Sentiment Classification of Product and Movie Reviews

Rebecca Johnson
Princeton University
raj2@princeton.edu

Diana Stanescu
Princeton University
stanescu@princeton.edu

Abstract

In the present project, we classify the sentiment of reviews into two classes–positive; negative–to help a client aggregate over a product's reviews and place positively-reviewed products higher in search results and negatively-reviewed products lower. Using a training data set of 2400 reviews and a held-out test dataset of 600 reviews, we compare the accuracy of two sets of representations of the feature: unigram bag of words and bigrams. We find the best performing methods to be ones like logistic regression with an L_1 penalty that incorporate a penalty term to push the parameters on non-meaningful terms to zero. In extensions, we find that the unigram representation, despite its stronger assumptions, outperforms the bigram representation, and that a simple ensemble method that averages across classifiers by taking the majority classification ("voting classifier") has good performance. We discuss next steps involving modifying the stopword dictionary.

Introduction

Who might be interested in classifying the sentiment of reviews (positive versus negative), which accuracy metrics might they care about, and how can we develop a model that performs well on those metrics? The best sentiment classification method for the reviews depends on one's purpose for labeling the documents. Suppose our task is *aggregating across documents to give a single product a rank*. A product search engine like Amazon wants to use reviews to position products in search results–for instance, when customer searches for "machine learning textbook," the engine wants positively-reviewed products–defined using some summary measuring across the product's reviews–to appear higher in the search results. It wants negatively-reviewed products to appear lower. Customers might be more upset by false positives than false negatives–that is, they might be more upset if they click on products that are actually negatively-reviewed but appear high in the search results due to a false positive-review label (false positive) than about "missing out" on products that were actually positively-reviewed but classified as negative (false negative).

Step zero: text processing

We used the pre-processing script provided with the assignment to obtain a bag-of-words representation of the corpus with a Feature Frequency specification. We (1) excluded stopwords; (2) stemmed/lemmatized the words; and (3) filtered out all words occurring five or fewer times in the corpus. This resulted in a 2400 (documents) x 541 (features) training and a 600 x 541 testing matrix. These matrices encode several assumptions, one of which (the unigram structure of meaning–"totally awesome" in one review gets separated into "totally" and "awesome," while "totally awful" in another gets separated into "totally" and "awful"; the two documents each have a count for "totally) we relax in the extension.

1 Step one: feature selection prior to fitting classification models

While some of the classification methods we discuss in Table 1 perform feature selection via the use of a penalty function, we perform feature selection as an additional step in pre-processing prior to model fitting. We took a sequential approach.

1.1 Approach one: manual feature selection

First was more manual selection methods, with the general goal of finding words over-represented in positive reviews relative to negative reviews, or vice versa. Categorical proportion difference (CPD) is one type of non model-based feature selection that captures differences in features between labels [3]. If pos indicates a positively-labeled document, neg indicates a negatively-labeled document, p indicates a feature, values closer to 1 indicate *more* imbalance in the features, and values closer to 0 indicate *less* imbalance: $CPD = abs(p_{pos} - p_{neg}) \div (p_{pos} + p_{neg})$. While we performed CPD on our sample, the result would have reduced our feature set to only 46 features even with a low threshold where we remove a term if the $CPD < 0.2$. As a result, we switched to the following simple metric to perform manual feature selection: diff $= abs(p_{pos} - p_{neg})$. If diff > 2, we retained the feature. This reduced our feature set to 355 words.

1.2 Approach two: two forms of model-based feature selection

We further reduce the feature set by taking the 355 features from approach one and performing further feature selection via two models. In each, we fit a model predicting the class label that explicitly incorporates some sort of penalty to move the coefficients on particular terms towards zero by introducing a constraint on the sum of the coefficients across the p predictors. For instance, if the terms "bought" and "awful" are each in p, "bought" may poorly predict the class label and $\beta_{bought} \rightarrow 0$, while "awful" may more strongly predict a class label and $\beta_{awful} \neq 0$. We then use a threshold that removes p where β_p falls belows that threshold.

1.2.1 Cross-validated Lasso for feature selection

Lasso is a linear model that contains a penalty term where the coefficients in the model not only aim to minimize prediction error (least squares) but also are constrained to sum to the lowest absolute value. Put more formally, where β represents a coefficient on a term, j indexes a coefficient, n represents the number of reviews, i indexes a review, p indicates a vector of term predictors, lasso aims to solve the following constrained minimization problem [7]:

$$min \frac{1}{n} \sum_{i=1}^{N} (\text{label}_i - \beta_0 - p_i^T \beta)^2$$

Subject to: $\sum_{j=1}^{p} |\beta_j| < \alpha$. Cross-validation–fitting the model on a portion of the data and using the coefficients to predict in a test portion–is used to select a regularization parameter. We then remove terms where $\beta_j < \frac{1}{p} \sum_{j=1}^{p} \beta_j$. This resulted in 273 features.

1.2.2 Logistic feature selection

Lasso is not specifically intended for cases where the outcome variable is binary, so we also performed feature selection with a logistic regression intended for binary outcomes. Within `sklearn`, we used the "liblinear" optimization method that works well in smaller datasets such as ours, an L_1 penalty that we discuss in greater detail below for our spotlight classifier, and cross-validation to tune the strength of regularization.

2 Step two: fit classification models on training data for the 3 feature spaces

Table 1 summarizes the classification models we used, along with different choices over specifications. We used [5] to implement all classifiers. Each choice is important for sentiment classification. First, *cross-validation*, which further splits the training sample into separate folds used to test and evaluate the model, is important to prevent over-fitting of the model to the particular set of 2400 reviews we have; since $p \rightarrow n$ in this context, over-fitting is a particular concern that we want to prevent. Second, a *penalty term* regularizes coefficients on some terms towards zero, and as such helps reduce the feature set to a few especially predictive features. Third, a *loss function* allows

incorporating information about misclassification of sentiment labels into the estimation phase. For example, the hinge loss function in SVM sums over incorrectly classified reviews ($\text{label}_i \neq \hat{\text{label}}_i$) and examines how the function that maps features p to labels scored each of these classes, with the "hinge" referring to the fact that when the scores of the two have the same sign, the loss term equals zero. Squared hinge squares the loss to penalize misclassifications more heavily [6].

Table 1: Classification models estimated on training data

Classifier	Cross-validation?	Penalty?	Loss function?
Decision tree	No	None	Default
Random forest	No	None	Default
AdaBoost	No	None	Default
Ridge	No	Default L_2 regularization	Default
Passive Aggressive	No	None	Squared hinge
Calibrated	Yes	None	Default
Logistic with default/liblinear optimization	No/Yes	L_1	Default
Logistic	No/Yes	L_2	Default
Support vector machine (SVM)	No	L_2/ L_1 / Elastic net	Hinge
Support vector machine (SVM)	No	L_2/ L_1 / Elastic net	Squared hinge

2.1 Spotlight classifier: logistic with L1 penalty chosen by cross-validation

Our spotlight classifier has three features: a logistic distribution used to model the outcome, an L_1 penalty used to penalize the coefficients on certain unigrams towards zero, and cross-validation used to select the parameter used in the penalty function. We discuss each in turn.

First, the logistic regression part of the classifier takes into account that we are mapping the inputs into a binary target of a positive versus negative sentiment label. It does so by using an inverse logit function that constrains the predicted values to lie between 0 and 1; as a result, even if a review has a set of highly weighted features, the model does not generate non-intuitive predictions like a review classified as having > 1 sentiment. More specifically, the inverse logit predicts the label as an inverse function of unigrams p weighted by β:

$$prob(label = 1|p; \beta) = \frac{1}{1 + exp(-\beta^T p)}$$

Second, the L_1 penalty aspect focuses on how we choose the vector of β_p (coefficient on each predictor). [4] discusses how incorporating this penalty into logistic regression is useful for cases where "there are many input features, but where there is a small subset of features that is sufficient to approximate the target concept well." As previewed in the feature selection section, the L_1 penalty selects this small subset by estimating the model (for the logit, via maximum likelihood estimation) subject to the constraint that $\sum_{j=1}^{p} |\beta_j| \leq \alpha$. Third, cross-validation helps us choose the parameters on the constraint.

The model provides two sets of interpretable results. First, we can identify reviews that are classified as positive or negative with the highest predicted probability. To do so, we fit the model on the best performing feature selection (lasso) and generated a length-2 vector for each test set sample i: $(\hat{y}_i = \text{negative}, \hat{y}_i = positive)$. We can then find $max(\hat{y}_i = \text{negative } \forall\ i)$ as an exemplar negative review in the test set:

> The acting sucks, the music sucks, the script sucks, the pacing sucks, the special FX suck, the directing sucks... basically, this movie sucks (Test set review #253)

Likewise, $max(\hat{y}_i = \text{positive } \forall\ i)$:

> All in all an excellent restaurant highlighted by great service, a unique menu, and a beautiful setting (Test set review # 584)

An ambiguous review with $\hat{y}_i =$ positive near 0.5 was: "Every time I eat here, I see caring teamwork to a professional degree."

Another interpretable output is, across documents/reviews, summarizing which terms have $\hat{\beta}_j \to 0$ and which terms have $\hat{\beta}_j \neq 0$, to identify informative terms. This might be useful for a business trying to use reviews to improve the product. Table 2 displays three sets of features, and shows how

the spotlight model provides us (somewhat) interpretable results–for instance, "bland" and "poor" predicting negative labels, "famili" predicting a positive label, and "product" not differentially producting either label.

Table 2: Selected features from spotlight model

Non-predictive	Highly predictive neg. label	Highly predictive pos. label
$abs(\hat{\beta}_j < 0.01)$	4 most negative $\hat{\beta}_j$	4 most positive $\hat{\beta}_j$
Ago	Bland	Appear
Back	Point	Famili
Product	Poor	Histori
Guess	Slow	Menu

3 Step three: evaluate predictive accuracy in test data

In the Introduction, we discussed how, depending on the purpose for sentiment classification, we might care about different metrics for model evaluation. Focusing on one example, we are pretending that we are a product search engine like Amazon trying to maximize consumer happiness. For each product, we: 1. classify the sentiment of each review; 2. average the sentiments, 3. assign an overall score to the product, 4. put products with more positive sentiment higher in the search results when a consumer searches for a given product category and products with lower sentiment lower in the results. Our main goal is to: *minimize false positives*– we want to avoid classifying a review/product that actually garners negative customer sentiment as a positive product ranked highly in the search terms. We care less about *false negatives* (inaccurately characterizing a positive sentiment product as negatively-reviewed, putting it lower in the search results), because we assume that customers will be more exasperated by clicking on and buying a product that is actually negatively reviewed than by missing out on a positively reviewed product because it's lower in the search queue. With this goal–minimizing *false positives*–in mind, how does each classifier perform, with this performance also differing across the three feature selection methods discussed in Section 2? We summarize the results using the following metrics:

1. *Confusion matrix:* this provides four measures. 1. True negatives $(neg_i, \hat{neg}_i)$. 2. False positives $(neg_i, \hat{pos}_i)$, 3. False negatives $(pos_i, \hat{neg}_i)$, and 4. True positives $(pos_i, \hat{pos}_i)$. As discussed, we care about minimizing false positives. Figure 1 highlights two findings. First, the model that provided the lowest false positive rate was the SVM with a squared hinge loss function and an elastic net penalty on the feature space selected with logit (see Section 1.2.2). However, this model does not perform as well on the other two feature spaces. The model with the most consistently low false positive rates across feature selection methods was the Adaptive Boosting method (AdaBoost), perhaps reflecting the fact that this method re-weights the training data when fitting the model to upweight observations misclassified in the previous iteration, which could make the predictions more robust to different feature selection choices.
2. Figure 1 highlights two other relevant measures of accuracy. The $F1$ score takes into account both the precision of the classifier and its accuracy, with higher values indicating a better score. The results highlight that adaptive boosting does especially well on this metric. The right panel summarizes the area under the receiving operating curve where the x axis is the true positive rate at different thresholds and the y axis is the false positive rate (ROC AUC). AUC closer to 1 depart more from the 45-degree line of random guesses, indicating a better score, and better separation of the reviews into the classes of positive and negative. The voting classifier we discuss in extension two below, and the logistic with L_1 penalty (with or without cross-validation) perform especially well on this metric.
3. *Model time*: due likely to the small size of the training set and performance of feature selection, all models were relatively quick to estimate. The fastest model across feature set specifications was logistic with cross-validation (0.009), but the lack of a penalty term led to low performance on the previous metrics. The slowest models were also some of the best performing–logistic regression with cross-validation and L1 penalty took an average of 3.4 seconds to estimate. While this is acceptable in a corpus of our size, for larger corpora, adaptive boosting seemed to strike a good balance between fast estimation (0.017 seconds) and predictive accuracy.

4 Extension one: relax an assumption from preprocessing stage and shift from bigram to unigram representation

As discussed earlier, a unigram representation of the reviews loses information–for instance, failing to distinguish between a review that rates as product as "good" and one that rates it as "not good." In this extension, we create a bigram representation of the corpus by using the same pre-processing steps we used to create the unigram representation. We filtered out bigrams occurring fewer than 2 times in the corpus. Despite our expectations that the bigram representation would outperform the unigram representation–for instance, the vocabulary captured terms like "slow_servic", "real_good", and "mediocr_food" that would seem informative for the labels–evaluation returned worse metrics. For instance, and comparing the lasso featured selected data, the average ROC AUC was a much better 0.79 in the unigram representation compared to 0.56 in the bigram representation (which is barely better than chance).

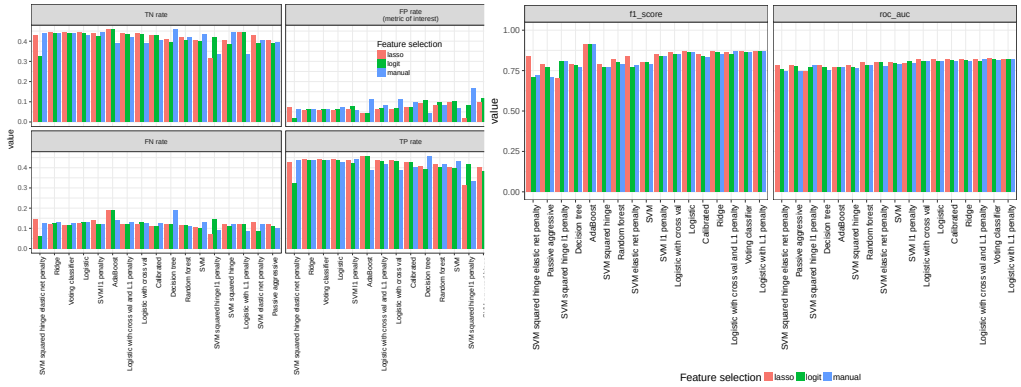
5 Extension two: averaging across the classifiers from step three for bias-variance tradeoff (ensemble methods)

In this extension, we use a very simple *ensemble* of classifiers to investigate whether combining the classification decisions from the unigram models in Table 1 can produce a better accuracy score overall. We know ensemble classifiers built from some aggregation of individual classifiers do this, and as such may improve predictive accuracy [1]. Following the discussion on *committee methods* in [2], we use a Voting Classifier to average over predictions across models via majority voting. For simplicity, we used an unweighted average of predictions. It performed in line with our highest performing models, with an AUC of 0.8. A future extension could include a weighted average of predictions. For example, we could use the BIC criterion, as [2] recommend.

6 Conclusion and related work

We see two extensions to improve classification accuracy and interpretability. First is modifying the stopwords dictionary and exploring bigram and trigram representations–the poor performance of the bigram may be related to the fact that words like "stop" and "very" were pruned from the corpus. Second, since of the better performing models (logistic with L_1 penalty and cross-validation) was the slowest to estimate even on the small corpus, we would want to develop better summary metrics that weight the tradeoff between estimation speed and predictive accuracy depending on the task (e.g., how quickly does the client need to be able to classify reviews).

Figure 1: The left hand panel depicts confusion matrix results, while the right hand panel depicts the F1 score and ROC AUC metrics



References

[1] Pedro Domingos. A unified bias-variance decomposition and its applications. In *In Proc. 17th International Conf. on Machine Learning*, pages 231–238. Morgan Kaufmann, 2000.

[2] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[3] Judy Kay, Paul Thomas, Andrew Trotman, Judy Kay, Paul Thomas, and Andrew Trotman. editors. In *Proceedings of the Workshop on Adaptive Hypertext and Hypermedia User Modelling '94, Cape Cod*, 1994.

[4] Andrew Y. Ng. Feature Selection, L1 vs. L2 Regularization, and Rotational Invariance. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 78–, New York, NY, USA, 2004. ACM.

[5] 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.

[6] Lorenzo Rosasco, Ernesto De Vito, Andrea Caponnetto, Michele Piana, and Alessandro Verri. Are loss functions all the same? *Neural Computation*, 16(5):1063–1076, 2004.

[7] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.