CS 229: r/Classifier - Subreddit Text Classification

Andrew Giel
BS Stanford 2015, Computer Science
agiel@stanford.edu

Jonathan NeCamp
BS Stanford 2015, Computer Science
jnecamp@stanford.edu

Hussain Kader
BS Stanford 2015, Computer Science
hkader@stanford.edu

***Abstract*—This paper presents techniques for text classification of reddit posts over 12 subreddits. Leveraging a variety of natural language processing techniques such as lexicalized features, TF-IDF weighting, sentiment classification, parts-of-speech tagging, and Latent Dirichlet Allocation along with machine learning practices such as filtered feature selection, Principal Component Analysis, and multinomial classifiers as well as domain-specific knowledge we were able to construct systems capable of high F1 over many classes.**

I. The Task

Reddit is one of the largest anonymous online communities in the world, with over 114 million unique users per month. Reddit is a collection of interest-based communities known as subreddits, whose content vary from news, to sports teams, to hobbies, to basically anything you can imagine. When posting a link or text post to reddit, one must select the subreddit to post to, as each post lives in a particular subreddit. Users can upvote or downvote posts, expressing approval or disapproval with the content of the post. The number of upvotes and downvotes are fed into a hot-ranking algorithm to determine a score for the post, with higher scoring posts rising to the top of the subreddit.

Our task is simply: given a text reddit post composed of a title and body, classify the subreddit the post belongs to. This can serve two main functions:

1) to lower the barrier to entry for new users to reddit who do not know which subreddit to post to
2) to help suggest which subreddit a post will be most successful in, helping users to achieve high visibility for their content

In order to make this project tractable, we reigned in the scope of the task. Currently, there are over 300,000 active subreddits, with varying degrees of activity. We chose a subset of 12 subreddits to classify over, with hopes that our efforts here can generalize over larger domains. Similarly, we chose to only classify text (self) posts, and not links. The majority of links shared on reddit are from image hosting sites such as imgur.com. Object classification is notoriously one of the hardest tasks in computer vision and machine learning, with cutting edge techniques only now beginning to show large improvements in performance. Avoiding these bleeding-edge pursuits, we have focused on text-posts, making much of our task NLP related. We feel that these two adjustments to the task definition allow for us to expect reasonable performance while remaining an academic and implementation challenge.

This task is commonly referred to as a text classification problem. More formally, given a post

$$p \in D$$

where D is the space of all reddit posts, and given a set of subreddits $S = \{s_1, ..., s_k\}$, train a classifier

$$h : D \to S$$

that appropriately determines the optimal subreddit assignment of p.

II. The Data

Our dataset can be found at: https://github.com/umbrae/reddit-top-2.5-million

We are using data from twelve subreddits:

NoStupidQuestions	*Showerthoughts*
shortscarystories	*DebateReligion*
confession	*relationship_advice*
UnsentLetters	*self*
askphilosophy	*ShittyPoetry*
AskMen	*AskWomen*

These particular subreddits were chosen based on an analysis that showed them to be the the subreddits with the largest percentage of text-only posts that had no very easily identifiable features. For example, when posting in *r/todayilearned* users lead their posts with the tag "TIL"- effectively making it incredibly easy to build a simple model that consistently correctly classifies every post from *r/todayilearned*. We didn't want to work with subreddits that had features such as these because we believed it would trivialize our task, so we particularly selected our subreddits to be vague enough to make the task interesting. Each post contained in the dataset is made up of information spanning everything from the author of the post to the number of upvotes. The only elements that we use are the title of each post and its text contents. In total, we have 12,000 posts which amounts to 1,000 posts from each of the twelve different subreddits.

III. Features

In order to give our models information regarding the correct classification of a reddit post, we used lexicalized features based on the text within a given post, incorporating multiple natural language processing techniques to do so. As is the case for many text classification problems, we found ourselves spending the majority of our time experimenting with different combinations of feature representations. Additionally, since most of our feature representations revolved around a *bag-of-words* model and the size of the set of possible words is quite large, we experimented with different ways of reducing our feature space.

A. Base Reddit Post Representation

Our base way of representing a reddit post in a form able to be used by our learning models was with the *bag-of-words* approach. That is, we created feature vectors where each element holds a weighted value corresponding to a word (or word pair in the case of bigrams) in the vocabulary. (NOTE: We experimented with using unigram and bigram terms and found unigrams to perform the best, so for the rest of this paper we consider only unigrams.) More formally, for some subreddit post p over some vocabulary $V = \{w_1, w_2, ..., w_n\}$, we created a vector $\phi_p =< a_1, a_2, ..., a_n >$ where a_i is some weight associated to w_i. We used two different methods for calculating this weight term, a_i, for a word:

1) **Binary:** a_i held either a 0 if w_i did not show up in the post p or a 1 if it had appeared in p. Using this weighting meant our feature vectors were binary vectors. This may seem like an overly simplistic of representing a post but it actually performed quite well and was computationally fast.
2) **TF-IDF:** This type of weighting aims at more accurately representing the reddit post as a mathematical object, taking into account term-frequency instead of just a binary value. In particular, TF-IDF weights are found as follows

$$tf(w_i, p) = 0.5 + \frac{0.5 \times f(w_i, p)}{\max\{f(w, p) : w \in p\}}$$

$$idf(w_i, D) = \log \frac{m}{|p \in D : w_i \in p|}$$

$$tfidf(w_i, p, D) = tf(w_i, p) \times idf(w_i, D)$$

where D is the set of all posts, $m = |D|$, and $f(w_i, p)$ is a function returning the number of times word w_i appears in post p.

One qualitative way to assess the helpfulness of binary and TF-IDF weighting in practice is to visualize the vectors. Using t-Distributed Stochastic Neighbor Embedding (t-SNE), a dimensionality reduction technique specifically helpful for visualizing high dimenionsal data, we plotted 2000 feature vectors corresponding to 2000 different reddit posts, as seen in Fig. 1 and 2. The color of a given point corresponds to the subreddit the post belongs to. One can notice that points in the TF-IDF plot are more clustered by color than in the binary plot. This seems to mean that TF-IDF weighting is better at representing documents as vectors where similar vectors correspond to posts belonging in the same subreddit. Although, in practice we found that binary vectors often performed similiarly to TF-IDF in terms of F1.

Initially, we started with just these base representations of the text of a reddit post, where the text was the combination

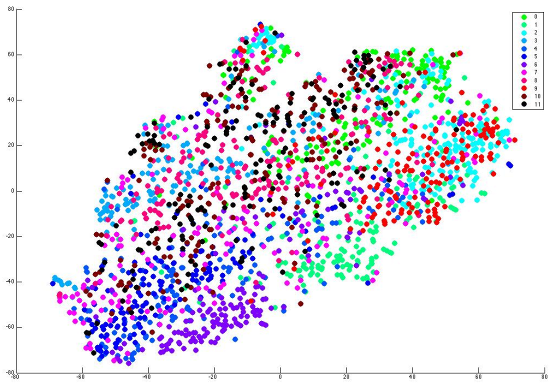

Fig. 1. Binary vectors in R^{3000} visualized in R^2 using t-SNE.

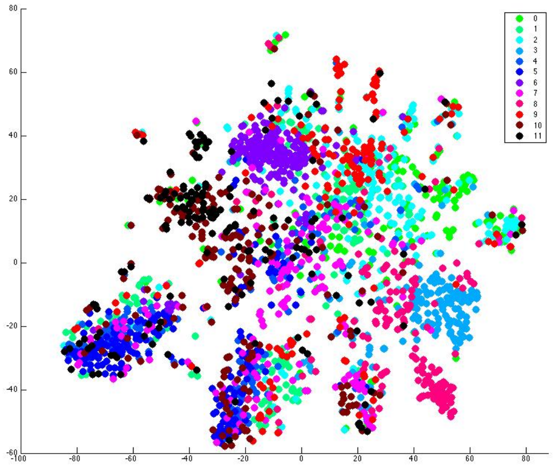

Fig. 2. TF-IDF vectors in R^{3000} visualized in R^2 using t-SNE.

of the text-body and title. However, there are a few problems with using just this approach. First, representing posts over a large vocabulary means having feature vectors of large dimension which can be computationally unwieldy and also lead to overfitting. Second, the *bag-of-words* approach is rather simplistic as it disregards word-ordering in a given post as well as higher-level post information. These problems and measures taken to remedy them are the topics of the sections that follow.

B. Reducing Feature Space Dimensions

There are $\approx 50,000$ different words in the 10,800 reddit posts we train over. Putting each of these words in our vocabulary means having feature vectors with $\approx 50,000$ dimensions. Initially, this is what we did and we were able to train our models. Although, it took a very long time to complete the training and we found this to not be conducive towards rapid experimentation. Additionally, since $50,000$ dimensions is larger than the $10,800$ posts we train over, our models could be prone to overfitting. To tackle both of these issues, we experimented with two methods of reducing our feature space:

1) Feature Selection via Mutual Information: Although 50,000 different words appear in the posts, only some of these are telling as to what subreddit a post belongs. As such, we wanted to intelligently select a subset of the words, filtering out those which give us little to no information regarding the classification task. We chose to filter using the notion of Mutual Information. Using the notation found in the Novovicova paper, we defined MI between a set of classes C and some feature w_i as follows

$$MI(C, w_i) =$$

$$\sum_{k=1}^{|C|} P(c_k, w_i) \log \frac{P(c_k, w_i)}{P(c_k)P(w_i)} + \sum_{k=1}^{|C|} P(c_k, \bar{w}_i) \log \frac{P(c_k, \bar{w}_i)}{P(c_k)P(\bar{w}_i)}$$

where $\bar{w}_i$ indicates that the word did not occur. Note that MI is the sum of the Kullback-Leiber (KL) divergence between the class distribution and the feature-presence distribution and the KL divergence between the class distribution and the feature-non-presence distribution. Intuitively, Mutual Information gives us a quantitative assessment of how 'helpful' a feature (in this case word) will be in classifying across our $K = 12$ classes. As seen in Fig. 3, filtering with MI performs much better than randomly selecting the subset of words to be used.

2) Principal Component Analysis: We used the dimensionality reduction technique of PCA to reduce feature vectors to a more manageable dimension. Although this helped with overfitting and achieved similar performance to feature selection with MI, performing PCA on the original large vectors is nearly computationally intractable itself (we actually had to use a faster randomized variation of PCA).

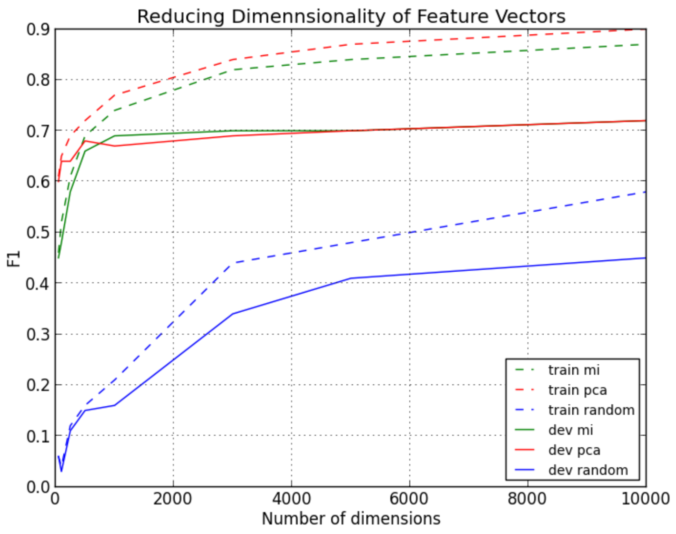

Fig. 3. Reducing dimensionality via random selection, MI, and PCA

C. Additional Features

Using just the *bag-of-words* features gave decent results. However, we discovered a slight modification by considering title and body separately performed noticeably better. Additionally, combining lexical features with features that gave higher level information such as word count, Latent Dirichlet Allocation topic distributions, sentiment scores, or number of Parts of Speech tags gave us consistently higher scores than any of the individual component feature representations. This was a major takeaway from this project.

1) Title Split: We wanted to create a feature that utilized some domain-based knowledge of reddit in order to boost our performance. We realized that there was a lot of implicit information lost when combining the title text and body text together. This led us to create a featurization procedure we called 'Title-Split'. Instead of lumping the title and body text together, we selected features and created feature vectors completely separately, then concatenated these two vectors to create our feature vector ϕ

$$\phi = [\phi_{title} \quad \phi_{body}]$$

We found this to be a very useful feature. Some of the subreddits we were experimenting on proved to be especially impacted by this feature, as a large portion of the subreddits which were question based (eg. AskMen) would contain only a title and no body. 'Title-Split' helped to encapsulate the separation of information found within a reddit post title and its body.

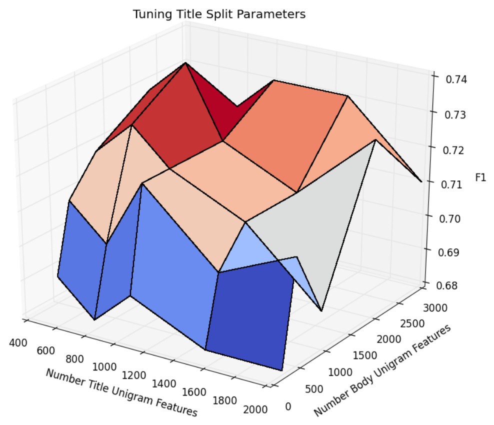

Fig. 4. Tuning the hyperparameters n_{title} and n_{body}

2) Latent Dirichlet Allocation for Featurization: Our most experimental feature vector representation was based on the topic distribution vectors inferred by Latent Dirichlet Allocation (LDA). LDA is a generative topic-modeling algorithm, that assumes that a document is created as a mixture of a finite number of topics, and each topic has some distribution over words. Given a corpus of documents and the number of topics k, LDA infers the the topic mixture model $\theta \in R^k \sim Dirichlet(\alpha)$ and the topic-to-word distributions $\phi \in R^{k\times|V|} \sim Dirichlet(\beta)$. Given a new document d, LDA will predict the topic distribution $\theta_d \in R^k$ in the form of a vector whose elements sum to 1.

Our hypothesis was that LDA was a perfect tool for our task, as we have a finite number of subreddits which are communities for a variety of topics. In order to utilize the topic models, we trained an LDA Model on the post text, presenting each post as a document (not applying 'Title-Split'). Once these models were trained (α and β are estimated via Expectation-Maximization), we ran our dataset through these models again, giving us the predicted topic distribution vectors for each of our titles and bodies. This vector of topic distributions $\theta_d = \phi \in R^k$ which we then gave to our classifier (The notation here is overloaded). At test time, we used the LDA models created during training to create θ_d and then fed this as ϕ to our linear model.

3) Word Count: One simple feature we used was simply the number of words in a post. We found this very simple feature to be very powerful, especially when combined with other features.

4) Sentiment Score: The sentiment score proved to be an interesting feature that did not make a large difference in the overall average F1 score, but did affect some specific subreddits quite substantially. For every post, we calculated a sentiment score that was a float ranging from -1 to 1. Some subreddits, like *relationship_advice*, *AskMen*, and *AskWomen* did significantly better with sentiment as a feature. Sentiment definitely helped us tell the difference between *AskMen* and *AskWomen*, which is something we struggled with throughout. Unfortunately, sentiment did not work well with some subreddits - particularly *shortscarystories* and *ShittyPoetry*, both of which it caused significant decreases in accuracy in. We believe this is because things such as poetry and stories can have a wide range of different sentiments, so there is less reliability in a sentiment score being indicative of the category.

5) Parts-Of-Speech Tagging: For our Parts-Of-Speech tagging feature, we tagged the part of speech for each word in the post and then iterated through the post and counted up the total number of adjectives, nouns, proper nouns, and verbs. We then normalized these numbers to account for the fact that all of our posts are of varying sizes. The results of the Parts-Of-Speech tagging feature were similar to sentiment in that it worked pretty well for some subreddits, but then was significantly worse for others. We saw a huge positive increase in our F1 for *DebateReligion* and *askphilosophy* but took subtantial hits in accuracy for *Showerthoughts* and *ShittyPoetry*.

IV. Models

We experimented with two multinomial models capable of classifying our reddit posts over 12 classes.

A. Multinomial Naive Bayes

Our first model was Multinomial Naive Bayes. Naive Bayes is a generative model which learns the class prior $p(c_k)$ for each $k \in K$ as well as $p(x_i|c_k)$, or the probability of the feature x_i conditioned on the class c_k. Given some new input x to evaluate, Multinomial Naive Bayes selects the class via

$$\arg\max_c p(c) \prod_{i=1}^{n} p(x_i|c)$$

B. Multinomial Logistic Regression

We also experimented with Multinomial Logistic Regression, a multi-class generalization of Logistic Regression. Just as in Logistic Regression, Multinomial Logistic Regression trains via Stochastic Gradient Descent, learning some parameters θ to minimize the cost function $J(\theta)$

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 + \frac{1}{C}\sum_{i=1}^{n}\theta_i^2$$

where

$$h_\theta(x)_j = \frac{\exp(\theta_j^T x)}{\sum_{k=1}^{K}\exp(\theta_k^T x)}$$

which is the softmax function.

V. Hyperparameter Tuning

In order to maximize our performance, we tuned our hyperparameters using 10-fold cross validation. We optimized our performance by evaluating on F1 (the harmonic mean between precision and recall) across all 12 classes. Recall, hyperparameters are parameters to our model $\notin \theta$ and therefore not associated with the optimization objective. These parameters must be optimized for using other methods, such as grid search. For our models, these parameters included n, the number of features, C, the regularization parameter, and k, the number of topics inferred by Latent Dirichlet Allocation. By tuning these parameters, we were able to find large increases in the performance of our overall system.

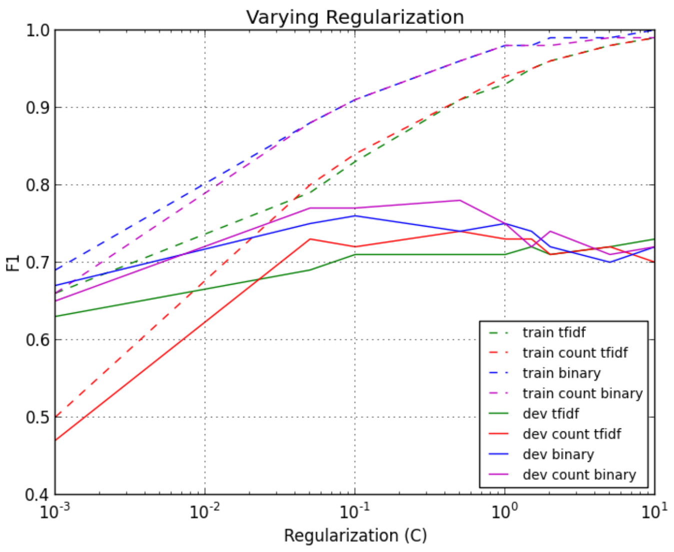

Fig. 5. Tuning the hyperparameter C, the inverse regularization parameter

VI. Results

Classifier	Train P	Train R	Train F1	Dev P	Dev R	Dev F1
Baseline - NB	.76	.69	.66	.60	.50	.46
Baseline - LR	.98	.98	.98	.67	.66	.66
LDA+TF-IDF - LR	.82	.82	.82	.70	.70	.70
Sentiment+Binary - LR	.97	.97	.97	.73	.72	.72
Count+Binary - LR	.95	.94	.94	.75	.75	.75

Performance of different systems on the development set

We are very pleased with the end results of our system, both on the development set and on the held-out test set. In our first table, you can see the performance of a subset of our systems on both the train and development sets.

In the end, we chose our best system to be a Multinomial Logistic Regression model using 'Title-Split', word count, and 3000 unigram Mutual Information selected binary valued features. This system was chosen as it consistently performed the best on the development set. As such, we evaluated this system on a held-out test set (obtained by scraping reddit) consisting of 35 posts from each of the 12 subreddits, with results documented in Fig 6 and our second table.

A large source of our errors in both dev and test came from trying to differentiate between *AskMen* and *AskWomen*. Often times when adding different features, such as sentiment or Parts-Of-Speech tags, the classifier was better able to differentiate between these two, but it still did not do very well. The reasoning behind this is that the two categories are inherently very similar. They have almost exactly the same average length (658 vs 665) and have have an overlap of seven words in their most frequent ten. Removing the *AskMen* subreddit gave us an astounding increase to .80 F1 for average dev accuracy.

It is also worth noting that two classes, class 4 *r/confession* and class 7 *r/self*, performed much worse on the test set than previously seen in dev. This may be due to the fact that our train and dev set were top posts of all time for the subreddit while our test set was top 35 of a week, or that these subreddits happen to inherently have high variability in the posts.

subreddit	P	R	F1
NoStupidQuestions	.62	.69	.65
shortscarystories	.80	.80	.80
Showerthoughts	.70	.94	.80
DebateReligion	.79	.66	.72
confession	.54	.57	.56
relationship_advice	.71	.77	.74
UnsentLetters	.78	.71	.75
self	.36	.29	.32
askphilosophy	.71	.77	.74
ShittyPoetry	.67	.80	.73
AskMen	.64	.46	.53
AskWomen	.60	.51	.55
Overall	.66	.66	.66

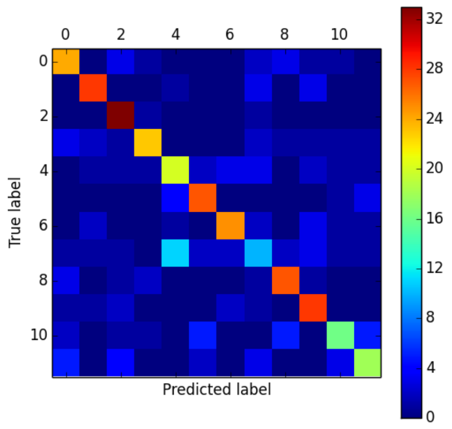

Fig. 6. Confusion Matrix for our best classifier (logistic regression with 'Title-Split', word count, and 3000 MI-selected binary features) on the held-out test set

VII. Future Work

Although our classifier performs well, there are a few improvement ideas we never had time to pursue.

One improvement which was discussed was attempting to use Latent Dirichlet Allocation as more than a tool to give us feature vectors. More concretely, we would like to use LDA to be our classifier. Since LDA is a generative model, it has probabilities for topics given a document and for words given a topic. It seems plausible that one could train K different LDA models, one for each class, and then in testing determine the class of a document d by a simple $\arg\max_c P(c|d)$ where $P(c|d)$ could be approximated from LDA. Unfortunately, we never found a tractable way to approximate this, and thus never was able to use LDA in more than a feature-space sense.

Besides looking at ways to improve our classifier, there are also ways of expanding our project. For one, we could explore how our classifier performs when classifying to a larger (> 12) number of subreddits. Similiarly, instead of limiting ourselves to text posts, we could try classfying link posts by following the link and scraping text and other data that could be used in a classifier to discern the subreddit. Both of these expansions are part of a general goal of ours. We believe that our classifier would be a useful tool on reddit and are interested in scaling this project to a level at which it could be actually used by reddit.

References

[1] Li, Lei, and Yimeng Zhang. "An empirical study of text classification using Latent Dirichlet Allocation."

[2] Novoviov, Jana, Antonn Malk, and Pavel Pudil. "Feature selection using improved mutual information for text classification." Structural, syntactic, and statistical pattern recognition. Springer Berlin Heidelberg, 2004. 1010-1017.

[3] Fuka, Karel, and Rudolf Hanka. "Feature set reduction for document classification problems." IJCAI-01 Workshop: Text Learning: Beyond Supervision. 2001.