Sentiment analysis of online reviews

Charles Stahl
Department of Physics
Princeton University
cnstahl@

Meir Hirsch
Department of Computer Science
Princeton University
ehirsch@

Abstract

With the proliferation of short text snippets on the Internet, especially in social networks, there is a growing need to be able to classify these texts with very little context. This can be as simple as classifying snippets as positive or negative. One place where this would be helpful is in online reviews, so that sellers can gauge the reaction through a metric separate from number-of-stars reviews. On social media, one might want to look at all mentions of a person or hashtag and see whether general sentiment is positive or negative.

1 Introduction

With the proliferation of short text snippets on the Internet, especially in social networks, there is a growing need to be able to classify these texts with very little context. This can be as simple as classifying snippets as positive or negative. One place where this would be helpful is in online reviews, so that sellers can gauge the reaction through a metric separate from number-of-stars reviews. On social media, one might want to look at all mentions of a person or hashtag and see whether general sentiment is positive or negative.

In this paper, we evaluate **some methods** in their ability to classify the sentiments of online reviews as either positive or negative. We evaluate from the perspective of looking for negative reviews or looking for positive reviews. We evaluate the methods using a simple bag-of-words representation with and without feature selection.

Such a sentiment classifier could conceivably be useful to producers and consumers. Producers would want to find the negative reviews in order to know what they most need to improve in their products. Consumers could use it to find the products with the most positive reviews.

2 Related Work

The data set consists of 2400 training sentences taken from Amazon, Yelp, and iMDB, preprocessed into separate lines and labeled with sentiment. There are also 600 labeled training sentences formatted the same way. The dataset was created by Kotzias et. al. [3], who analyzed the data using a sentence-level classification method. The optimal accuracy reported in their paper is between 86.0% and 88.2%. Preprocessing of the dataset on our end was done in part using the NLTK Python Libraries [1].

The naïve attempts in our paper come from the discussion on text classification in Kevin Murphey's *Machine Learning: A Probabilistic Perspective* [4]. Specifically, he first suggests using a Bernoulli naïve Bayes, then improving upon that with mulinomial naïve Bayes (MNB) model. We took the MNB as our starting point.

3 Methods

Generative vs. Discriminative Generative classifiers p(x,z) = p(z)p(x—z) model feature distribution of each class O(K) p-dim distributions to estimate good performance with small training sets accuracy stalls as training set grows robust to missing data

Discriminative classifiers p(x,z) = p(z—x)p(x) model distribution of classes in feature space O(p) K-dim distributions to estimate poor performance with small training sets accuracy improves for large training sets missing data strategies need to be explicit

3.1 Classification Methods

We used the following classification methods

a) Multinomial Naïve Bayes

All methods we used were implemented using the SciKit Learn Python libraries [6].

3.2 Feature Extraction and Selection

Feature extraction was performed during the training portion of the project. One feature used was categorical proportional difference [5], defined in terms of positive document frequency (pdf) and negative document frequency (ndf)

$$\text{CPD} = \frac{|pdf - ndf|}{pdf + ndf}, \tag{1}$$

where the denominator is equivalent to the total number of occurrences of the term in the whole document. Thresholds for both total document frequency and CPD score were set using 5-fold cross-validation.

3.3 Spotlight on Multinomial Naïve Bayes

The naïve Bayes algorithm rests on the assumption of class-conditional independence. The graphical model corresponding to this assumption with n samples and K parameters can be seen in figure 1. This figure is taken from a slide in Barbara Engelhardt's lecture on classification [2].

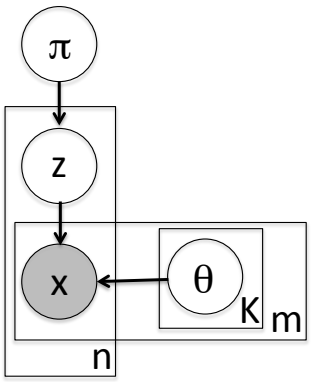

Figure 1: The graphical model corresponding to the assumptions of Naïve Bayes. The top circle represents the proportion of positive sentiments. K is the number of classes, each with its own parameters, while n is the number of samples. In our case, $\mathbf{x}$ is a vector of m features.

In the model, π is the proportion of samples with positive sentiment. In this case $\pi = .5$. The variable Z is the label of a given sample, i.e. whether the sample is positive or negative. $\mathbf{x}_i$ is the observed feature vector. When we only use the bag-of-words representation, the features are just the counts of each word. However, when using feature selection, $\mathbf{x}$ would include those features we selected.

The letter n indicates the number of samples observed. K is the number of classes. Θ lies inside the K plate because each class has its own probabilities. The last letter, m, corresponds to number of features observed.

The class-conditional independence assumption can be seen in the graph. Once z_i is observed, all components of $\mathbf{x_i}$ are independent. Dropping the subscript i corresponding to a single observation and now subscripting w.r.t. components of the feature vector, this can be written as

$$P(x_i|z, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_m) = P(x_i|z). \tag{2}$$

This simplifies $P(z|\mathbf{x})$ as follows:

$$\begin{aligned} P(z|\mathbf{x}) &= \frac{P(z)P(\mathbf{x}|z)}{P(\mathbf{x})} \\ &\propto P(z)\sum_{i=1}^{m} P(x_i|z), \end{aligned} \tag{3}$$

using the fact that $P(\mathbf{x})$ is a constant.

The values for $P(x_i|z)$ are learned from the training data. In multinomial naïve Bayes, $P(x_i|z)$ is just the number of times x_i appears in class z. This can be a problem if a word doesn't appear in a class. This can be dealt with by smoothing. The scikit-learn implementation of MNB uses Laplace smoothing, which is equivalent to adding a pseudocount to each feature.

Once the model is trained, prediction is usually carried out using the maximum a posteriori (MAP) estimate, meaning the predicted class is just the class with the highest probability. Note that

$$\sum_z P(z)\sum_{i=1}^{m} P(x_i|z) = \sum_z P(z|\mathbf{x})P(\mathbf{x}) = P(\mathbf{x}). \tag{4}$$

This means that using MAP without multiplying by $P(\mathbf{x})$ (as was done to derive equation 3) is equivalent to using a cutoff likelihood of .5.

4 Results

For all methods, hyperparameters were fit using 5-fold cross validation. With optimal hyperparameters, the models were then trained on the 2400 training data and tested on the 600 testing data. The results can be seen in table 1.

4.1 Evaluation Metrics

The most naïve method for evaluating classifiers would be to just calculate total correct/total data point. however, this is complicated by any asymmetric relationship between the two classes. The canonical example is in interpreting cancer screening results. A false positive test is problematic and could lead to painful and expensive treatment with no benefit. However, one could argue that a false negative leads to an even worse outcome.

Two important quantities in looking for positive results are precision and recall. These are defined in order to specifically look for positive results, fixing the problem above. Denoting true positives as PN and false negatives as NF, etc., these are defines as

$$\text{precision} = \frac{\text{TP}}{\text{TP + FP}}, \quad \text{recall} = \frac{\text{TP}}{\text{TP + FN}}. \tag{5}$$

Intuitively, this means that precision is, the probability that a positive result will be a true positive, while recall is the probability that a positive case will be identified as positive.

Another quantity is the F_1 score, defined as

$$F_1 \equiv 2\frac{\text{recall} \times \text{precision}}{\text{recall + precision}} = \frac{\text{2TP}}{\text{2TP + FP + FN}} \tag{6}$$

F_1, precision, and recall all range from 1 (perfect) to 0, with random guessing on balanced data giving .5 for all three.

For online reviews, manufacturers may often be more interested in finding the negative results. If the sought-after class is called "positive," this leads to the awkward position of calling negative reviews positive results. Therefore we will stick to calling them negative, and clarify the sought-after class when necessary. Customers may however want to find positive reviews. In order to make our results useful to both groups we report the above quantities and their analogues when searching for negative reviews.

The analogue of recall is specificity, TN/(TN+FP). The analogue for precision doesn't have a name, but is calculated as TN/(TN+FN). We call the analogue of the F_1 score the F_1' score,

$$F_1' = \frac{2\text{TN}}{2\text{TN} + \text{FN} + \text{FP}} \tag{7}$$

		Looking for Positives			Looking for Negatives			
Classifier	Features	Precision	Recall	F_1	TN/(TN+FN)	Specificity	F_1'	Time (s)
MNB	BoW	pre	rec	F_1	TN/(TN+FN)	Specificity	F_1'	Time (s)
MNB	Select	pre	rec	F_1	TN/(TN+FN)	Specificity	F_1'	Time (s)

Table 1: **Results from however many classifiers on sentiment analysis data.** We list values for however many classifiers, each with and without using feature selection. We also divide results into those useful when looking for positive and negative sentiment. Lastly, we state runtimes for each classifier.

The reported values are those of the classifier with the hyperparameters set to maximize accuracy on the training data set. That means the reported value of recall for MNB, for example, is not the highest achievable precision. In fact, recall $= 1$ is achievable by marking all test cases as positive. A receiver operating characteristic (ROC) curve captures the tradeoff between true positives (recall) and true negatives. It varies the threshold for predicting a positive sentiment and then plots both TPR and TNR. An ROC for a perfect classifier would be a step function. For an example ROC curve see figure 2.

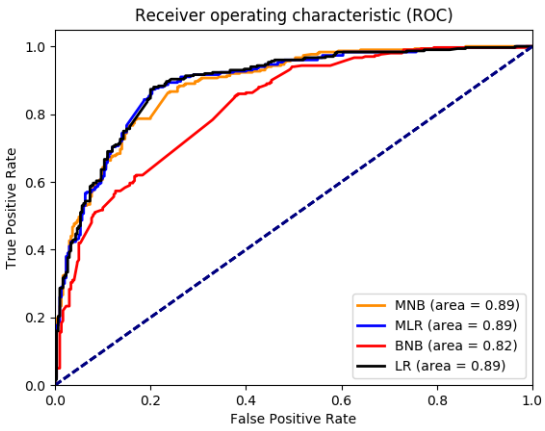

Figure 2: ROC curves for multinomial naíve Bayes, multinomial logistic regression, Bernoulli naïve Bayes, and logistic regression. BNB can be seen to be worse across all thresholds.

5 Discussion and Conclusion

low threshold for df, especially

high threshold for

roc shows that logistic regression overfits less

Acknowledgments

References

[1] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, Inc., 1st edition, 2009.

[2] Barbara Engelhardt. Probabilistic classification. Lecture Slides.

[3] Dimitrios Kotzias, Misha Denil, Nando de Freitas, and Padhraic Smyth. From group to individual labels using deep features. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD 15*. Association for Computing Machinery (ACM), 2015.

[4] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective (Adaptive Computation and Machine Learning series)*. The MIT Press, 2012.

[5] Tim O'Keefe and Irena Koprinska. Feature selection and weighting methods in sentiment analysis. 2009.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.