Homework 1: Sentiment Analysis

Joshua Wallace
Department of Astrophysical Sciences
Princeton University
joshuajw@princeton.edu

Abstract

1 Introduction

Classification is important in all fields of science. Some examples specific to astrophysics: classifying celestial objects based simply on the emission of light from the object, and often do so in an approximate way without having detailed observations of defining characteristics; classifying whether a transient dip in a star's brightness is due to a planet or another star crossing between us and the star; and classifying galaxies based on images. Less grandiose but more useful, in this work I try to understand useful classifiers and parameters for sentiment classification of a given set of reviews. Information from sentiment analysis of reviews can give companies valueable insights as to which of their products are most successful, as well as allow them to identify bad products and situations and focus resources towards mitigating fallout from the situation producing the negative reviews.

In this work, I investigate various classifiers and associated tools (e.g., feature selection) to figure out the best and most accurate classifiers. Specifically, I look at X classifiers in this way and that way. Bag-of-words. Maybe speed. Feature selection.

2 Related Work

This work makes use of the `nltk` [1] and `sklearn` [2] Python packages. The former was primarily used to tokenize the data and remove stop words while the latter was the main source of the classifiers used in this work.

3 Methods

3.1 The Data and Preprocessing

The data consisted of 3,000 online reviews, of which 2,400 were taken as a training set and 600 were taken as a test set. These reviews were given without explanation of source, but a personal perusal of the reviews revealed that they cover a variety of reviewable things, from cell phones to movies to restaurants. The data were preprocessed using a code kindly provided by the professors. This code tokenized the reviews as well as removed stop words. It then determined a vocabulary list based on words that had a frequency across the entire training set above a certain threshold. I tried different values for for this threshold, between 3 and 6 inclusive, to determine a best value to use. The code then converted each review into a bag-of-words representation based on the vocabulary list. The length of the vocabularly list for the threshold values of 3, 4, 5, and 6 are respectively 812, 640, 541, and 462. These are the initial lengths of the feature vectors; feature selection will shorten these.

3.2 The Real Meat of this Work: Classifying

3.3 Naïve Bayes in Detail

Naïve Bayes may be a simple classifier, but to me it is special because it was the first classifier I formally learned. Before that, machine learning was nothing but magic to me, but after I learned Naïve Bayes I saw how formal, understandable, and simple machine learning was. Naïve Bayes is named such because it makes use of Bayes' theorem and also makes the "naïve" assumption that all the features are independent. For the specific application of sentiment analysis, counter-examples of the independence assumption are easily thought of: the existence of the word "great" in a review can have a very different classification interpretation depending on whether the word "not" occurs before it. Despite this, Naïve Bayes still enjoys prominence as a classifier because of its simplicity.

4 Results

4.1 Naive Bayes

I first used the Naive Bayes classifier from the `nltk` package [1]. I tried different minimum number of words to be counted as vocab. For 3 I got training: 86.4% accurate and test 78.8% accurate, for 4 I got training: 85% accurate and test 78.3% accurate, for 5 I got training: 84.1% accurate and test: 77% accurate, and for 6 82.8% accurate for training and 77.3% accurate for training.

I also tried to do a little bit of "feature selection" on my own. I identified some words I thought would have no impact on the sentiment of a review such as brand names (e.g., "samsung" and "bluetooth") and food items (e.g., "potato" and "taco"). I then ran the `nltk` Naive Bayes analysis on these. I chose these words because it seemed to me that these words were not positive or negative in themselves but rather were focused on the product being reviewed itself. This had a marginal effect on the percentages: some went up 0.1–0.2%, some went down 0.1–0.2%, some remain unchanged. This version of feature selection did not seem to offer any improvement (especially since it made some accuracies higher and some lower), and our feature set isn't so large as making it smaller by a few words to speed things up, so I decided to not use this "feature selection".

Implementing the `sklearn` [3] Gaussian Naive Bayes had worse results overall: for 3, 81.8% training and 71.7% test; for 4, 79.3% training and 71.8% test; for 5, 77.9% training and 70.5% test; for 6, training 76.5% and test 69.7% accurate.

Bernoulli and Multinomial do much better. Almost up to 80%!

Number of words to be included as vocab is: 3

GaussNB Training percentage for 3 : 0.818333333333 Gauss NB Test percentage for 3 : 0.716666666667

BernNB Training percentage for 3 : 0.863333333333 BernNB Test percentage for 3 : 0.793333333333

MultNB Training percentage for 3 : 0.862916666667 MultNB Test percentage for 3 : 0.791666666667 Number of words to be included as vocab is: 4

GaussNB Training percentage for 4 : 0.792916666667 Gauss NB Test percentage for 4 : 0.718333333333

BernNB Training percentage for 4 : 0.849583333333 BernNB Test percentage for 4 : 0.79

MultNB Training percentage for 4 : 0.852083333333 MultNB Test percentage for 4 : 0.786666666667 Number of words to be included as vocab is: 5

GaussNB Training percentage for 5 : 0.779166666667 Gauss NB Test percentage for 5 : 0.705

BernNB Training percentage for 5 : 0.840416666667 BernNB Test percentage for 5 : 0.773333333333

MultNB Training percentage for 5 : 0.8425 MultNB Test percentage for 5 : 0.77 Number of words to be included as vocab is: 6

GaussNB Training percentage for 6 : 0.765416666667 Gauss NB Test percentage for 6 : 0.696666666667

BernNB Training percentage for 6 : 0.828333333333 BernNB Test percentage for 6 : 0.773333333333

MultNB Training percentage for 6 : 0.828333333333 MultNB Test percentage for 6 : 0.766666666667

VarianceThreshold is a bust

Slightly better SelectKBest. Around k=310, 80% accuracy. However, fairly insensitive to value of k, and indeed there are several "local maxima", very small maxima though relative to the average. Accuracy starts to fall off for k¡=100 or so. This is chi2.

Using funderscoreclassif, also get 80% accuracy around k=310, but accuracy doesn't fall off so much. Also several "local maxima".

mutualunderscoreinfounderscoreclassif takes a much longer time, much longer! And lower accuracy.

5 Discussion and Conclusion

Acknowledgments

This work greatly benefitted from not only the guidance of the professors and instructors but particularly benefitted from the preprocessing code `preprocessSentences.py` provided by the professors.

References

[1] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python*. "O'Reilly Media, Inc.", 2009.

[2] 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.

[3] Jun Zhu, Amr Ahmed, and Eric P Xing. Medlda: maximum margin supervised topic models for regression and classification. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1257–1264. ACM, 2009.