Mood Detection with Tweets

Wen Zhang[1], Geng Zhao[2] and Chenye (Charlie) Zhu[3]

[1]Stanford University, zhangwen@stanford.edu
[2]Stanford University, gengz@stanford.edu
[3]Stanford University, chenye@stanford.edu

December 12, 2014

Abstract

In our project, we applied naive Bayes and SVM models to classify an arbitrary Tweet message into positive and negative mood category. A properly-optimized SVM model with linear kernel yields satisfactory results, though the learning curve suggested that it suffered from substantial overfitting. Additionally, we have shown that all features are important for classification and thus shall be neglected in the problem.

I. Introduction

Twitter is an immensely popular social network with more than two hundred million users worldwide with hundreds of millions of Tweets posted every day. A great fraction of them are highly personal, and are expressive of the users' emotions. In this project, we use machine learning algorithms to predict the mood of a Tweet – happy or sad. The ability of mood detection has numerous real-life applications. For example, advertisers may personalize ads based on the sentiment of a user towards certain products.

II. Dataset

The training and test data used in this project were obtained from *Sentiment140*[1], containing 160,000 Tweets, evenly divied between positive and negative sentiments. The Tweets were scraped from the Twitter website, and were properly labelled according to the emoticons contained: Tweets that include emoticons such as :) were perceived as positive examples, whereas those with symbols like :(were interpreted as negative examples. Ambiguous ones were removed from the dataset. Furthermore, the emoticons, strong indicators themselves, were stripped off after the labelling, as not doing so would introduce a strong bias into the model [1].

For the purpose of this project, we randomly selected 123,480 Tweets from the database, 70% of the which (86,436 Tweets) were used for training and cross-validation for paremeters, while the remaining 30% (37,044 Tweets) were reserved for testing.

III. Features and preprocessing

A natural way to approach text classification problems is to use tokens as features. During preprocessing, we removed certain irrelevant tokens: URLs, mentions (@username) and retweets (RT @username). Then we counted the total number of occurrences for each token in the training examples. We obtained a total number of 48,235 tokens for single words.

To take into account phrases and expressions such as "not bad", we also added in two-grams and three-grams (i.e. two/three adjacent tokens in the text) as features. This enabled us to extract more information from the text, and also helped mitigate underfitting by increasing feature dimension. The total count of tokens is 426,256 for $\leq$2-grams, and 1,096,433 for $\leq$3-grams.

We then transformed the count matrix into a normalized tf (term-frequency) or tf-idf (term-frequency times inverse document-frequency) representation to account for varying contribution for classification of each token. The tf of a token t in a document d (i.e. a Tweet) is the total occurrences of t in d:

$$\text{tf}(t,d) = \sum_{t' \in d} 1\{t' = t\}.$$

[1]http://www.sentiment140.com/

The idf of a token t in a collection of documents D is inversely related with the number of documents in which token t appears:

$$\text{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}.$$

For a token that appears in most Tweets, such as "the", it is less likely to contribute much to the classification, and it will also have a low idf value. Therefore, idf measures how informative the token is in the whole collection of documents. tf-idf is product of tf and idf:

$$\text{tfidf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D).$$

tf-idf values weigh the occurrences of a token by how meaningful it is in the classification.

Thus, for a Tweet d in the collection of Tweets D, its corresponding training example would be

$$x^{(d)} = [\text{tfidf}(t_0, d, D) \quad \text{tfidf}(t_1, d, D) \quad \ldots],$$

where $t_0, t_1, \ldots$ are the tokens. We may replace $\text{tfidf}(t, d, D)$ with $\text{tf}(t, d)$ if we decide not to use idf.

IV. Learning Models

I. Naive Bayes

First, we used a naive Bayes classifier from the scikit-learn package [4] as a baseline. Simple and fast to train, this model tends to yield acceptable results for text classification problems. It classifies a new input $\tilde{x}$ with features $(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ into class y^*, where

$$y^* = \arg\max_c p(y) \prod_{i=1}^{n} p(\widetilde{x}_i | y).$$

II. Support Vector Machine

We futher experimented with soft-margin support vector machines (SVM) with linear kernels, using the implementation from scikit-learn [4]. The SVM algorithm finds a separating hyperplane with maximal margin:

$$\begin{aligned} \min_{\gamma, w, b} \ & \frac{1}{2}||w||^2 + C \sum_{i=1}^{m} \xi_i \\ \text{s.t. } & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i \\ & \xi_i \geq 0, i = 1, \ldots, m. \end{aligned}$$

SVMs are a desirable model for text classification as the built-in regularization mechanism mitigates the potential overfitting, a prevelant phenomenon in text classification problems with very high dimentional input feature space [2]. Furthermore, data points in these problems are typically linearly separable [2], and hence training an SVM with a linear kernel is remarkably fast using the SMO algorithm.

V. Experimental Results and Analysis

I. Parameter Selection

There are multiple decisions involved in parameter settings, for instance the choice whether to use plain tf or tf-idf, and what value is appropriate for C for the objective function in the SVM model. We used cross-validation to determine the values of parameters: hold back 30% of the traning data into a cross-validation set; train the learning models using different parameters on the remaining 70%; and choose the value that yields lowest cross-validation error.

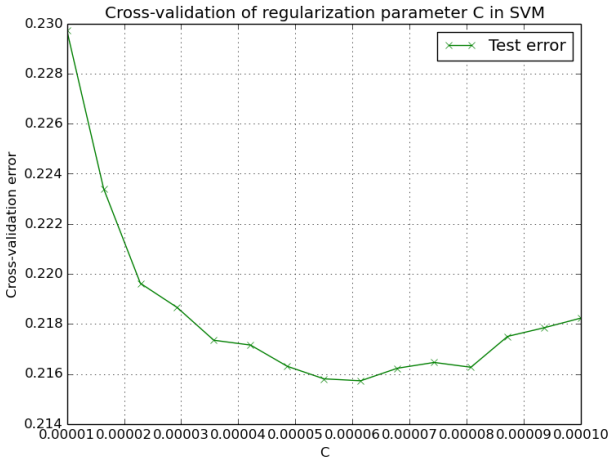

Figure 1: *Cross-validation errors on different choices of parameter C in an SVM*

	Naive Bayes	SVM
tf-idf	21.75%	20.30%
tf	21.50%	21.58%

Figure 2: *Cross-validation errors when we use / do not use idf in feature selection with 1- and 2-grams*

The figures above represent two sample cross-validation processes we run to attain the suitable parameters. For instance, the graph on the left reports cross-validation errors of SVMs with unigram features, and regularization parameter C ranging from 10^{-5} to 10^{-4}. We see that setting $C = 6 \times 10^{-5}$ yields the most satisfactory result. The table on the right shows cross-validation errors when we choose to use tf or tf-idf with 2-gram features.The SVM model gives slightly better predictions when tf-idf is used, while the performance of naive Bayes model is essentially unresponsive to the change.

II. Error Rates

After various experiments and trials with different combinations of learning models, parameter values and feature selections, here we report the best results.

Features	Naive Bayes with tf features		SVM with tf-idf features	
	Training error	Test error	Training error	Test error
1-grams	0.2020	0.2258	0.1960	0.2176
1- and 2-grams	0.1496	0.2155	0.0487^2	0.2030
1-, 2-, and 3-grams	0.1490	0.2154	0.0230	0.2017

Table 1: *Training errors of naive Bayes and SVM with different feature dimensions*

From the data we obtained, a linear SVM with $\leq$3-gram features yields the best testing accuracy (79.83%). Furthermore, when we trained our best model on a larger data set with 100,436 training examples, the largest we experimented with, it reported an even higher accuracy of 80.14%.

Additionally, SVM models perform slightly better than their naive Bayes counterparts, and increasing feature size (while adjusting parameters accordingly) helps reduce error rates.

III. Algorithm Performance

We have plotted learning curves for all six models. Here we present two examples.

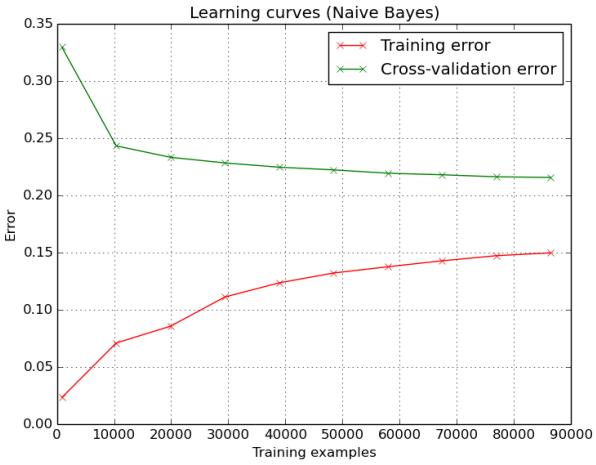

Figure 3: *Learning curve for naive Bayes (≤2-grams)*

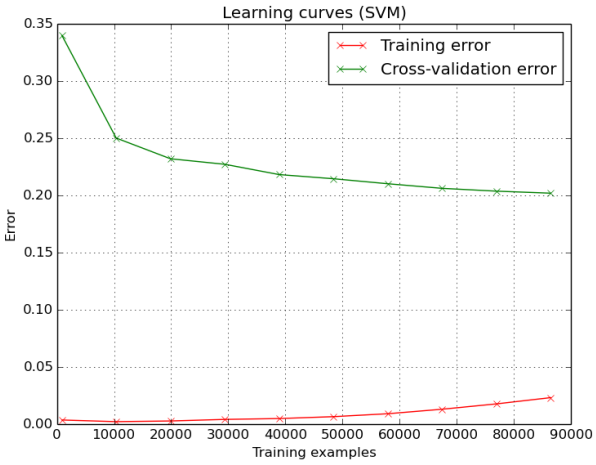

Figure 4: *Learning curve for SVM (≤3-grams)*

Figure 3 on the left represents the learning curve for the naive Bayes model with 1- and 2-gram tf features. There is no clear indication whether overfitting or underfitting results in the generalization error. [3]

Figure 4 on the right shows the learning curve for the SVM model with 1-, 2-, and 3-gram tf-idf features. Clearly the learning algorithm suffers from substantial overfitting, with notable discrepancy between training errors and test errors. We further noticed that the training errors had not stablized yet, suggesting that adding more training examples might raise the training error.

VI. Diagnostics

To alleviate overfitting problems in the SVM model, we attempted to reduce the feature dimensions by selecting the most relevant features. Specifically, the relevance of a token mainly depends on:

- Frequency in training set: tokens that appear more frequently in the training set are more likely to appear in new Tweets, and thus are more relevant for the classification problem.
- Ratio of frequencies in the two categories: tokens that appear frequently in Tweets of one category but rarely in those of the other are more indicative of the class the Tweet belongs to.

We formalized our reasoning by building the following heuristic and determining the relevance of each token:

$$h(x) = \frac{\max(A_x, B_x) + 1}{min(A_x, B_x) + 1} + \alpha \frac{A_x + B_x}{\sum_y (A_y + B_y)},$$

where A_x and B_x denote the frequency of token x in the positive and negative category respectively. The parameter α represents the weights of two factors.

We selected the k features with the highest heuristic scores for different values of k and adjusted the values of α accordingly. We then trained the model on the selected features.

To prove that our heuristic is reasonable, we set $\alpha = 100000$ and compared the behavior of an SVM trained on k features selected according to the heuristic and k random features from 1- and 2-gram tokens. The reported cross-validation errors are as follows:

	Heuristically-selected features	Random features
$k = 5,000$	23.03%	46.60%
$k = 10,000$	22.42%	45.78%
$k = 50,000$	22.10%	36.14%

Figure 5: *Comparison of errors obtained by choosing k features according to the heuristic vs. randomly*

[3] For computational reasons, we reduce the feature dimention to 130,000 based on their occurances in all training examples.

Choosing features according to the heuristic yields much better results than choosing randomly. This verifies that our heuristic is valid.

Below is a table of testing errors of an SVM on the selected features from 1- and 2-gram tokens (out of 426256 features in total).

	5,000	10,000	50,000	100,000
1,000	0.2801	0.2607	0.2243	0.2224
10,000	0.2483	0.2395	0.2213	0.2200
100,000	0.2284	0.2224	0.2203	0.2184

Figure 6: *Test errors obtained by adjusting the values of k (columns) and α (rows)*

Firstly, notice that the greater the value of α is, the better results we obtain. When α dominates, $h(x)$ is approximately equal to term frequency divided by the length of the corpus. Thus, the ratio between the number of occurences of each token in the two classes is not particularly indicative of the relevance of each token.

More importantly, we found that reducing the number of features only increases test error, and that the more features we kept, the lower test error we got. This implies that most tokens are informative in this text classification problem, and selecting features may lead to a loss of information, validating the findings by Joachims [2].

VII. Conclusion and Future Work

We applied naive Bayes and SVM models to classify Tweets into positive and negative sentiment category. After setting the proper parameters for each model through cross-validation, we found that the SVM model with linear kernel yielded the best test result, and yet suffered from significant overfitting. In our attemp to reduce feature dimension and select most "relavent" tokens, we found that most features are important for the classification and should not be omitted.

In the future, we will consider more sophisticated feature representations and dataset preprocess to improve the performance of our method. Moreover, resorting to more advanced learning algorithms, such as neural networks for automatic feature extraction, and NPL models, such as LDA, for better approaches to solve the problem.

References

[1] Go, A., Bhayani, R., & Huang L. (2009). *Twitter Sentiment Classification Using Distant Supervision*. Retrieved December 6, 2014, from `http://cs.stanford.edu/people/alecmgo/papers/TwitterDistantSupervision09.pdf`

[2] Joachims, T. (1998). Text categorization with support vector machines: learning with many relevant features. *European Conference on Machine Learning (ECML) 1998*.

[3] Manning, C. D., Raghavan, P., & SchÃijtze, H. (2008). Support Vector Machines and Machine Learning on Documents.*Introduction to Information Retrieval*. (pp. 319-347). Cambridge University Press.

[4] scikit-learn developers. *Working With Text Data*. Retrieved December 2014, 5, from `http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html`