Keyword Extraction and Semantic Tag Prediction

James Hong
Stanford University
Stanford, CA - 94305
jamesh93@stanford.edu

Michael Fang
Stanford University
Stanford, CA - 94305
mjfang@stanford.edu

Abstract

Content on the web is often organized through user generated tags for intuitive search and retrieval. Such tags convey meta-information about the subject matter of the texts they represent. For this project, we applied machine learning (Bayesian co-occurrence, k-NN, SVM, NNS) to predict tags of StackExchange posts obtained from Kaggle: "Facebook Recruiting - Keyword Extraction III." Using our non-parametric, fuzzy Nearest Neighbor Search algorithm, we achieved a F1-score of 0.471 on a testing set with unseen data and 0.773 on the Kaggle test set (containing many duplicate data points). Furthermore, when predicting a single tag per post, our algorithm attained approximately 71.1% accuracy (on unseen data), surpassing the 0.65 accuracy attained by Stanley & Byrne (2013). Our keyword-tag co-occurrence model and fuzzy NNS proved to be fast and practical for large-scale subject and tag prediction problems with tens-of-thousands of tags and training documents.

1. Introduction

Tagging is one of the simplest ways to organize web content. Allowing users to specify subjects, metadata tags are vital for efficient search and indexing. Many contemporary social media sites support some system of content tagging (Stanley & Byrne, 2013). An automated tagging system to model user generation of tags would therefore be useful for purposes of tag suggestion, document and topic clustering, and tag validation.

2. Data Analysis and Preprocessing

From the Kaggle competition, we obtained a training set with 6,034,195 StackExchange posts, consisting of the title, body, and tags, as well as a testing set consisting of 2,013,337 title and bodies of posts. These posts cover a broad range of topics from 110 StackExchange sites (e.g. StackOverflow, MathOverflow, AskUbuntu). Each post has up to 5 tags, and the number of tags is roughly modeled by a normal distribution with a mean of 2.89 and standard deviation of 0.73. In accordance with Zipf's law, tag frequency drops off dramatically; the top 100 tags constitute 41.1% and the top 1,000 tags constitute 71.5% of all tags.

Through data analysis we observed a high degree of redundancy in the raw datasets. For instance, the raw training set contained only 4,125,233 unique points. In addition, 1,048,575 points in the raw testing set are duplicated from the training set. In conducting our experiment, we elected to train and test our algorithms exclusively on non-duplicate data, since this would more closely resemble real-world conditions and studies by other researchers (Stanley & Byrne, 2013). The data was preprocessed as follows (with an emphasis on speed)[1]:

1. Create a custom lexicon using an English dictionary with stop words removed and terms extracted from the tag set.[2]
2. Apply greedy multi-word matching to capture multi-word terms in the lexicon.
3. Filter out terms not in the lexicon (removing noise such as misspellings or arbitrary variable names in embedded code segments).

[1] We also tried stemming, but it proved ineffective for anything other than space reduction later in the co-occurrence model as most keywords could not be stemmed. POS tagging proved to be too slow on large datasets. Our preprocessing procedure also had the effect of reducing the training dataset from over 7 GB to under 1 GB.
[2] An alternative method of generating a domain specific lexicon for StackOverflow posts would be to use computer science textbook indexes as a corpus.

3. Evaluation Criteria

We measured performance in two ways: the Mean F1-score, and accuracy predicting a single tag (both on non-duplicated test data, using hold-out cross validation for 10,000 labeled posts separated from the non-duplicated training set). F1-score is formulated below, where p is precision and r is recall, tp is true positive frequency, fp is false positive frequency, and fn is false negative frequency.

$$F_1 = \frac{2pr}{p+r}\ ,\ p = \frac{tp}{tp+fp},\ r = \frac{tp}{tp+fn}$$

4. Model Selection

We applied a variety of models to tag prediction. From our initial experiments with naïve Bayes and SVM classifiers for individual tags (e.g. to predict whether c# is a tag), we concluded that such an approach is intractable on large tag sets containing hundreds or thousands of tags. Instead we focused on algorithms that considered thousands of tags using heuristics (co-occurrence) to reduce search space. To measure learning rate, we trained each model on successively larger datasets.

4.1 (Greedy) "Tag in Text" Baseline

In our baseline algorithm, we compiled a list of common tags. For each test example, we simply predicted whichever tags from the list were present in the text of the post. This returned an F1-score of 0.1654.

4.2 Nearest Neighbor Search for Top 500 Tags[3]

Compared to the greedy "tag in text" approach, Nearest Neighbor Search (NNS) on the top 500 tags provided a more realistic baseline for a simple tagging system. The algorithm is as follows:

1. For each tag y in the top 500, concatenate all training examples with tag y. Generate a bag-of-words (BOW) vector with frequency counts, and apply the TF-IDF transform with document frequency computed using the entire training set. (These vectors correspond to "centroids" for each tag.)
2. For each test example, generate a BOW vector and apply the TF-IDF transform. Compute the distance between the test example vector and every tag centroid vector using cosine similarity.
3. Rank the tags by similarity score to obtain predictions.

There are many flaws with this approach. First, the top 500 tags account for only 61.8% of all the tags in all posts in the training set, imposing a strict upper bound on recall. Also, more efficient space partitioning algorithms for NNS break down in high dimensional feature spaces, making NNS computationally expensive (linear search). Consequently, we were unable to efficiently process any training set larger than 100,000 samples using NNS for 500 tags. However, as a baseline, we attained a F1-score of approximately 0.23 and an accuracy of 39.7% when predicting a single tag.

4.3 Tag-Keyword Co-occurrence Model

Using our training examples, we constructed a co-occurrence matrix to model the joint probability of word and tag presence in a post. From this we can approximate upper bounds on the conditional probabilities for each tag, given a set of words in a query. This allows us to quickly identify keywords that are strongly correlated to individual tags.[4]

1. For each tag y, estimate $\max\limits_{word \in query} P(\,y|word)$ using the co-occurrence matrix.
2. Return the top k tags.

[3] In the project milestone write-up, this NNS approach was referred to as "Nearest Tag Distributions".
[4] To reduce space complexity, non-keyword entries in the matrix can be removed. This serves as an extra filter against stop words.

While simple, this algorithm captures approximately 70% of the true tags within the first 10-20 tags predicted. Thus, it serves as a potential intermediate filter for the k-NN and SVM "is tag?" binary classifiers.

4.3.1 SVM and k-NN Extensions to Co-occurrence Model

The co-occurrence model suffered from high bias when predicting less salient tags (those lacking specific keywords). We attempted to remedy this bias by using additional text features based upon the word-tag results from the co-occurrence matrix, and feeding them to SVM and k-NN classifiers. This retains the advantage of working in lower dimensions (instead of working in the space of the lexicon).Then, we applied forward search to determine the most relevant features on each algorithm, which were: the probability bounds given in the above, tag presence in text, and the overall tag probability.

First, we used the k-Nearest Neighbors algorithm with large k (50) to obtain probability estimates based on the proportion of positive neighbors. Next, we experimented on class-weighted and regularized SVM classifiers with linear, polynomial, rbf, and sigmoid kernels, using the functional margin as a measure of confidence for each candidate tag. For the SVM, the best performance was observed with an rbf kernel and positive:negative weights of 7:1. Both SVM and k-NN improved performance on small training sets, where the co-occurrence model was prone to over-fit. However, they were less effective on larger training sets.

4.4 (High Dimensional) Fuzzy Nearest Neighbor Search

The main issue that plagued our 500-tag NNS algorithm was the high dimensionality of the feature space. To address the curse of high dimensionality, we implemented a fuzzy/approximate NNS algorithm for efficient lookup and computation of semantic similarity across all of the 42,048 possible tags observed in the training set. Using the keyword-tag co-occurrence model presented above, we narrowed the search space of possible tags to 10-20 candidate tags, and applied linear NNS in this local search space. Our algorithm:

1. For a test example, use the keyword-tag co-occurrence model to predict 10-20 candidate tags.
2. For each candidate tag, extract its corresponding row in the keyword-tag co-occurrence matrix as a "tag centroid". To increase the salience of the keywords (top non-zero entries) corresponding to each tag, we zeroed out all but the largest 500 components and renormalized the centroids.
3. Compute cosine distance (similarity measure) between the test example (a normalized BOW vector) and each candidate tag-centroid, and re-rank the tags by similarity.
4. Output the tags corresponding to the k-nearest tag centroids.

In subsequent experiments, we adjusted our fuzzy NNS similarity function to penalized candidate tags with low semantic similarity, and reward tags with high similarity. Whereas the keyword-tag co-occurrence model generally performed well on common and specific tags such as PHP, CSS, and C#, which correspond strongly to specific keywords, fuzzy NNS improved performance on more general tags with high semantic similarity (e.g. algorithms, parsing, image). Furthermore, fuzzy NNS improved precision when predicting similar, hierarchical tags (e.g. facebook, facebook-api, and facebook-graph-api). Predicting three tags per query, we achieved 0.481 recall and 0.461 precision on 10,000 unseen test examples when considering all 42,048 possible tags.[5]

5. Results

Table 1. Accuracy Predicting a Single Tag

Model \ # of training examples	100	1000	10k	100k	3000k
Keyword-tag co-occurrence	0.0230	0.1433	0.4042	0.5723	0.6315
co-occurrence & k-NN	0.1814	0.3104	0.5057	0.5971	0.6370
co-occurrence & SVM	0.0255	0.1571	0.4165	0.5595	0.6170
Fuzzy NNS	0.1250	0.2487	0.4662	0.6394	0.7105

[5] When only considering the top 200 tags, fuzzy NNS achieves precision of 0.450, recall of 0.655, and F1 of 0.533.

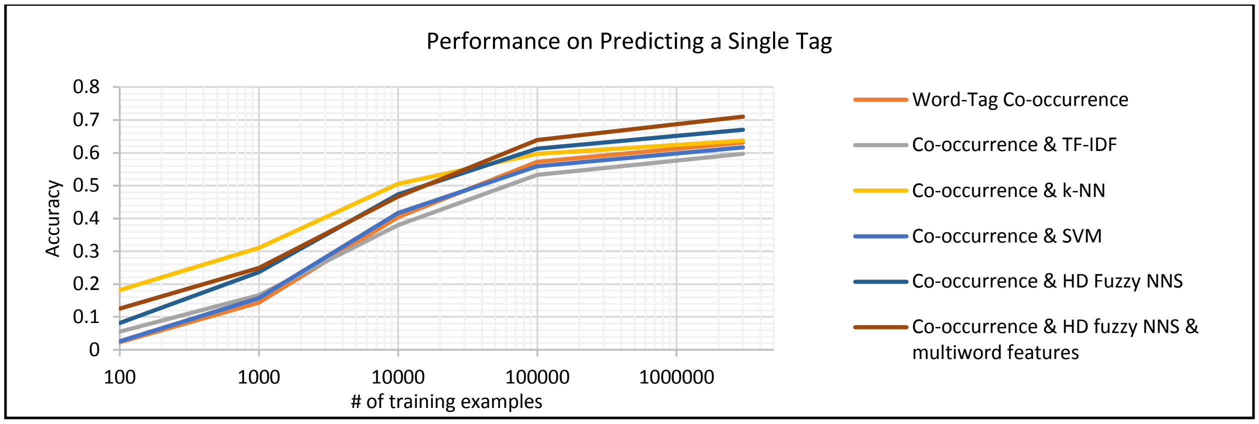

Figure 1. *On the single tag prediction benchmark, our high-dimensional fuzzy-NNS algorithm performed 7.9% better than the keyword-tag co-occurrence model alone when trained on 3,000,000 training examples. On smaller training sets, the k-NN classifier (on intermediate features) with keyword-tag co-occurrence improved test accuracy as the co-occurrence model (even with smoothing) is highly prone to over-fitting on small training sets (<100,000).*

Table 2. F1-Score Performance (Predicting 3 tags per query)

Model \ # of training examples	100	1000	10k	100k	3000k
Keyword-tag co-occurrence	0.0418	0.1123	0.2595	0.3648	0.4100
co-occurrence & k-NN	0.0987	0.1736	0.3129	0.3861	0.4149
co-occurrence & SVM	0.0463	0.1296	0.2818	0.3749	0.4123
Fuzzy NNS	0.0834	0.1564	0.2986	0.4156	0.4704

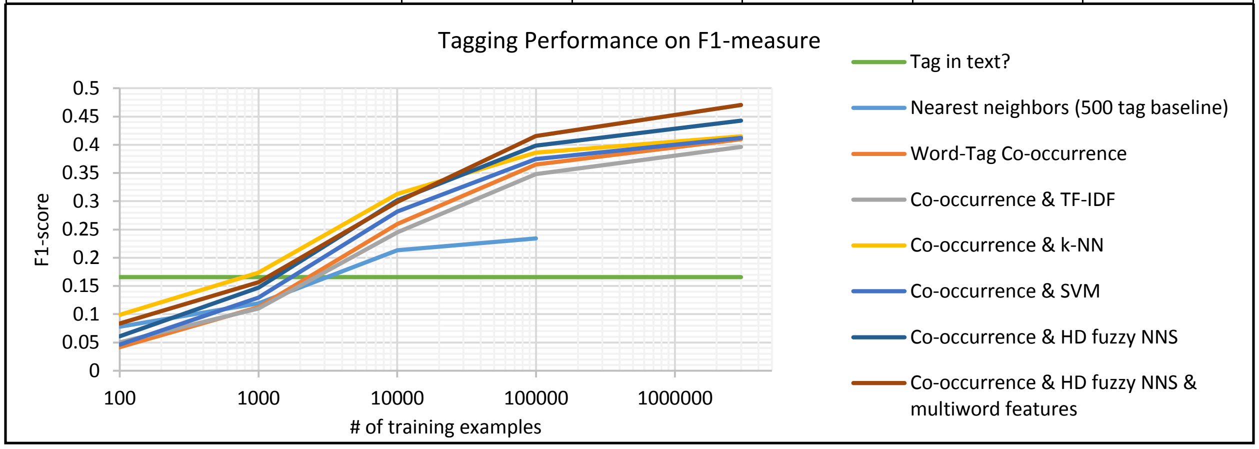

Figure 2. *Similar to the single tag metric, the fuzzy-NNS algorithm attained significantly better F1 performance (+0.0604) over our standalone co-occurrence algorithm and (+0.2364) top-500 tag NNS algorithm.*

Results in Kaggle Competition: Using title string hashing to identify duplicates in the Kaggle test set and then fuzzy NNS to tag the remaining non-duplicates, we achieved a F1-score of 0.77326[6] on the Kaggle competition metric, and ranked 14 of 280 teams at the time of submission and 22/301 as of 12/12/2013.

6. Discussion

The main challenges of the StackExchange tag prediction problem include: (1) the large number of possible tags, (2) the large number of word/multi-word features, (3) the size of the training/testing sets, (4) the difficulty of measuring semantic similarity between domain-specific terms, and (5) high variability in user tagging behavior.

[6] For each test query, we returned exactly 3 tags, regardless of how many tags are in the solution. This is one area that can be optimized for better performance on the F1-metric used by Kaggle.

Our co-occurrence model addresses (1) and (2), allowing us to estimate the probability of a tag. Furthermore, the co-occurrence matrix generated from our 3,000,000 point training set can be easily stored to fit into main memory, allowing for high performance keyword extraction. To reduce the dimensionality of our features and estimate semantic similarity between documents, we originally implemented a LSI model. However, LSI does not scale well to large training sets with over 1 million data points. Instead, tag centroids generated from the keyword-tag co-occurrence were more efficient to compute[7] and allowed us to process the entire Kaggle testing set in reasonable time and overcome the issues of high dimensionality (3) with considerable improvement in precision and recall over the co-occurrence model without fuzzy NNS (4). Finally, (5) users varied on the number and the specificity of tags. For this reason, accuracy predicting only 1 tag and F1-score on all tags are both critical metrics of a tagger's performance, the former testing accuracy on overall subject prediction, and the latter on the viability of the algorithm as a tag recommendation system. Using a regression model to predict the number of tags would probably improve performance on F1 and Kaggle rankings. Finally, compared with the StackExchange prediction study by Stanley and Byrne (2013), which used an ACT-R inspired co-occurrence to achieve 65% accuracy on the single tag metric, fuzzy NNS achieved better accuracy (71%) with similar training set sizes. The difference is most likely the product of re-ranking candidate tags by tag-centroid similarity.

Ultimately, our results demonstrate that a fuzzy NNS algorithm based on tag-keyword co-occurrence and cosine distance can be an effective method for large scale tag and subject classification given enough training data. This relates to similar problems such as hashtag prediction, tag recommendation, and search and indexing. An additional application for tag prediction would be to filter spam/irrelevant tags and posts (Stanley & Byrne, 2013).

7. Future Directions

Here are five possible areas of improvement. In order to further improve precision and recall of the tagging fuzzy NNS algorithm, (i) a more sophisticated similarity function can be used to estimate semantic similarity. For instance, Wikipedia pages or Google search results could be used to generate expanded text representations for less common keywords (Sahami & Heilman, 2006). Additionally, syntactic parsing strategies could improve data preprocessing. (ii) Unsupervised clustering can be applied training examples with common tags, so as to model multiple tag-sub-centroids for more precise similarity comparison. (iii) Applying locality sensitive hashing (MinHash) could allow for tractable NNS against not only tag-centroids but the training examples themselves (Das, Datar, Garg, & Rajaram, 2007). (iv) For practical subject identification systems, modeling tag specificity through tag-tag hierarchies can better balance the trade-off between accuracy and specificity in tag prediction (Deng, Krause, Berg, & Li, 2012). Finally, (v) using distributed computing would allow more sophisticated preprocessing steps, feature extraction, and semantic similarity measures.

8. Bibliography

[1] Das, A., Datar, M., Garg, A., & Rajaram, S. (2007). Google News Personalization: Scalable Online. *WWW 2007* (pp. 271-280). Banff: ACM.

[2] Deng, J., Krause, J., Berg, A. C., & Li, F.-F. (2012). Hedging Your Bets: Optimizing Accuracy-Specificity Trade-offs in Large Scale Visual Recognition. *CVPR*, (pp. 3450-3457). Providence.

[3] Kaggle. (2013). *Facebook Recruiting III - Keyword Extraction.* Retrieved from Kaggle: http://www.kaggle.com/c/facebook-recruiting-iii-keyword-extraction

[4] Kuo, D. (2011). *On Word Prediction Methods.* Berkeley: UCB/EECS.

[5] Lott, B. (2012). *Survey of Keyword Extraction Techniques.*

[6] Manning, C. D., & Schutze, H. (1999). *Foundations of Statistical Natural Language Processing.* Cambridge: MIT Press.

[7] Sahami, M., & Heilman, T. D. (2006). A Web-based Kernel Function for Measuring the Similarity. *WWW 2006* (pp. 377-386). New York: ACM.

[8] Stanley, C., & Byrne, M. D. (2013). Predicting Tags for StackOverflow Posts. *ICCM* (pp. 414-419). Ottawa: ICCM.

[7] Using fuzzy NNS, we were able to predict tags for the entire Kaggle test set (~2 million queries) on a 2012 MacBook Air with a 1.8 GHz DC ULV processor and 8GB of RAM in the span of 5 hours.