Identifying Tags from millions of text question

Chintan Parikh, chintanp@stanford.edu

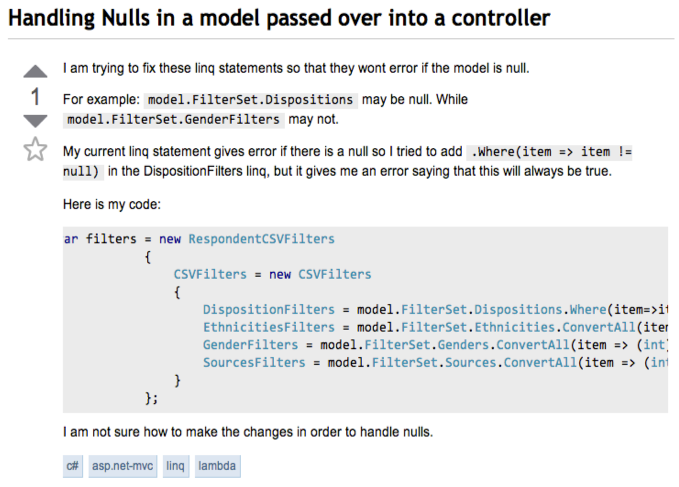
Abstract—Identifying tags or keywords from text has been a very important class of application of text data mining. In the case of Questions and Answer sites such as Stack overflow or Quora tagging allows users to explore more related content, build and showcase expertise in a given area and in general get more visibility to the question at hand. In this paper I take on the problem of identifying tags for the questions asked at Stack exchange sites based on title and text of the question. For this problem Vowpal Wabbit is used as a tool to build set of discriminative classifiers for each of the tags in the training set. The resulting tags for each of test questions are predicted using running through each of the classifiers.

Index Terms—Machine Learning, Clustering, Keyword extraction, Text Analysis

I. Background and Motivation

Tagging has become popular way to categorize text and non-text information. With the advent of twitter hashtags, people have started promoting usage of tags in order to categorize and find related content easily. Stack overflow is a popular site for discussing programming related questions, and now they have dataset of over 6 million questions. On stack overflow a user can tag each questions up to five tags to categorize a question, using existing tags or create a new tag in certain cases. Although they restrict ability to create a new tags by having requirement on certain reputation.

In this problem, I look at dataset obtained from Kaggle competition [1] which contains questions and related tags from Stack Exchange sites in the training set. The training dataset has over 6 million questions with associated 42,049 unique tags. Each question has average of 2.9 tags associated with it. Figure 1 shows one example of training set with its associated tags.



Handling Nulls in a model passed over into a controller

1

I am trying to fix these linq statements so that they wont error if the model is null.

For example: `model.FilterSet.Dispositions` may be null. While `model.FilterSet.GenderFilters` may not.

My current linq statement gives error if there is a null so I tried to add `.Where(item => item != null)` in the DispositionFilters linq, but it gives me an error saying that this will always be true.

Here is my code:

```
ar filters = new RespondentCSVFilters
           {
               CSVFilters = new CSVFilters
               {
                   DispositionFilters = model.FilterSet.Dispositions.Where(item=>it
                   EthnicitiesFilters = model.FilterSet.Ethnicities.ConvertAll(iter
                   GenderFilters = model.FilterSet.Genders.ConvertAll(item => (int)
                   SourcesFilters = model.FilterSet.Sources.ConvertAll(item => (int
               }
           };
```

I am not sure how to make the changes in order to handle nulls.

c# asp.net-mvc linq lambda

Figure 1:Question with tags (c#, asp.net-mvc, linq, lambda)

The problem statement is to predict the tags for the test set of over 2 million questions using only the model learned from the training set.

Top ten tags sorted by their frequency are shown in table 1. The top 10 tags account for 17.26% of all the tags. Similarly when analyzing top 100 tags they account for around 40% total tags. This follows perfect power law distribution as shown in figure 2.

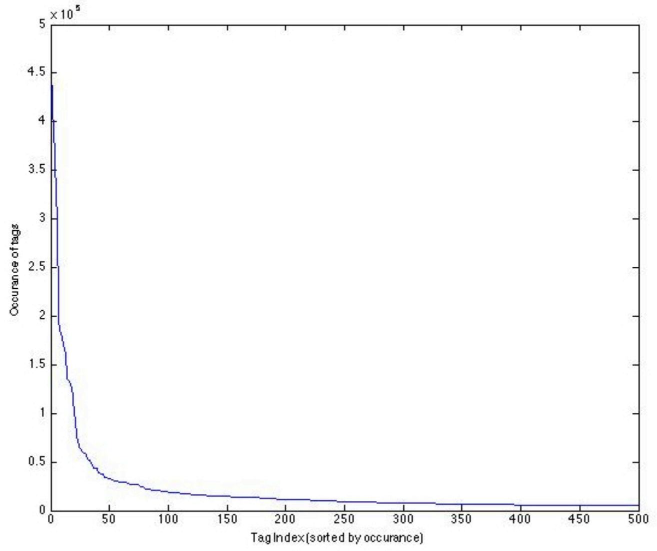


Figure 2: Tag distribution

Tags	*Times occurrence*	*Percentage occurrence*
c#	463,526	2.66
java	412,189	2.36
php	392,451	2.25
javascript	365,623	2.1
android	320,622	1.84
jquery	305,614	1.75
c++	199,280	1.14
python	184,928	1.06
iPhone	183,573	1.05
asp.net	177,334	1.02

Table 1: Top ten tags

II. Approach

This section details the approach tried out for predicting the tags for each of the question. In this discussion I take Top500 tags into account as they account for 61.8% of all the tags and also it speeds up the process of testing multiple hypothesis.

A. Setting up the baseline:

Kaggle has supplied with multiple baselines for comparing the results, one with mean F1 score of 0.07 where it predicts top 5 tags (c#, java, python, php and javascript) for all the tags. For setting up another naïve baseline, first the data set was cleaned removing XML tags, punctuation text and common stop words in English language using Natural Language Toolkit [3]. This was tokenized and feed through a simple classifier, which searches the post for the Tag keywords and if they are present then predicts that tag. This is then aggregated and then finally selects the top three tags for each of the question. This approach results in mean F1 score of 0.19.

B. Building discriminative classifiers

Two approaches were identified for tag prediction for the questions, one was to build a global multiclass classifier and then use method such as One vs All to select the final class. The second approach was to build a discriminant classifier for each of the tags and then predict the final tags choosing the most likely tags. Since we need to predict more than one tags for each of the question, it was decided to use the second approach. The block diagram below explains the approach.

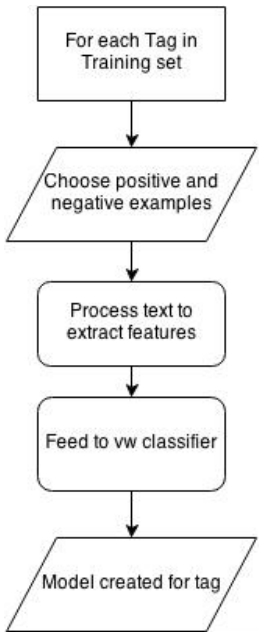

For each Tag in Training set
Choose positive and negative examples
Process text to extract features
Feed to vw classifier
Model created for tag

Figure X: Training of classifier for each tag

To build a discriminative classifier Vowpal Wabbit (vw) [4] was selected. It provides several loss functions as well as learning algorithms. vw provides a sparse matrix input format which easily allows a bag of words models. Also vw hashes the features names to 2^18-1 space, there in reducing the dimension and allows faster lookups.

Feature Selection:

The feature selection was done using vw wrapper called as vw-varinfo, which exposes all variables of models in a human readable form. The output includes the input variable names, including name-spaces where applicable, the vw hash value, the range [min, max] of the variable values in the training-set, the final model (regressor) weight, and the relative distance of each variable from the best constant prediction.

Using this, we learn the relative importance of the words and we can remove the features with 0% relative importance, so as to reduce model size. Table below shows the feature vectors for the model for c#.

Feature	*Rel score*	*Feature*	*Rel score*
c#	100.00%	qt	-80.51%
initquestion	100.00%	autoslideinterval	-76.03%
winform	54.71%	andriod	-76.03%
copypathlist	51.33%	printf	-72.89%
xmldocument	46.29%	xcode	-70.12%
addin	43.13%	rails	-67.70%
button_neutral	43.04%	wikiversity	-65.32%
csharp	40.42%	gcc	-65.32%
enumerable	39.48%	boost	-62.96%
containskey	38.08%	java	-61.70%

Table 2: Relative importance of features for C# classifier

Choosing loss function:

For the classification task, vw has support for two loss functions namely ‘hinge’ (SVM) and ‘logistic’. The parameters The default values of regularization and learning rate of 0.1 gives the best result for SVM classifier. Figure 3 and 4, compares the performance on SVM and logistic classifiers, on the metric of Precision and Recall for the classifier built for Top 100 tags.

Choosing input samples for building models

The SVM classifiers are sensitive to the ratio of positive (M) training documents and negative (K) training documents. A previous study [5] suggests that a discriminative model produces the result for a class that has 60 positive and 1500 negative examples. In the initial phase of training, this fixed value was being used to create training set for each of the classifiers. But the total sample size of 2100 was not sufficient to capture and it has high variance.

The approach which was implemented in the final classifier, which gave the maximum precision and recall is discussed below,

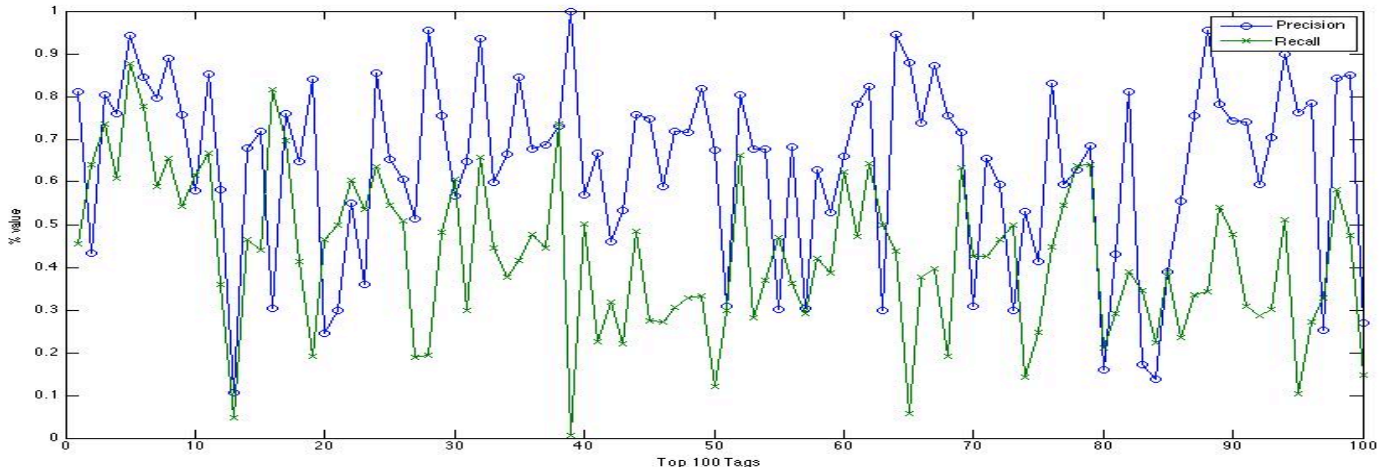


Figure 3: Logistic loss function, mean P = 0.64, R = 0.42

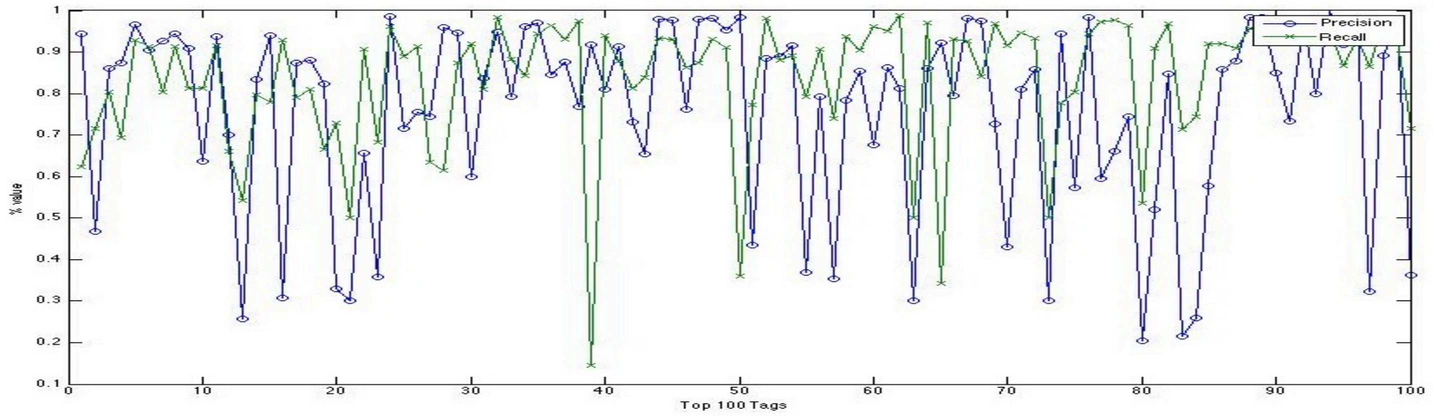


Figure 4: Hinge loss function, mean P = 0.76, R = 0.72

For each of the tags:
- Get the occurrence value (O).
- Choose M to be O/2; K to be 25*M
- Each M and K are chosen from the question which has more than 800 characters so as to have enough information in the model.

C. Tag Suggestion

Once we have the models built for each of the Top500 tags, next step is to predict multiple tags for each of the question. For this we use the algorithm as below,

----

Step 1: For each question in the test set:
- Run through all the classifiers in Top500 set
- Add the SVM output of each classifier to a list

Step 2: Now sort the list to get the maximum 5 output for particular question, and assign these tags to the question.

The output of vw classifier is between -1 and 1, we choose the top 5 values which are above a fixed threshold (set to 0.1) in the list as our final tag output.

III. Results

From the above section it is clear that hinge loss function performs much better than the logistic loss function, hence it was used in the final classifier.

A. Evaluation metric:

Mean F1 score is used as evaluation metric, which measures accuracy using statistics precision p and recall r. Precision is the ratio of true positive (tp) to all predicted positive (tp+ fp). Recall is ratio of true positive (tp) to all actual positives (tp + fn).

$F1 = 2pr / p + r$

So in order to maximize the F1 score, the algorithm should maximize both recall and precision simultaneously.

B. Some results from the tag suggestion

Original Tags	*Suggested Tags*
Php, image-proecessing, file-upload, upload, mime-types	Image, file, php
Firefox	Firefox, windows
R, matlab, machine-learning	Ubuntu, apache, networking
C#, url, encoding	C#, string, json
Php, api, file-get-contents	Php, api, file
Core-plot	Ios, iphone
C#, asp.net, windows-phone-7	Windows, asp.net, c#
.net, javascript, code-generatio	Javascript, c#, linq
Visual-studio, makefile, gnu	Visual-studio, file
Html, semantic, line-breaks	Html

Table 3: Original and suggested tags for questions from test set

Note in the above results, the tags are predicted from the Top500 tags classifier set.

We can set that the top tags such as c#, php are being predicted with a very high accuracy, where as lower occurring tags which are not part of the Top500 tags are missed or being predicted with some synonym from the Top500 set. The example for that being makefile -> file, windows-phone-7 -> windows.

C. Classifiers performance

From the figure 4 we see that we can build a fairly accurate classifier with a mean Precision of 0.76 and Recall of 0.72. This value is obtained from Test set of 100,000 samples which were not part of the training set.

The tags for which the precision was lowest in the Top100 set were; file, windows, forms, list, api, oop, class. The precision and recall for them is shown in the table below.

Tag	*Precision*	*Recall*
File	0.2044	0.5350
Windows	0.3284	0.7286
Forms	0.3526	0.7404
List	0.2594	0.7447
Oop	0.3620	0.7159
Api	0.2142	0.7138

Table 4: Tags with lowest precision values in Top100

The classifiers in the above list have a noticeably low precision and higher recall values. This could mean that the algorithm is bit too liberal in making the classification leading to lower precision values. This could be true for the tags, which occur generically in the context of multiple tags. For example, api, list, file. This could have multiple connotations and don't particularly belong to a particular language or a tag set.

D. Suggested tags performance

To measure the performance of the entire algorithm to predict the suggested tags, we run through the test set of 100,000 samples through Top500 classifier set. Following is the result from the run

TP = 53219	P = 0.6439
FP = 29426	R = 0.2551
FN = 206221	F1 = 0.3648

Table 5: Final result of F1score

The low recall is some what expected, as we are not classifying from the entire set of 43k tags.

E. Kaggle Submission:

The algorithm described in the above section was submitted to the Kaggle, where the test set contains over 2 million test question. The competition is particularly intense, as Facebook is conducting it for recruiting. The competition ends on 12/20/2013 and as of 12/12 the algorithm described above had the standing of 74th out of 310 total teams.

The mean F1 score of the submission using the methods describe above was 0.71132 compared to the top of 0.81539. Note the high values of F1 score compared to above result are largely due to overlap of Test set in the Training data set. Thus for the questions in Test set if they belonged in Training set, then the same tags were predicted for them.

IV. Conclusion

Vowpal Wabbit was used extensively in the development of classifiers and its sparse input format, hashing trick and particularly vw-varinfo wrapper had been very useful to debug the models and come up with valid features. The hinge loss function works much better than logistic loss function.

As discussed in the earlier sections, it is possible to build highly accurate classifier for each of the tags in the Training set. The precision is higher for specific tags such as php, python and it decreases for generic tags such as file, java etc. The results show that average precision of 0.76 is obtained for the tags in Top500 set. The recall is particularly low in the results, since we are not predicting tags from the entire tag set.

V. Future work

The next immediate thing to try out it to build a set of Top2000, Top10000 and all the tags and see how the

Precision and Recall values vary. The expectation is that mean F1 score should go up by few percentage points.

The other thing to try out could be to add more features so as to improve accuracy of the existing Top500 tags. Also could look at techniques such as LDA to give us a list of topics for documents, which could be, then used a feature.

References

[1] Kaggle competition Facebook, Keyword Extraction http://kaggle.com/c/facebook-recruiting-iii-keyword-extraction

[2] Kaggle leaderboard, http://www.kaggle.com/c/facebook-recruiting-iii-keyword-extraction/leaderboard

[3] Bird, Steven, Edward Loper and Ewan Klein (2009), Natural Language Processing with Python. O'Reilly Media Inc.

[4] J. Langford, L. Li, and A. Strehl. Vowpal wabbit online learning project, http://hunch.net/?p=309, 2007.

[5] Wang Jian, Davidson Brian, Explorations in tag suggestion and query expansion," in Proceedings of the 2008 ACM workshop on Search in social media, ser. SSM '08. New York, NY, USA: ACM, 2008, pp.

[6] Saha A, Saha R, Schineider K, A discriminative model approach for suggesting tags