Brandon Croarkin

# HW #8 – Lie Detection

## Introduction

Some people claim that machine learning algorithms can now figure out whether a person is lying or not. Being able to tell when a person is lying is an important part of everyday life. Despite everyone claiming that they are honest, people lie in all types of situations. A machine algorithm that could detect these lies could be extremely valuable. Being able to spot lies in a courtroom could literally be the difference between a verdict of innocent or guilty.

To test this claim, analysis will be done on a collection of customer reviews, some of which are true and some of which are fake. This data set also has sentiment labels for each review. So, in addition to using machine learning techniques to detect lies, there will be additional analysis to determine sentiment. This combination of features is useful, since sentiment analysis is a well-documented field, the performance of the models on sentiment analysis will provide an interesting comparison to the lie results.

## Analysis

### The Data

The dataset to explore this question comes from a collection of customer reviews, some are true, and some are fake. This data set also has sentiment labels for each review. There is a total of 92 reviews in the dataset. Of these 92, half are true, and half are false. Of these two halves, half are positive reviews and half are negative reviews.

### Data Exploration

Before running any machine learning models on the data, it is important to explore and visualize the data to understand it better. Since the dataset in question contains word data, a great way to do this is via word clouds. Word clouds display the more frequently appearing words in a dataset and show them with the most frequent terms appearing larger in size than less frequent terms.

**Whole Dataset Word Cloud**



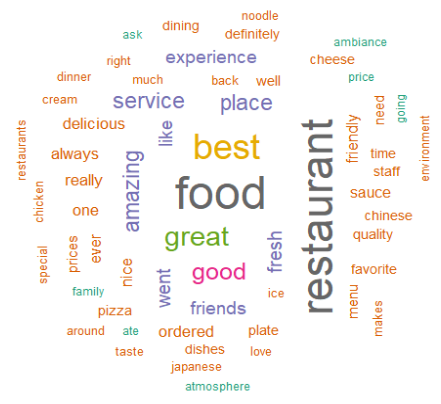
**Lies Word Cloud**



**Not Lies Word Cloud**



**Positive Sentiment Word Cloud**



**Negative Sentiment Word Cloud**



### Data Preparation

To do any data analysis on this dataset, it needs to first be cleaned and formatted. The original dataset consists of three columns: “lie”, “sentiment”, and “review”. Since the main data to analyze in this data set are the words in the text, this data needs to be formatted for analysis by transforming words into features.

The first step in processing text data involves creating a corpus, which is a collection of text documents. This is done using the VCorpus() function in the tm package. To actually perform the analysis, these reviews then need to be divided into individual words. However, before this can be done the text needs to be cleaned.

There were two rows that only contained “?” as the only text. This text does not seem to have any meaningful information, so those rows were removed. After this, the next data cleaning step involves removing punctuations, converting all words to lower case, removing numbers, and removing stop words. These steps are all meant to standardize the words by removing punctuation and other characters that could clutter the result.

The next step after cleaning the data in the “text” column is to split the messages into individual components through tokenization. A token is a single element of a text string. After doing this transformation through the DocumentTermMatrix() function, a data structure is created in which rows indicate documents and columns indicate words. Each cell stores a number indicating the count of the times the word appears in the document represented by the row.

The final step taken to clean the data was to filter down the features. The document term matrix created had 1334 terms, one for every word that appears at least once in a document. It is unlikely that all these features are useful for classification, especially given that many do not occur in certain pieces of text at all. To reduce the number of features, words that appear in less than 5 documents of the train data were eliminated.

After the data has been cleaned and converted to a document term matrix, the next step is to split the data into training and test sets. This was done using an 80-20 train to test split. There are two sets of training and test labels, one for the “lie” column and another for the “sentiment” column since they are being predicted separately.

The train and test set created works for SVMs since it can use numeric variables, but the Naïve Bayes classifier is typically trained on data with categorical features. Since the document term matrix is a sparse matrix with numeric data, this poses a problem. To solve this, an additional data frame was created where the words were converted to a categorical variable that indicate whether the word appears at all.

### Data Processing

#### Naïve Bayes

The analytical approach taken for both the lies and sentiment detection were the exact same. It began by using the default parameters of Naïve Bayes from the e1071 package in R. The model was then tuned to include the Laplace estimator. Without the Laplace estimator, words that appeared in none of the lie or sentiment texts would have a large impact on the classification process. Since Naïve Bayes computes probabilities by multiplying the probabilities of the features, any feature with 0% probability in “lie” or “no lie” would result in a 0 probability. The Laplace estimator solves this problem by adding a small number to each of the counts in the frequency table, which ensures that each feature has a nonzero probability of occurring within each class.

The tables below reflect the different models used, their parameters, and their results.

**Lies**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Parameter Setting** | **Overall Accuracy** | **Precision in category I** | **Recall in category I** | **Precision in category II** | **Recall in category II** |
| Default | 44.4% | 63% | 42% | 30% | 50% |
| Laplace = 1 | 50% | 63% | 45% | 40% | 57% |

In both models created with Naïve Bayes on the lie column the accuracy does not break 50%. This means that the ability to predict a lie is no better than a flip of a coin!

**Sentiment**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Parameter Setting** | **Overall Accuracy** | **Precision in category I** | **Recall in category I** | **Precision in category II** | **Recall in category II** |
| Default | 77.8% | 73% | 89% | 86% | 67% |
| Laplace = 1 | 83.3% | 73% | 100% | 100% | 70% |

In the Naïve Bayes model on sentiment detection, the best performing model reached 83% accuracy. With both lies and sentiment there is an increase in accuracy when the Laplace estimator is added. This makes sense given the sparse nature of the dataset, where the 0’s could influence the outcome.

#### SVMs

**Lies**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Parameter Setting** | **Overall Accuracy** | **Precision in category I** | **Recall in category I** | **Precision in category II** | **Recall in category II** |
| Linear kernel | 61.1% | 73% | 89% | 86% | 67% |
| Linear kernel, C = 5 | 61.1% | 73% | 89% | 86% | 67% |
| Radial kernel | 44.4% | 100% | 44% | 0% | NA |
| Polynomial kernel | 61.1% | 73% | 89% | 86% | 67% |
| Polynomial kernel, C = 10 | 61.1% | 73% | 89% | 86% | 67% |

While the SVMs perform better than the Naïve Bayes at detecting lies, they do not see an overall accuracy above 61%. This is even after trying multiple different kernels (“linear”, “radial”, and “polynomial”) and different cost of constraints.

**Sentiment**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Parameter Setting** | **Overall Accuracy** | **Precision in category I** | **Recall in category I** | **Precision in category II** | **Recall in category II** |
| Linear kernel | 66.7% | 55% | 86% | 80% | 80% |
| Radial kernel | 77.8% | 64% | 100% | 100% | 64% |
| Radial kernel, C = 5 | 72.2% | 64% | 88% | 86% | 60% |
| Radial kernel, C = 10 | 72.2% | 64% | 88% | 86% | 60% |
| Polynomial kernel | 66.7% | 55% | 86% | 88% | 58% |

Interestingly, the SVMs performed worse on the sentiment analysis than the Naïve Bayes model even though the SVMs performed better with the lie data. Still, overall there is higher accuracy in sentiment classification accuracy than there is in lie classification accuracy. There is a notable boost from changing from the linear kernel to a radial kernel from 66.7% up to 77.8%, however there is no boost from increasing the cost of constraints. In fact, there is actually a drop in accuracy.

### Results

From the analysis above, there is a clear difference in performance in classifying sentiment vs. classifying lies. It is far easier to detect a positive vs. a negative review than it is to classify a lie. The best result in lie detection was 61.1%, which is barely larger than a flip of a coin. In comparison, the best model in detecting positive sentiment was 83.3%. This is a strong result given the difficulty of sentiment analysis.

The most important words for classifying lies with Naïve Bayes were “dining”, “pizza”, “quality”, “good”, “dishes”, “need”, “salad”, “like”, “order”, “staff”, “waitress”, “back”, “best”, “experience”, “really”, “went”, “asked”, “just”, and “never”. No word showed up as an important feature using SVM. This shows that there is no pattern to these words that is helpful in determining a lie. This makes sense given the poor results of the models and makes evident one of the reasons that makes lie detection so difficult.

The most important words for classifying sentiment with Naïve Bayes were “always”, “even”, “favorite”, “need’, “waitress”, “restaurant”, “food”, “one”, “wait”, “back”, “best”, “experience”, “good”, “place”, “really”, “service”, “went”, “asked”, “just”, and “like”. The only word that registered as an important feature using SVMs was “best”. These words make sense as they seem to point to things/factors that would influence how one’s visit to a restaurant was. However, some of these words could also be used in a positive or negative context. The role of context makes sentiment classification difficult and likely explains why the top accuracies reached were 83% and not in the 90’s.

These top feature words were found with the FSelector package in R. It uses information gain to select the best combination of attributes. Information gain is the amount of information that’s gained by knowing the value of the attribute, which is calculated by comparing the entropy before and after the attribute is added to the model. The largest information gain is equivalent to the smallest entropy.

## Conclusions

Based on the results of this analysis, lie detection is not yet within the realm of the possibilities of machine learning. However, this may also just be a limitation of the data used. The small size of the dataset may not give enough data for the machine learning algorithms to find any pattern. Or, the data in this dataset just may not be the right kind of data needed to detect lies. Other attempts at using machine learning to detect lies used body cues, such as eye movement and temperature readings. So, it may not be that machine learning is incapable of detecting when humans lie, but that we are not able to do it from just what they say. Although this is technically a non-result, this is valuable information.

Sentiment classification, on the other hand, is well within the scope of machine learning. However, it is still a difficult task due to the large degree of variation in meanings of word, especially based on the context within which they are used. The next step to make a leap in the performance of these models is to be able to figure out the context in which a word is used. Doing this would make for significant performance boosts.