Mining StackOverflow to Filter out Off-topic IRC Discussion

Shaiful Alam Chowdhury, Abram Hindle
Department of Computing Science
University of Alberta, Edmonton, Canada
Email: {shaiful, abram.hindle}@ualberta.ca

***Abstract*—Internet Relay Chat (IRC) is a commonly used tool by OpenSource developers. Developers use IRC channels to discuss programming related problems, but much of the discussion is irrelevant and off-topic. Essentially if we treat IRC discussions like email messages, and apply spam filtering, we can try to filter out the spam (the off-topic discussions) from the ham (the programming discussions). Yet we need labelled data that unfortunately takes time to curate.**

To avoid costly curration in order to filter out off-topic discussions, we need positive and negative data-sources. On-line discussion forums, such as StackOverflow, are very effective for solving programming problems. By engaging in open-data, StackOverflow data becomes a powerful source of labelled text regarding programming. This work shows that we can train classifiers using StackOverflow posts as positive examples of on-topic programming discussion. YouTube video comments, notorious for their lack of quality, serve as training set of off-topic discussion. By exploiting these datasets, accurate classifiers can be built, tested and evaluated that require very little effort for end-users to deploy and exploit.

I. Introduction

The emergence of social networks and other web-based communication systems have made on-line discussion groups and forums a reality. Programmers use a variety of discussion platforms with different ranges of time constraints combined with synchronous and asynchronous behaviour. StackOverflow (SO) is an example of asynchronous long lived discussion medium: questions and answers are created at different times and the discussion is archived. Internet Relay Chat (IRC) is synchronous and short-lived. Only those in the chat channel, at the time, see the discussion and the discussion occurs mostly at the same time.

IRC is a valuable resource for programmers because it can enable developers to interact in near-real-time with other developers. One can contact a project's developers and seek clarification and help. Immediate help available from OpenSource experts makes IRC quite attractive to programmers [9].[1]

As such there are many general-purpose programming IRC channels such as #python, #java, and #haskell. The success of IRC programming channels is so pervasive that even the official Python documentation site recommends Python programmers to use different IRC Python channels.[2]

IRC channels are built around channel names, the communities, and the topics of the channels. Sometimes users, in their own sense of community, post public off-topic messages to the IRC channels. This can hinder the effectiveness and popularity of a particular channel.[3] Since active administration of a channel takes human effort, the idea of an off-topic filter came up: what if we could improve programming IRC channels for the users by simply highlighting those messages that were on-topic.

Our *contribution* is leveraging StackOverflow data to produce a filter that can hide off-topic discussions in IRC channels and similar instant messaging and channel-based chat products. We have implemented machine learning techniques to filter out off-topic discussions from on-line programming communities, using the Python IRC channel as a test case.

The problem with using a classifier to filter out IRC off-topic discussion is the absence of labelled IRC data.

Making labels from manual annotation of IRC discussions is not a feasible approach; a very large number of labelled examples are necessary for building an accurate text classification model [8]. This led us to design our training phase by exploiting on-line community discussions from other sources rather than IRC. We used Python related questions/answers from StackOverflow as our positive examples (i.e., programming discussions), and YouTube video comments as our negative examples (i.e., off-topic discussions).

The contributions of this paper are two fold: 1) We show successful machine learning approaches for filtering out Python IRC off-topic discussions along with a prototype implementation of our proposed model; 2) We show that similar data from different sources can be used for text classification, which is helpful in case of scarcity of labelled training examples.

II. Related Work

Text classification has been considered as one of the classical information retrieval problems. By combining labelled examples with classification algorithms such as Neural Networks, Support Vector Machines (SVM), and Naive-Bayes, accurate classifiers of unseen examples have been produced [1]. Labelled text data is sometimes very scarce, thus methods to work with partially labelled data have been proposed [8].

[1] Fizer Khan, Every Programmer should use IRC, http://www.fizerkhan.com/blog/posts/Every-Programmer-should-use-IRC.html last accessed: 22-Feb-2015

[2] https://www.python.org/community/irc/ last accessed: 22-Feb-2015

[3] https://news.ycombinator.com/item?id=5587268 last accessed: 22-Feb-2015

978-0-7695-5594-2/15 $31.00 © 2015 IEEE
DOI 10.1109/MSR.2015.54

In cases, where labels for both classes are missing, or very limited, semi-supervised or unsupervised learning algorithms, such as clustering, have been used [1]. Shahib *et al.* [9] were the first to investigate mining IRC logs and IRC log abstraction techniques. Squire *et al.* [10] studied profanities in OpenSource related IRC channels by text filtering. Classifying IRC data is, however, even more difficult as large amount of data is not publicly available and many of the messages are short; it can be impractical and privacy invasive to collect a large number of examples—thus leading us to follow an unconventional machine learning approach: using data from different sources for training.

III. Methodology

Our proposed system consists of five different phases: 1) Retrieve SO questions/answers tagged with Python (positive examples); 2) Retrieve YouTube video comments (negative examples); 3) Train and cross validate the models; 4) Test model performance; 5) Employ trained model on IRC chat.

A. Data Collection

StackOverflow (SO) (http://stackoverflow.com), with more than three million registered users, is perhaps the most popular programming Q&A site. We collected 200,000 questions and answers on Python randomly from SO [12][11], and used them as positive examples for training; the associated tags of a post (e.g., html) were removed and the combined *text* and *code* parts were used. StackOverflow questions and answers are moderated and thus some semblance of quality is enforced; the same might not be true for user comments as they do not generate reputation. On the contrary, we considered video comments on YouTube (http://youtube.com)—the most popular user-generated video site—as a source of learning off-topic discussions. We have selected only News & Politics, Sports, Games, Movies, Music, Entertainment and Comedy videos for data collection, as our personal judgment suggests that these are the most common off-topic topics of discussion in IRC programming channels. We collected 200,000 comments from approximately 800 popular YouTube videos using Python YouTube API.[4] Finally, using a small number of publicly available discussions in IRC python channel,[5] we manually annotated two different datasets: 1) *Gold_Cross_Set* with 50 positive and 50 negative examples for cross validation phase and 2) *Gold_Test_Set* with 150 positive and 150 negative examples for testing the accuracies of our models.

B. Data Pre-processing

We processed all the collected texts using Python NLTK [2]. In order to represent each post as a bag of words, we used *WordPunctTokenizer* to separate words by whitespace and punctuation, and *SnowballStemmer* to transform inflected words to their root forms. English stop words were removed to reduce the feature space size. We retrieved a total of 1,414,765 words with 76,512 unique words from 200,000 YouTube comments and a total of 15,028,825 words with 250,179 unique words from 200,000 SO Python posts. Finally, Each message from IRC, StackOverflow, and YouTube was converted to a feature vector—a vector representing the number of occurrences of every words in a text. The best performance was achieved when bags of words were formed by using both text and code from SO.

[4] https://developers.google.com/youtube/1.0/developers_guide_python

[5] http://www.irclog.org/freenode/python.html last accessed: 23-Feb-2015

C. Classification Algorithms

In this paper, we compare the performance of two classification algorithms in classifying IRC data: *Multinomial Naive-Bayes* classifier (MNB)—a simple generative classifier widely used for text classification; and *Support Vector Machines* (SVM)—a powerful discriminative classifier with the ability to generalize in the presence of large number of features.

1) Multinomial NaiveBayes: The MNB classifier models a document as a bag of un-ordered words. We represent our objective function in MNB as follows: The predicted class of a document (positive or negative) is obtained from equation 1; $P(c_i)$ is the prior probability of class i. $P(w|c_i)$ is the probability of the word w given the document class is c_i, as presented in equation 2 with Laplace smoothing to deal with unseen words in test dataset—$|V|$ is the number of unique words after combining both the positive and negative documents; N^{c_i} is the total number of words in class c_i; and $n_w^{c_i}$ is the frequency of the word w in class c_i. The Laplace smoothing can lead to an inaccurate estimation when the numbers of words differ significantly between the two classes. For example, a word with the same relative frequency in both classes should have no discriminative power, but this is no longer true after applying Laplace smoothing. This is why a normalized count of words is used (equation 3)—thus ensuring no distortion to a word's discriminative power, which holds for any given value of α [4].

$$C = \underset{c_i \in \{+,-\}}{\arg\max} \{\log P(c_i) + \sum_{w \in doc} \log P(w|c_i)\} \quad (1)$$

$$P(w|c_i) = \frac{1 + n_w^{c_i}}{|V| + N^{c_i}} \quad (2)$$

$$nr_w^{c_i} = \alpha * \frac{n_w^{c_i}}{N^{c_i}} \quad (3)$$

Another important aspect of text classification is feature selection—identifying only the words with significant discriminative abilities in order to avoid over-fitting [5]. One of the commonly used techniques to select important words is to calculate the gini-index [1], as presented in equation 4; $p_c(w)$ is the probability of a class c document given the document contains the word w; P_c is the prior probability of class c and K is the number of classes. A word with a gini-index $1/K$ indicates that the word has no discriminative power, whereas a gini-index 1 implies full discriminative power. For simplicity we used our own Python implementation of MNB.

$$G(w) = \sum_{c=1}^{K} \left(\frac{p_c(w)/P_c}{\sum_{i=1}^{K} p_i(w)/P_i}\right)^2 \quad (4)$$

2) Support Vector Machines: The Soft-margin cost sensitive SVM has the primal [3]:

$$\arg\max_{\boldsymbol{w},b,\boldsymbol{\xi}} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + \boldsymbol{1}\{y_i = 1\}C_+ \sum \xi_i + \boldsymbol{1}\{y_i \neq 1\}C_- \sum \xi_i \tag{5}$$

$$st: \quad y_i(\boldsymbol{w}^T\boldsymbol{x} + b) \geq 1 - \xi_i \tag{6}$$

Here $\boldsymbol{w}$ is the weight vector; $\boldsymbol{x}$ is the feature vector; $\boldsymbol{1}(.)$ is an indicator function which returns either 0 or 1; C_+ (C_-) is the cost for misclassifying a positive (negative) example; and ξ_i is a slack variable through which misclassification is allowed to deal with noisy data. Such a formulation of SVM is helpful when making mistakes for one class is more expensive than other. Finally, SVM^{light} was used for our experimentation [6].

IV. Experiments and Result Analysis

In this section, we attempt to show the effectiveness and weaknesses of different machine learning treatments of off-topic discussion filtering.

a) Parameter tuning: The trained models (using SO posts and YouTube comments) for MNB and SVM were validated using the *Gold_Cross_Set* data (not the test data). We adjusted the model parameters based on the following observations: 1) The Snowball stemming algorithm performs better than the Porter stemming for both MNB and SVM; 2) For MNB, selecting the most important features using gini-index works better when words with less than three occurrences are deleted; 3) In case of SVM, vector representation with term-frequency outperforms the binary representation; 4) Classification accuracy of SVM improves dramatically if the intercept parameter (b) in equation 6 is set to zero; and 5) Simple linear kernel offers much better performance than non-linear kernel for IRC filtering—complementing earlier findings in text classification [13].

b) Accuracy: We used the *Gold_Test_Set* to evaluate the performance of both MNB and SVM. Table I shows the classification accuracies of MNB (using all the parsed words in training) and SVM (using all the words and equal penalty for false positive and false negative, i.e., $C_+ = C_-$ in equation 5). The results are interesting and offer different avenues of improvements—although the F-scores are very similar, the two algorithms outperform each other significantly when precision and recall are considered separately. For example, when SVM predicts a message as positive, the chance is very high that the prediction was correct (i.e., SVM offers high precision). Unfortunately, SVM also misclassifies a significant number of positive examples (i.e., low recall), which can be unacceptable for lots of IRC users. Conversely, MNB offers better reliability by detecting almost 97% positive examples correctly, although the noise reduction is much lower than SVM.

c) Precision-recall trade-offs: A system which enables the filtering out of off-topic messages is encouraging and can be adopted immediately. We believe that, for a context like IRC, a method with 100% recall and 80% precision is more desirable than a method with 90% recall and 90% precision—it is unacceptable to classify a programming discussion as off-topic, in contrast to labelling an off-topic message as a programming message. To achieve that goal, we try different modeling approaches—selecting only the important words as features for MNB, and imposing higher penalty for false negative than false positive in SVM. Figure 1(a) shows the impact of increasing the number of words on MNB's precision and recall; gini-index was used as the ranking parameter for selecting the most important words. The prediction accuracy of MNB is very poor if the number of words is less than 50,000. Yet the recall is better (very close to 100%) with the top 50,000 words than with the top 60,000. Thus we recommend MNB with 50,000 most important words for IRC as it avoids missing any programming discussions with 65% off-topic message reduction. Interestingly, the performance of SVM was significantly worse for any selected number of words compared to using all words. This could be because of SVM's capability of overcoming the over-fitting problem by separating positive and negative examples with the largest possible margin—consequently, increasing the number of features offer better accuracy as redundant features, if there is any, are excluded automatically.

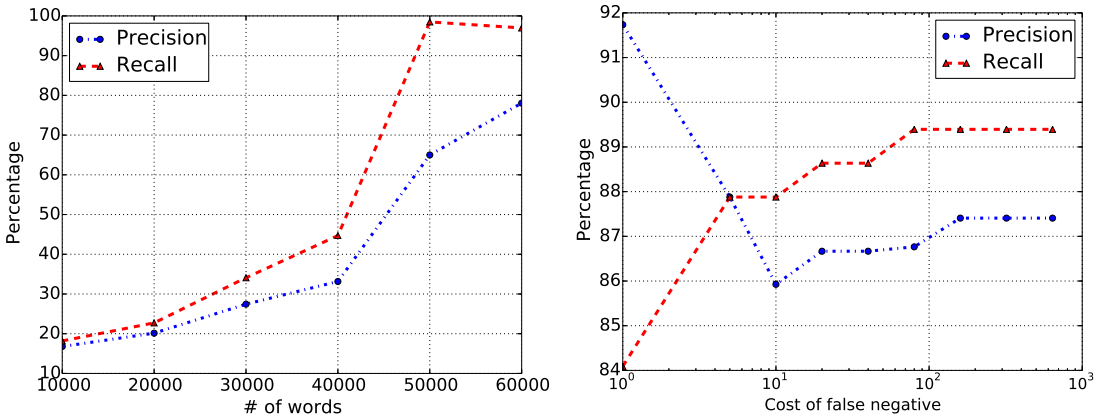
TABLE I
Performance of Classifiers on *Gold_Test_Set* (IRC messages)

	Precision	**Recall**	**F-score**
NaiveBayes	80.5%	96.9%	87.9%
SVM	91.7%	84%	87.6%

We also experimented with cost sensitive SVM to make the trade-off between precision and recall. Figure 1(b) starts with equal cost for both false negative and false positive, showing much higher precision than recall for such a configuration. Not surprisingly, an improvement in recall with a degradation in precision is observed as the penalty for false negatives is increased. The recall and precision become 89.4% and 87.4% respectively when the cost of false negative is 160 times more than the cost of false positive; and after that increasing the sensitivity does not change the result as no more false negative occurs in the training data. Although this is an improvement, in terms of reliability, over the previous cost insensitive methodology, the recall of SVM is still much worse than the recall of MNB with 50,000 words.

d) Discussion: Our initial experiments suggest that filtering off-topic discussion out from programming discussions, such as from the IRC python channel, is possible with acceptable accuracy. Our work can be extended substantially by experimenting with more IRC data—the IRC discussions we used for testing can be biased although we have tried to select IRC messages from different aspects of programming. Moreover, the size of the test set can be increased by selecting and labelling more messages through manual inspections.

Figure 2 shows the prototype implementation (using MNB with 50,000 selected words) for filtering actual IRC messages. Positively predicted messages by MNB are coloured black in contrast to gray for negatively predicted messages. This type



(a) Multinomial Naive Bayes

(b) Cost sensitive Support Vector Machine

Fig. 1. Trade-offs between Precision and Recall: Vocabulary Size of Multinomial Naive Bayes, and False Negative cost of a Support Vector Machine.

of system would be very helpful for IRC users to get through quickly on programming related messages. Another important aspects of adopting such a system is that inappropriately written programming messages (especially when the message is very short) are sometimes misclassfied, thus incentivizing the users to write more formally. For example, the second last message in Figure 2 should have been coloured black instead of gray, as *BeautifulSoup* is a Python package for parsing HTML documents; and in our models *BeautifulSoup* has a very high discriminative power when written without an intermediate space. Interestingly, the second last message in our figure is predicted as on-topic if we remove the space (shown in the last message)—motivating the users to write more precise posts.

<anonymous> yes, i have a class element that has .parent
<anonymous> Stick some debug info the "if self.parent:" block, at line 12.
<anonymous> so I guess it's not the canadians fault
<anonymous> I was a philosophy "major" (we don't use that term here) so I'm pretty useless to anyone.
<anonymous> e.g. get it to print the output of c.getOrigin(...) in the block there
<anonymous> country and perhaps major city are the best you can hope for
<anonymous> o = element.getOrigin(); debLog("final origin: %s" % (o))
<anonymous> use beautiful soup
<anonymous> use beautifulsoup

Fig. 2. Example of how the off-topic filter could be used on IRC: black text indicates relevant, grayed out text indicates off-topic.

V. Conclusion

We evaluated the performance of two machine learning algorithms to detect off-topic discussions in programming related IRC channels. These classifiers could be used to highlight on-topic messages for an IRC user. This methodology is accessible to users because the training sets require no annotation on their part: StackOverflow posts and YouTube video comments for classifying IRC messages can be curated and shared to enable building topic-specific classifiers, without manual annotation. We have shown that StackOverflow acts as a corpus of software specific texts that can be used by end-users to avoid manual annotation. Future work is to evaluate this tool with users on different contexts and channels.

Acknowledgment

Shaiful Chowdhury is supported by the Alberta Innovates - Technology Futures (AITF) to pursue his PhD research.

References

[1] C. C. Aggarwal and C. Zhai. A survey of text classification algorithms. In *Mining Text Data*, pages 163–222. Springer, 2012.
[2] S. Bird. Nltk: The natural language toolkit. In *COLING-ACL*, pages 69–72, Sydney, Australia, Jul. 2006.
[3] P. Cao, D. Zhao, and O. Zaiane. An optimized cost-sensitive svm for imbalanced data learning. In *Advances in Knowledge Discovery and Data Mining*, volume 7819, pages 280–292. Springer, 2013.
[4] E. Frank and R. R. Bouckaert. Naive bayes for text classification with unbalanced classes. In *PKDD*, pages 503–510, Berlin, Germany, Sep. 2006.
[5] K. Javed, S. Maruf, and H. A. Babri. A two-stage markov blanket based feature selection algorithm for text classification. *Neurocomputing*, 2015.
[6] T. Joachims. Making large-scale SVM learning practical. In *Advances in Kernel Methods - Support Vector Learning*, chapter 11, pages 169–184. MIT Press, Cambridge, MA, 1999.
[7] C. A. Lampe, E. Johnston, and P. Resnick. Follow the reader: Filtering comments on slashdot. In *CHI*, pages 1253–1262, San Jose, California, USA, April 2007.
[8] B. Liu, Y. Dai, X. Li, W. S. Lee, and P. S. Yu. Building text classifiers using positive and unlabeled examples. In *ICDM*, pages 179–186, Melbourne, USA, Nov. 2003.
[9] E. Shihab, Z. M. Jiang, and A. Hassan. On the use of internet relay chat (irc) meetings by developers of the gnome gtk+ project. In *MSR*, pages 107–110, Vancouver, Canada, May 2009.
[10] M. Squire and G. Rebecca. FLOSS as a source for profanity and insults: Collecting the data. 2015.
[11] StackExchange. http://data.stackexchange.com/stackoverflow/queries last accessed: 28-mar-2015.
[12] A. T. T. Ying. Mining challenge 2015: Comparing and combining different information sources on the stack overflow data set. In *MSR 2015*, page to appear, 2015.
[13] W. Zhang, X. Tang, and T. Yoshida. Tesc: An approach to text classification using semi-supervised clustering. *Knowledge-Based Systems*, 75(0):152 – 160, 2015.