# Classification of Customer Reviews using Sentiment Analysis Techniques

**Gregory McCord**
Princeton University '20
gmccord@princeton.edu

## Abstract

One of the best ways for a company to receive feedback on its product is through customer reviews like those on third party sites like IMDB or Amazon. A shoe company like Nike or a film studio like Marvel Studios might seek feedback on their most recent product before beginning development on the next generation of products to cater to the desires of the customers. Companies may want to gather several data points about their product from the reviews, but in general, the major source of data they seek from a review is whether the customer felt positively or negatively about their experience with the product. In this paper, we address the problem of automating the process of determining the sentiment of each customer in their review. We initially analyze 3000 customer reviews from Yelp, IMDB, Netflix, and Amazon and classify them as either positive or negative. We train the classifiers on bag-of-words representations of each review, with and without feature selection. We find that most of the methods tested, especially the Support Vector Machine, prove effective in classifying the data based on estimates of misclassification rate and the area under the Receiver Operating Characteristic (ROC) curve while the K-Nearest Neighbors classifier fell short. However, we also observe the power of a pre-trained recursive neural network, which proves to be even more effective than any of our classifiers.

## 1 Introduction

The need for sentiment analysis is everywhere. Companies need to understand how their products are being received by consumers by reading reviews. However, it's impractical for humans to read thousands of reviews across the web in order to determine the general sentiment that people felt about the product. Furthermore, there are countless places to pull these reviews from including Twitter, Facebook, or any other website with a well-documented API. It is because this data can be so easily retrieved that data analysis methods prove to be both useful, practical, and efficient.

In this paper, we evaluate $5$ binary classification models for determining the sentiment of a review. We modeled the reviews using a bag-of-words representation due to its spatial efficiency and scalability. We evaluated the performance of the datasets with and without feature selection to highlight the varying levels of effectiveness of different classification models and the benefits of using feature selection.

## 2 Related Work

We have been unable to find evidence of published sentiment analysis research being done using this same data set. However, there are likely hundreds of other Princeton COS 424 students who have used this data set for their own projects before. Most will likely have used the bag-of-words implementation for their analysis, and some may have even tested the same set of classifiers. However, later we will present extensions that will likely not have been considered by other users of the data set. It is unknown whether feature selection proved useful in their research.

### 2.1 Data Processing

We downloaded the $3,000$ reviews ($2,400$ of which are training samples) on February 7, 2018 from the course Piazza page. We used the Python NLTK library to tokenize, convert to lowercase, remove stop words, lemmatize, stem using the Porter method, and filter out the words that occurred fewer than $5$ times in all of the reviews. Before feature selection, there were $541$ words (features) in the vocabulary. After feature selection there were $275$ words in the vocabulary, which is approximately $50\%$ of the original feature set. Feature selection was performed using a support vector machine (SVM) with a linear kernel and $\ell_1$ penalty. The hyperparameter for the SVM model feature selection was tuned using 5-fold cross validation to obtain the ideal number of features. Feature selection and tuning were performed exclusively using the training data.

### 2.2 Classification Methods

We used $5$ classification methods from the SciKitLearn Python library [1]. All methods with the exception of the Random Forest function had their corresponding hyperparameter tuned using 5-fold cross validation. Each classifier was tuned for performance twice, once for the data with all of the features present (N) and once for the data post-feature selection (FS). The Random Forest was not tuned due to the lengthy training time per iteration and large number of iterations required to properly tune it. The $5$ classifiers used are:

1. *Support Vector Machine with $\ell_1$ penalty and linear kernel* (SVM): the penalty parameter was $0.2$ for N and $0.9$ for FS
2. *Naive Bayes classifier* (NB): using the multinomial variation; the smoothing parameter was $0.9$ for N and $0.7$ for FS
3. *K-Nearest Neighbor* (KNN): the number of neighbors was $5$ for N and $9$ for FS
4. *Logistic Regression with $\ell_2$ penalty* (LR): the penalty parameter was $0.9$ for N and $0.8$ for FS
5. *Random Forest* (RF): the number of trees was set to $500$ for N and FS

### 2.3 Evaluation

For each classification method and feature set (either N for all the features or FS for post-feature selection), we trained the data on the training set and evaluated the model on the testing set using the corresponding feature set. As stated before, each model's hyperparameter was tuned using the same 5-fold cross validation folds before fitting the model and predicting the binary class labels on the test data. We compared these models using the misclassification rate (MCR) and the area under the Receiver Operating Characteristic (ROC) curve, which compares the true-positive rate (TPR) to the false-positive rate (FPR). The characteristics can be defined as follows in terms of the number of false positives (FP), true positives (TP), false negatives (FN), and true negatives (TN):

$$\text{MCR} = \frac{FN + FP}{FN + FP + TN + TP}$$

$$\text{TPR} = \frac{TP}{TP + FN} \quad , \quad \text{FPR} = \frac{FP}{FP + TN}$$

## 3 Spotlight Classifier: Support Vector Machine

The Support Vector Machine (Vapnik, 1963) is a linear classification algorithm that finds a hyperplane that maximizes the margin between the two classes of samples in binary classification. Given the data $\mathcal{D} = (\mathbf{x}_1, z_1), ..., (\mathbf{x}_n, z_n)$ where $z \in \{-1, 1\}$ represents the binary classes, the SVM attempts to define the hyperplane by the following equation:

$$\mathbf{w^T x} + \mathbf{w_0} = 0$$

For SVMs, we also define the hinge loss, which represents our loss function for categorizing a sample as $+1$ or $-1$. The hinge loss is defined as follows:

$$L_{\text{hinge}}(z, \eta) = \max(0, 1 - z\eta)$$

Lastly, we will introduce the slack term $\xi_i$, so instead of requiring $z_i\eta_i \geq 1$, we now allow for $z_i\eta_i \geq 1 - \xi_i$. This substitution has two benefits. First, it allows for us to use non-linearly separable data by imposing a soft rather than hard constraint. Second, it gives us an optimization equation that is differentiable since we are no longer worried about our original hinge loss function. This leads us to the following optimization equation:

$$\min_{\mathbf{w}, w_0} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i \quad s.t. \quad \xi_i \geq 0, \quad z_i(\mathbf{x}_i^T\mathbf{w} + w_0) \geq 1 - \xi_i, i \in \{1, N\}$$

We can then solve and find the minimum, which has the form:

$$\hat{\mathbf{w}} = \sum_i \alpha_i z_i \mathbf{x}_i$$

This solution therefore defines our hyperplane. In fact, this is the hyperplane that maximizes its distance from both the $+1$ and $-1$ classes. Additionally, $\alpha_i$ is sparse due to the hinge loss function ($\alpha_i = 0$ for any prediction $|\eta| \geq 1$ where the prediction and truth $z$ share the same sign). As such, the $\mathbf{x}_i$ where $\alpha_i > 0$ are known as support vectors because they define the hyperplane. These support vectors additionally represent the selected features of the SVM model. In order to use the model for prediction, we will return to our equation for our hyperplane, substituting in our estimate.

$$\hat{z}(x^*) = \text{sgn}\left(\hat{w}_0 + \sum_{i=1}^{n}\alpha_i z_i \mathbf{x}_i^T\mathbf{x}^*\right)$$

To allow the method even more flexibility, we may use the kernel trick for defining the margin. It is important to note that for our classifier in this experiment, we simply used the linear kernel.

$$\hat{z}(x^*) = \text{sgn}\left(\hat{w}_0 + \sum_{i=1}^{n}\alpha_i z_i \kappa(\mathbf{x}_i^T, \mathbf{x})^*\right)$$

The time to compute a prediction for future samples is directly proportional to the number of support vectors (those variables contained within the margin of the hyperplane). As we did in our experiment, it is possible to control the width of the margin to select additional features. By adjusting the C term as seen before in the optimization equation $C\sum_{i=1}^{N}\xi_i$, we affect the weight of the sum of the slack terms, which represent misclassified points. By increasing their weight, we force the model to accommodate more errors and thereby select more features [4].

Before concluding, we must address the fundamental assumption that SVMs make about the data - that the data are independent and identically distributed. While this is generally a fair assumption to make, any samples that are dependent upon each other will lead to an improper model. For our purposes however, knowing the sentiment or word content of another review in the training set did not affect our prediction of the label of another in the same set. As such, it is a fair assumption to make for our purposes.

## 4 Results

### 4.1 Evaluation of Results

As detailed before, we tested all 5 of our classifiers on both the FS and N datasets. We used the misclassification rate as our primary point of comparison, but we reference the ROC scores as well, because they help differentiate between similar models. We can see from the data in Table 1 some striking statistics.

| Classifier | Feature Selection | | No Feature Selection | |
|---|---|---|---|---|
| | MCR | ROC | MCR | ROC |
| SVM | 0.182 | 0.881 | 0.178 | 0.890 |
| NB | 0.193 | 0.877 | 0.230 | 0.862 |
| KNN | 0.298 | 0.799 | 0.340 | 0.747 |
| LR | 0.182 | 0.887 | 0.197 | 0.877 |
| RF | 0.188 | 0.882 | 0.216 | 0.868 |

Table 1: **Results from five classifiers on 600 reviews in the test set.** For each classifier, we report the Misclassification Rate (MCR) and the area under the ROC curve for both the data with the selected features and the data with all of the features.

As expected, feature selection helped to reduce the noise in the data allowing most of the methods to perform significantly more effectively. In fact, in all methods other than the Support Vector Machine, the classifiers had a misclassification rate on the FS data that was on average 12% better than the corresponding model's misclassification rate on the N data. It then begs the question, why did the Support Vector Machine, the model with the highest ROC score and lowest misclassification rate, perform worse on the FS data? The answer lies in our data processing step. SVM performs feature selection internally while it fits the model due to the support vectors. However, when we performed feature selection, we also used the SVM to select features and removed all the others in order to create the FS dataset. This creates overfitting however, because the SVM has already removed the least indicative variables from the model, but it is now forced to remove even more, thereby decreasing the model's efficiency. On a similar note, the Random Forest classifier also performs feature selection internally, but it's ROC score on the FS data was significantly higher than on the N data. This is likely due to the fact that the RF method found a certain set of features to be more indicative of performance than the SVM, and because there was not much overlap between those two sets, the act of selecting features using two different models actually helped to weed out more noise for the random tree model than would have been the case otherwise.

We actually would have expected the Random Forest method with 500 trees to have performed even better when compared with some of the other methods. Especially with a vocabulary of only 541 words in the N dataset, the RF method should have found more efficient branches to make. Similarly, we also would have expected better from the KNN classifier, which was the worst performing classifier by a significant margin on both datasets. In both cases, we believe that the small sample size of 2400 reviews in the training data led to a small vocabulary, which ultimately caused these methods to perform less efficiently. The RF method is a discriminative classifier and therefore thrives on large datasets, which partly explains why it did not perform as well as expected. Similarly, KNN relies on clustering, but with such a small test set, it is likely that there are no samples in the training set that are very similar to samples in the testing set, leading to a pseudo-random classification of some reviews. In both cases, large datasets should help to improve performance significantly. The last question to answer would then be, why would the SVM method, which is also a discriminative model, perform so well despite being dependent on large training sets? The answer lies in the fact that most text is linearly separable, leading to strong performance of classifiers with a linear kernel such as the SVM [2]. As a result of this tendency in text classification, linearly separable classifiers such as the SVM naturally perform very well, even on data with small training sets.

### 4.2 Feature Selection

| Top Selected Features | | | | |
|---|---|---|---|---|
| great | love | good | bad | excel |
| worst | movi | would | wast | poor |
| nice | best | amaz | delici | well |
| disappoint | like | fantast | terribl | film |

Table 2: **Top 20 predictive words in the model.** The top 20 words were ranked according to their Gini scores using the data set with the selected features. The words were identified using the Random Forest classifier with 500 trees.

We have already touched on the general approach to feature selection, but now we would like to address some specific (stemmed) features that were selected as shown in Table 2. While sklearn does not have a native way of determining the relevance of each of these important features for classifying the data as either the positive or negative class, we can manually tag most of these quite easily as seen in the Table 3.

| Positive Sentiment | |
|---|---|
| great | love |
| good | excel |
| nice | best |
| amaz | like |
| delici | well |
| fantast | |

| Negative Sentiment |
|---|
| bad |
| worst |
| wast |
| poor |
| disappoint |
| terribl |

Table 3: **Top 20 predictive words broken down into positive and negative categories.** Note that the 3 words with no clear positive or negative sentiment were held out from these charts, hence the presence of only 17 features.

However, there are three terms that stand out as not obviously positive or negative - ”would”, ”film”, and ”movi” (short for ”movie”). In the case of ”would”, the value of the feature seems to tie in with other features that it is dependent on such as the features ”go” and ”never” (which would likely heavily affect the positive or negative sentiment of the nearby terms). In the case of ”film” and ”movi”, it seems that the random forest classifier highly valued knowing the type of review before classifying it.

## 5 Discussion and Conclusion

In this work, we compared 5 different classifiers on a data set with two different sets of features: the N data which contained all features and the FS data which contained only those features selected by the SVM during data processing. When considering the misclassification rate, the most effective form of classification appears to be the SVM with $\ell_1$ penalty and linear kernel with no feature selection having already been performed, instead allowing the SVM to perform its own feature selection. The standard logistic regression model also performed well on both data sets due to the linear separability of most cases of sentiment classification. As the data shows, the KNN classifier had a significantly higher misclassification rate, and therefore performed much worse, than all of the other models.

We believe that there are many ways to extend this project in order to achieve even better results. The first and most obvious method would be to increase the sample size by an order of magnitude or even two. This would allow specifically the KNN and Random Forest methods to improve significantly by introducing more features and improving clustering. Another way to extend the project would be to test the data using more modern methods that are used in text classification today. One of the most powerful NLP libraries used today is the Stanford CoreNLP library, which uses a recursive neural network [3]. This pre-trained sentiment classifier classifies reviews as either positive, negative, or neutral. We analyzed two different treatments when running the data through the model. In the first case, we randomly assigned the 68 neutral reviews in the test case to either positive or negative sentiment, which resulted in a misclassification rate of 0.176 (better than even the SVM). However, when we factored out the neutral reviews, we found that the classifier had a misclassification rate of 0.135, which was significantly better than any other classifier. This final rate demonstrates the peak performance of the model, which could hypothetically be achieved by preventing the model from classifying reviews as neutral. This classifier shows the magnitude of the effect of training discriminative classifiers on massive data sets and on using the most powerful classifiers available to the task at hand.

## References

[1] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, and et al. Grisel O. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[2] Thorsten Joachims. *Learning to Classify Text Using Support Vector Machines*. Springer-Verlag New York Inc, 2002.

[3] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014.

[4] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2013.