Stack Exchange Tag Prediction through Keyword Extraction

Gio Borje
University of California, Irvine
gborje@uci.edu

Paul Kang
University of California, Irvine
pdkang@uci.edu

ABSTRACT

StackExchange is a set some of the most renown, community-driven question and answer sites. The annotation of questions with tags enables users to find and respond to questions of interest immediately. However, the annotation of questions with tags is tedious and error-prone because users may not be aware of the best tags to categorize their question. By automating the process of annotating questions with tags, the community is relieved of their crowdsourcing work and enables them to focus on the important aspect of asking and answering questions. We survey several approach to keyword extraction and tag suggestion to StackExchange posts using a Bag-of-Words (BOW) model.

1. INTRODUCTION

Kaggle is a community of data scientists focused on solving complex data-science problems by providing company-hosted, public, data science competitions. Competitions frequently proceed as follows. A company begins hosting a competition by providing a public data set as well as an evaluation metric. Participants register for the competition, download the data set, manipulate the data, and creates an appropriate model. Then, the participants download the test data set and make predictions on the test data using the trained models. The predictions for the test data are uploaded into the competition page, scored by the specified evaluation metric, and ranked accordingly on a public leaderboard.

The challenge is proposed through Kaggle: given only the title and body of a question, predict the question's tags. The training data set provided comes from Stack Exchange sites which mixes both technical and nontechnical questions.

The evaluation metric for this competition uses the Mean F_1-Score which is commonly used to measure classification accuracy through *precision* and *recall*. Precision can be defined as the ratio of true positive classifications over the true and false positive classifications. The recall can be defined as the raito of the true positive classifications over the true posities and false negative classifications. The Mean F_1-Score formula is then

$$F_1 = 2\frac{pr}{p+r}$$

where p and r are precision and recall respectively. The Mean F_1-Score is maximized through good precision and recall.

1.1 Data Set Analysis

First, we begin with a preliminary analysis of the StackOverflow data set. The training data contains three attributes: question ID, body and tags. The testing data contains two attributes: question ID and body while the tags are to be predicted. The body of each question contains raw HTML which also separates the title and body of each question.

Since the challenge fixes the number of tags for a question to be in between 1 and 5, we can utilize the distribution of the training data to suggest a particular number of tags. The distribution of the number of tags per questions can be seen in Table 1.1. It is necessary to realize that

Number of Tags	Questions	Percentage
1	492518	13.76
2	951594	26.65
3	1023028	28.65
4	685565	19.17
5	420319	11.77

Table 1: Distribution of the number of tags per question

the distribution is similar to a normal distribution. A useful descriptive statistic is the average number of tags per question which has been observed to be 2.86.

2. METHODS

2.1 Tools

The solution is programmed using Python. The implementations of the models, cross validation and metrics are provided in the scikit-learn library which is coupled with scikit and numpy.

2.2 Problem Decomposition

We begin by decomposing the problem into a set of binary classifications tasks: for each tag, generate a classifier that predicts whether a given question should have the tag with a degree of certainty. Each tag must therefore have its own training set. We design the training set for each tag such that there are exactly 60 questions with the tag and 1500 questions without the tag.

2.3 Model Selection

In order to select the best model for each tag, several models are assessed using cross-validation such that the model with the best mean score is used as the estimator for the tag. In particular, we use stratified K-Fold cross validation that partitions the training data to k folds such that the expected value of the predictions are approximately similar across each fold. We select k to be three folds since 60 and 1500 are equally divisible by three.

Each model assessed uses grid search for hyperparameter optimization. That is, the parameters of each model are tuned exhaustively with bounds. Once all parameter combinations have been assessed, the parameters with the best mean cross validation score are used in their model.

2.4 Preprocessing

Since we are using a Bag-of-Words (BOW) approach, we must vectorize the data set. That is, for each question, we use a count vectorizer which generates a histogram for each of the words in a question. The vectorized form is then used as the feature set for training models and making predictions.

A vectorizer's transformation of the data proceeds as follows. First, given the corpus of questions, it is tokenized and each token is used as a feature. Then, a histogram of the tokens are computed per question and stored as rows of a matrix. Hence, this matrix should have dimensions $N \times M$ where N is the number of questions and M is the number of tokens extracted. We call this matrix, the *question-token matrix*. We can visualize this matrix as seen on Figure 1.

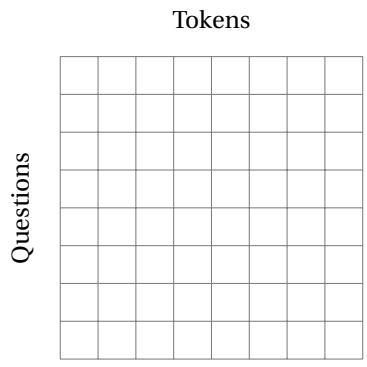

Figure 1: Question-Token Matrix

Once the question-token matrix has been compputed, an additional preprocessing step follows: latent semantic analysis. That is, we apply Singular Value Decomposition (SVD) on the matrix to reduce the dimensionality. We specify the reduced dimensionality to be 512 for rapid prototyping.

2.5 Bernoulli Naive Bayes

2.6 Random Forests

The Random Forest approach is an ensemble method that uses a set of weak learners to to form a strong learner. The weak learners are frequently decision trees which are grouped together into a forest. The number of decision trees generated is parameterized by T. Each split of a decision tree is computed starting with a random subset of the features of size m and selecting the best of those features.

The prediction process for classification in Random Forests is as follows. First, the feature vector is input into each of the trees of the forest. Then, the prediction of the classifier should be the class that is the voting majority of the trees in the forest.

In our solution, we hyperoptimize $m \in \{\sqrt{N}, \log_2 N\}$ where N is the number of features and we hyperoptimize T.

2.7 Linear Support Vector Classifier

The idea of a Support Vector Machine is to define a separating hyperplane which distinguishes a set of classes, which, in this case, is a boolean value of whether or not a tag should be accepted. There are, however, many feasible separating hyperplanes. Hence, we define the optimal hyplerplane to be the hyperplane that maximizes the *margin*. The margin is defined to be the largest minimum distance of the hyperplane to each of the training samples.

We can compute the optimal hyperplane by reformulating the problem as an optimization problem. The primal form of the optimization problem can be described as follows:

$$\underset{\mathbf{w},b}{\operatorname{argmin}} \frac{1}{2}\|\mathbf{w}\|^2$$

where $\mathbf{w}$ and b define the optimal hyperplane.

Since, in our data set, it is rarely the case that the data is linearly separable, the optimal hyperplane can be redefined using the *Soft Margin* method which

$$\underset{\mathbf{w},b,\xi}{\operatorname{argmin}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i$$

introduces non-negative slack variables, ξ_i, and a penalty parameter, C. In our solution, we hyperoptimize $C \in \{10^i | 0 \leq i \leq 9\}$ using grid search.

2.8 Gradient Boosting Machines

Gradient Boosting Machines (GBM) is an ensemble method on weak predictors that builds its model in a stage-wise fashion where each stage minimizes a specified loss function using the Gradient Descent method. Each weak predictor is described as a function $h(\mathbf{X};\mathbf{p}_i)$ where $\mathbf{X}$ is the in-

put and $\mathbf{p}_i$ is its parameter set. A loss function, $L(\mathbf{y}, \mathbf{y}')$ is then specified on $\mathbf{y}$ and $\mathbf{y}'$ which are the training target and predicted response respectively. The predicted response is computed as a linear combination of the weak learners:

$$\mathbf{y}' = \sum_{i=1}^{K} \beta_i h(\mathbf{X}; \mathbf{p}_i)$$

where K is the number of weak learners and β_i is a parameter for a specified weak learned, i.

Hence, the model is described by β and $\mathbf{P}$ which are the weak learner weights and parameter sets respectively. We can then formulate the problem as an optimization problem:

$$\underset{\beta, \mathbf{P}}{\operatorname{argmin}} L(\mathbf{y}, \sum_{i=1}^{K} \beta_i h(\mathbf{X}; \mathbf{p}_i))$$

In our solution, we hyperoptimize $K \in \{100, 200, 300\}$, max_features $\in \left\{\sqrt{M}, \log_2 M\right\}$ and min_samples_split $\in \{1, 2, 3\}$.

3. RESULTS

The Mean F_1-Score for the each of the models on the specified subset of tags are shown on Table 3.

For results with the Latent Semantic Analysis dimensionality reduction step, there are two classifiers that have been significantly impacted. First, the Random Forest approach has a noticeable decrease in its score. Second, the Naive Bayes approach. In some cases, Naive Bayes even performs better than the Linear Support Vector Classifier. The Support Vector Classifier, on the other hand, does slightly poorly.

4. DISCUSSION

5. CONCLUSION

Tag	Bernoulli Naive Bayes	Linear Support Vector Classifier	Random Forests	Gradient Boosting
codeigniter	0.1764	0.7202	0.5176	0.6122
spring	0.2957	0.7268	0.4874	0.6456
sqlalchemy	0.3516	0.8959	0.7742	0.8543
oauth	0.4117	0.8238	0.4157	0.6899
mysql	0.1256	0.3877	0.0240	0.2006
oracle	0.1817	0.6398	0.2893	0.5034
postgresql	0.1957	0.6299	0.5194	0.3807
sqlite	0.1731	0.7293	0.4919	0.5642
ubuntu	0.2694	0.5701	0.2040	0.4390
debian	0.3079	0.7239	0.5674	0.7042
centos	0.2353	0.5878	0.1641	0.3173
osx	0.1283	0.4061	0.1635	0.3469
windows-7	0.2498	0.4527	0.0465	0.2070
python	0.0921	0.3181	0.0951	0.1563
java	0.0727	0.1725	0.0081	0.0662
c++	0.1291	0.2110	0.0082	0.0830
c	0.1314	0.2962	0.0313	0.1168
ruby	0.0978	0.3972	0.1328	0.2048
haskell	0.2524	0.7708	0.7378	0.6077

Table 2: Mean F_1-Scores of Models with Count Vectorized Feature Vectors

Tag	Bernoulli Naive Bayes	Linear Support Vector Classifier	Random Forests	Gradient Boosting
codeigniter	0.5405	0.6683	0.0799	0.4278
spring	0.6817	0.7770	0.3708	0.6045
sqlalchemy	0.7551	0.9038	0.5051	0.7662
oauth	0.6719	0.8200	0.3958	0.6820
mysql	0.4733	0.3961	0.1069	0.3550
oracle	0.5908	0.6530	0.3187	0.5542
postgresql	0.6137	0.6495	0.1870	0.4694
sqlite	0.6090	0.7090	0.3813	0.6321
ubuntu	0.5917	0.5702	0.2821	0.5273
debian	0.6987	0.7426	0.3100	0.6028
centos	0.5644	0.5653	0.2238	0.4831
osx	0.4699	0.4023	0.1565	0.3290
windows-7	0.4685	0.4535	0.1844	0.4149
python	0.4078	0.3731	0.0927	0.2826
java	0.1853	0.1601	0.0322	0.0995
c++	0.1801	0.2352	0.0082	0.0434
c	0.2446	0.1727	0.0164	0.0801
ruby	0.4512	0.3877	0.1316	0.3022
haskell	0.6769	0.7351	0.2941	0.6033

Table 3: Mean F_1-Scores of Models with Latent Semantic Analysis