Progress Report
Rose Perrone, Chi Zheng, Guoxing Li

Tagger: Generating Tags for Stack Exchange Questions

http://www.kaggle.com/c/facebook-recruiting-iii-keyword-extraction

The algorithm we chose for our problem of text classification is SVM, in favor of Multinomial Naive Bayes and k-Nearest Neighbors.

We chose among the following options, but the main question was whether to use SVM or Multinomial Naive Bayes.

The following papers show that SVM performs (sometimes much) better for general text classification.

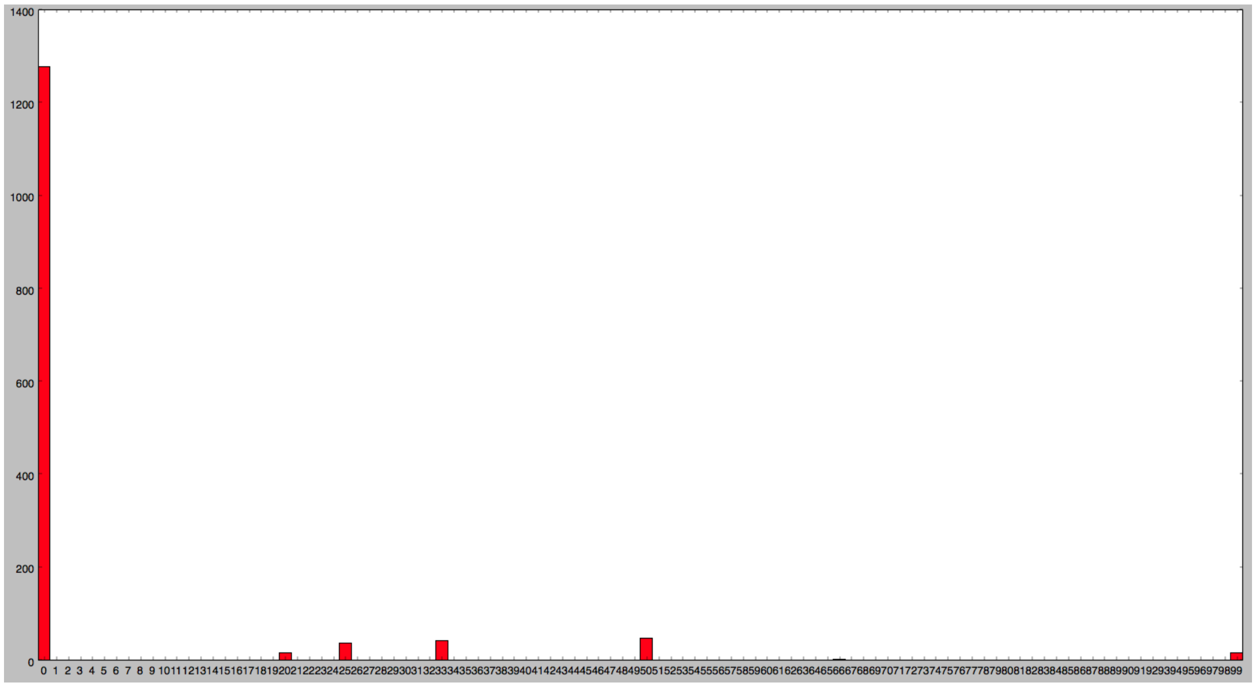
Comparing SVM and Naïve Bayes Classifiers for Text Categorization with Wikitology as knowledge enrichment (2011) (arxiv.org/pdf/1202.4063)
Study on SVM Compared with the other Text Classification Methods (2010)
(http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5459006&tag=1)
Content-based Text Categorization using Wikitology (2012)
(http://arxiv.org/abs/1208.3623)
Text Categorization for Multi-label Documents and Many Categories (2007)
Text Categorization with Support Vector Machines
(cited by 5877) (Joachims)
http://www.cs.cornell.edu/people/tj/publications/joachims_98a.pdf

These articles made bold statements like, “The experimental results show that SVMs consistently achieve good performance on text categorization tasks, outperforming existing methods substantially and significantly.” (Joachims)

One downside to using the SVM is it’s more expensive than Naive Bayes, and we have 7GB of data to run through. For now, we’re not taking performance too much into account, because we don’t yet know the scale of the feature space we’ll be using on the 7GB. We may prefer kNN, because “this algorithm continues to achieve very good results and scales up well with the number of documents, which is not the case with SVM. As for naive Bayes, it also achieved good performance” according to: Comparison of SVM and Some Older Classification Algorithms in Text Classification Tasks
http://sedici.unlp.edu.ar/bitstream/handle/10915/23885/Documento_completo.pdf?sequence=1
This chapter, titled “Comparison of Classifiers”, covers the performance drop of SVM in detail (https://openaccess.leidenuniv.nl/bitstream/handle/1887/13575/Chapter%207.pdf?sequence=12)

Comparison of SVM and Some Older Classification Algorithms in Text Classification Tasks

Results of our SVM implementation:
==============================
Our baseline implementation splits the training data into about 400 files, and trains a scikit multilabel classifier on one of the files containing 2MB (about 1/4000th of the full dataset by bits). I then train the dataset on another 1/4000th of the training set. Here's the histogram showing the classifier's success. The y-axis is number of testing examples, and the x-axis is the percentage of tags predicted correctly.



Here's how our baseline system works:

- Load one/four-thousandth of the training data into a python array
- Strip the HTML markup and newline characters in the titles and bodies
- Merge the titles and bodies so there is one text entry to classify
- Turn the bodies of text into a matrix such that each row represents a body of text, and each column is the count of that word in the body of text. This step can incorporate bigrams, but for now, we just use unigrams.
- Use a term frequency inverse document frequency transformer that decreases the importance of frequent words like "the" and "is".
- Use a One vs Rest classifier that uses a support vector classifier.

Plans for improvement

Finding out which of these domain-specific features matter:
- question contains code markup (blockquote, LaTeX, grid of data)
- If there's code, classify its language based on some library
- question contains latex markup
- unigrams (this is a huge one)
- bigrams
- trigrams?
- length of description
- length of question
- Is there a high frequency of one pronoun?
- questions contains numbers
- misspellings
- redundancy on question and description
- is some text in a foreign language
- poor grammar
- number of question marks
- confidence of asker (do they use weak phrases that inexperienced posters typically do, like "Any
- use of a TextBlock
- Use of blockquote
- human names (Jeff, Amanda)
- are there many other questions that may be the same as this one? It probably has a popular tag.
- is there emoji? :)
- are sentences capitalized?

As soon as we decide how to parse these features, we'll can use Filter Feature Selection (rather than the more computationally expensive Forward Search procedure) to select the features.

For each feature, we may also select which algorithm works best to classify the feature. We could continue to use teh same SVM and treat each feature just like a word frequency -- that is, we've already seen how to count the instances of words, find the fraction of the word out of all words tagged with the label, and multiply by the class prior to find p(tag | word). For example, consider misspellings. We could take the a corpus which is commonly misspelled words, find the frequency of misspelled words, and we're off and running.

We also need to figure out how to work in the thresholds for the number of tags each message has, and use that to inform our algorithms.
- there needs to be at least one tag
- find the variance of the number of tags
- learn what makes a question have many tags.

    - e.g. History questions only have 1-3 tags,
      but coding questions have 4-7
- learn what makes a question have more tags. Should it
  simply be the tags that pass a certain threshold?

It would be interesting to improve the text classification by keeping track of synonyms, as this paper shows: Improving Text Classification by Using Encyclopedia Knowledge
(2007)
(http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4470257)
The categorization of the English language might also be a good feature. That is, for every word, categorize it as, PERSON (<human name>, "Jesse") or ORG (companies, teams, "Microsoft"), GPE (that is geographical: "United States", "Republic"). This paper shows that technique:
Mining Wiki Resources for Multilingual Named Entity Recognition
http://www.mt-archive.info/ACL-2008-Richman.pdf

We can also apply some machine learning to the tags themselves. That is, we could generate a corpus of words that could be tags that are categories, like "artificial intelligence" and "ai", and we could group together those tags.