Classifying Reddit Submissions
CS 229A Final Project

Stanislav Moreinis

Dec. 16, 2011

1 Introduction

This project concerns itself with user submissions to the social news site Reddit. Each post to this site must be made to a particular subreddit, which can be thought of as a message board for a collection of users interested in a (possibly loosely defined) topic. The largest subreddits are generally oriented around a single dimension of the posts - which can either be content-based, like the subreddits for politics and gaming, or presentation-based like subreddits for pictures and videos. The goal of this project is to create a statistical classification system which, trained on a sufficient amount of new submissions (labelled with their intended subreddit), will be able to classify future submissions into these subreddits. We will explore our model's performance on an increasing number of top subreddits, both to better characterize its performance and to monitor the amount of ambiguity introduced by new categories.

2 Data Extraction

To collect our data, we downloaded the new posts every 2 minutes from Reddit, collecting 3 roughly equal-sized data sets, each corresponding to a week of submissions. Each new Reddit submission has a link to some material (unless it is a text-only self post) and relatively short headline text (which may or may not come from the link itself). To extract useful information from the link about the link content, we get its domain name with the assumption that the domain's relevance for each particular subreddit is constant across all of the content hosted on the domain. To unite equivalent domains like "i42.tinypic.com" and "i43.tinypic.com", we only keep the top-most domain name (in this case "tinypic.com") - while this has the benefit of better estimating load-balanced content, we lose the granularity of authorship on blog posts from sites like Google+, Tumblr, and Posterous (which host blogs at domains like author.site.com). Finally, because every text-only post is tagged with a domain containing its subreddit, we strip this information and label the domain of a self post to any subreddit with "self" (as these are called self-posts).

The other part of our input is the headline text associated with each submission. Like most text classification problems, we first sanitize our text by lower-casing all of the words, and removing the punctuation along with a rather conservative list of stop words. We then proceed to use the Snowball algorithm to map each token in the title to its stem, which allows us to reduce the feature sparsity due to verb conjugations or noun plurality. Finally, we construct separate matrices for the bag-of-words vectors representing the words used in a post's title, and the vectors representing the posted link's domain. Having made our title occurrence count matrix as dense as we can, we also apply the TF-IDF transformation to it, which multiplies each term in it by a calculated weight. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The weight increases proportionally to the number of times a word appears in the document, but is offset by the frequency of the word in the corpus. In effect, this scaling lets our models which words are more important and which aren't - instead of the uniform weighting we had initially, which can make a difference in models sensitive to scaling (e.g. stochastic gradient descent).

3 Model Construction

Although it was clear to us that both the post's title and domain carry significant information about its potential target category, it was unclear what the best approach to exploit both of these pieces of data would be. To evaluate potential ways of combining our features, we decided to construct a classifier model from just one feature set to serve as our baseline. It was clear (jugding by the sparsity alone) that the post title could offer us more nuanced insight into the post content than the domain, so this was our chosen baseline feature set.

3.1 Building Blocks

Having determined our feature set, there are several basic models that we can train to perform supervised classification. The first is a logistic regression classifier (which we will refer to as the *log* model), regularized either using the L1 or L2 penalty. Because we can only train this classifier with positive and negative labels, the model we use will use the one-vs-all strategy to assign labels of multiple classes. In other words, for K possible labels, we will train K classifiers (labeling the examples in the category positive and all others negative). To make a prediction using this model, we will make the decision of the classifier with the highest confidence score. The second is another linear model classifier which uses stochastic gradient descent to minimize regularized empirical loss. We will use this model (which we will call *sgd*) with a logistic regression loss function and regularize it using either the L1, L2 penalty, or a convex combination of the two ('elasticnet'). Because it is a binary classifier just like our previous model, we will use a similar approach to apply it to our multiclass problem.

Our last two models are Naive Bayes classifiers which can be trained on discrete data. The first assumes the data fits a multivariate Bernoulli distribution, while the second

assumes it fits a multinomial one. While the Bernoulli Naive Bayes classifier (the *bnb* model) is designed for binary features, the Multinomial Naive Bayes classifier (the *mnb* model) lets us take advantage of occurence counts and the weighting introduced by our TF-IDF transformation. We chose not to document the application of SVM models to our classification process (although they are often a popular choice), as these required a significant amount of tuning to achieve optimal performance - and provided no advantage to warrant the added complexity in our experiments. A final note of interest is that all 4 of these models are capable of predicting both class labels as well as individual class probabilities for our test data - which allows us to combine model predictions over a set of labels, something we make use of later.

3.2 Text Classification

Armed with the models from the previous section, we construct our set of baseline classifiers which only use a post's title for training and testing. To get a feel for the performances of the different models, we first tried to train on and classify posts from the 5 most popular categories for our week only (which in our case were 'AdviceAnimals', 'pics', 'videos', 'skyrim', and 'atheism'). This yields us a split dataset of 10,194 training posts and 1,799 test posts - which are classified using the 7,865 token features extracted from our training set. As we can see from our results (not included here because of the 5-page limit), all of our classifiers achieve essentially similar performance on this dataset (with an F-score around 0.6) - which, combined with the lack of significant variation of accuracy across different categories for any classifier model, suggests that there is no significant content overlap between our chosen categories. Using our *log* and *sgd* models, we can extract the features with the highest coefficients for each individual category, and the lack of any overlap only drives home our point about the categories being well-defined (relative to each other).

As we would expect, our model's performance deteriorated as we expanded the training and test posts used to the top 20 categories of our dataset (which are 'AdviceAnimals', 'pics', 'videos', 'skyrim', 'atheism', 'WTF', 'firstworldproblems', 'Music', 'reportthespammers', 'aww', 'occupywallstreet', 'politics', 'Minecraft', 'starcraft', 'battlefield3', 'tf2trade', 'circlejerk', 'mw3', 'fantasyfootball', 'buildapc'). This time, we extracted 13,545 token stems from 21,481 training posts and tested our models on the other 3,791 posts.The most obvious characteristic of our results is that each classifier's performance varies greatly by category - which suggests that our categories are no longer as well defined relative to the other ones as before. Looking at the categories which our models struggle with, 'pics', 'videos', 'WTF' feature prominently as consistent offenders - which makes sense as they are the most loosely defined categories of our list. We can also see that categories like 'politics' and 'occupywallstreet' are very likely to be confused, which is reinforced by the fact that 'occupi' (and similar token stems) occupy the top .1% of the coefficients of both categories in most of our models. On the average, our models perform comparably, although it is clear that the Naive Bayes models are more prone to have varying performance - with poor performance on some categories being cancelled out by superior performance on others (or not, as is more the case for the multinomial

classifier, which has the lowest average of all by some distance).

3.3 Combined Classification

Having established a reasonable classification baseline using post titles alone, we proceeded to adding the post domain to our feature set. For a first approximation of how much useful information is to be gained from this change, we simply appended the domain feature vector for each post to its title feature vector. The main problem with this approach was that significant variations in the post domain could be ignored due to the overwhelming relative sparsity of the domain occurrence matrix compared to the title occurrence matrix, but it was unclear whether there is enough interaction between title and domain features to warrant this approach (as well as the added complexity that it requires). Having tested it on the list of top 5 categories from above, we can see that there is value added by this change, as the category and average F-scores rise to about 0.7 for all of the models. Expanding our selection to 20 categories, however, we see that this approach actually yields worse results than if we'd simply used the title alone, as the average F-score falls to 0.35 (compared to 0.5 we had from the title classifier), and most of the models fail to classify the less popular categories completely (i.e. an F-score of 0.0).

Having decided to model the effects of the title and domain on the relevance of the post to a particular category independently, we now construct a simple classifier model which does just that. To be more precise, for some basic model type m, we will construct two classifier models m_{title} and m_{domain}. Each one will be trained on the labeled title and domain data, respectively, to train the overall classifier. To generate a prediction for an example, we will first obtain the individual class probabilities (instead of just the optimal label as before) of the example given its title and domain - call them p_{title} and p_{domain} respectively. Then for some category c, we will have the probability of the example being classified in this category $p(c) = p^{\alpha}_{title}(c) * p_{domain}(c)$. We can use the α term to control the relative importance of the title probability versus the domain probability, although we began by weighting them equally (i.e. $\alpha = 1$) and were unable to find a better choice. Finally, to classify our example we simply return the label c that maximizes $p(c)$.

We could immediately see the improvement in performance after running this combined classifier on the posts from only the top 5 categories, as the average F-score was now around 0.85 - with no significant performance variation by individual category or model type. This improvement was also reflected once we considered the posts from the top 20 categories, as we can see that the average F-score is now around 0.60 (which is a significant improvement on using the title only). Our model is much more robust as well, as the minimum F1-scores are now within an acceptable range and no categories are completely misclassified by any of our models. Furthermore, we can see that the topics our new model is likely to misclassify are exactly the ambiguous topics we found earlier (e.g. 'pics, 'videos', 'WTF'), so the optimization cannot be blamed too much for failing to find any meaningful patterns in training data from those categories.

4 Conclusion

In conclusion, we've shown a relatively straightforward but effective method to construct a model of the submission guidelines for our selected categories. The modularized approach also makes it simple to extend the system - as it would most likely be beneficial to train the title model for each category on the text from the comments in that category as a means of reducing the sparsity in the training data. We could also perform additional clustering on the domain names (to segregate them into types like 'news', 'social', and 'image hosting', for example), which would again reduce the sparsity of our training data, and allow us to better estimate the parameters needed for making predictions on the test data.

Furthermore, because our final classification uses underlying probabilities to make its decision, we can also use them to suggest a ranking of potential categories for a given new post. Finally, although all of the new posts in our training set were uniformly weighted, we could utilize the feedback loop which made Reddit so popular, and weight them proportionally to the number of upvotes the post receives after it's been submitted. The resulting model would then not only be able to suggest categories where content similar to a given post is submitted, but would be able to rank the categories in the order of probability that content similar to a given post will be received positively. Alternatively, we could also use a weighting consistent with personal interest in a given post to construct a rudimentary recommendation engine.