[4] Eibe Frank, Gordon W Paynter, Ian H Witten, Carl Gutwin, and Craig G Nevill-Manning. Domain-specific keyphrase extraction. In *16th International joint conference on artificial intelligence (IJCAI 99)*, volume 2, pages 668–673. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[5] TE Gupta. Keyword extraction: a review. *International Journal of Engineering Applied Sciences and Technology*, 2(4):215–220, 2017.

[6] Anette Hulth. Improved automatic keyword extraction given more linguistic knowledge. In *Proceedings of the 2003 conference on Empirical methods in natural language processing*, pages 216–223. Association for Computational Linguistics, 2003.

[7] Marina Litvak and Mark Last. Graph-based keyword extraction for single-document summarization. In *Proceedings of the workshop on Multi-source Multilingual Information Extraction and Summarization*, pages 17–24. Association for Computational Linguistics, 2008.

[8] Rada Mihalcea and Paul Tarau. Textrank: Bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing*, 2004.

[9] Thuy Dung Nguyen and Min-Yen Kan. Keyphrase extraction in scientific publications. In *International conference on Asian digital libraries*, pages 317–326. Springer, 2007.

[10] Ian H Witten and Olena Medelyan. Thesaurus based automatic keyphrase indexing. In *Proceedings of the 6th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL'06)*, pages 296–297. IEEE, 2006.

[11] Chengzhi Zhang. Automatic keyword extraction from documents using conditional random fields. *Journal of Computational Information Systems*, 4(3):1169–1180, 2008.