Jacob Dineen

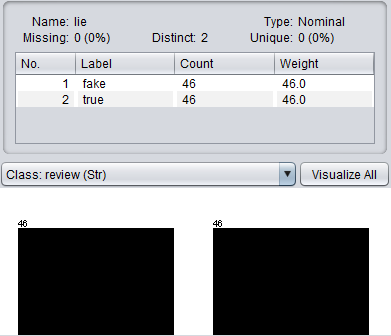
NETID: Jdineen

IST565

Homework 8

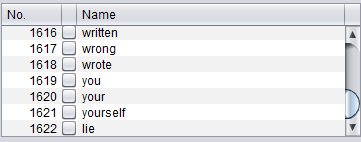
In this assignment we will explore the user of text mining in regards to a classification problem. The class problem at hand is deciphering truth amongst, what appears to be, exclusively the restaurant industry. We will be utilizing Weka for the entirety of this experiment.

Starting with the initial load of the data, we can begin to understand the structure. There are 3 features: ‘Lie’, ‘Sentiment and ‘Review’. The first two attributes are nominal, and the last is a string. If we look at the number of instances, or we enter the editing window, we can see that there are 92 rows/instances of data. Of those 92 instances, we have an even distribution amongst our two nominal variables, being lie and sentiment. In this case we have two dependent variables that are both reliant on the ‘review’ feature. Of those dependent variables, ‘lie’ is the underlying outcome that we are desiring to predict, so that will be our class variable. It seems necessary for us to have a single dependent variable during induction, discovered through trial and error, so we will remove the variable ‘Sentiment’ for our dataset to perform the task of lie detection, and circle back to remove ‘Lie’ when looking at sentiment analysis.. From coursework, we understand that we will need to tokenize/vectorize that feature before performing inductive methods.



In Preprocessing, we navigate to the StringToWordVector Filter and look at the default settings. Initially, we will leave most of the parameters default, but change the OutputWordCounts to True. We don’t see many delimiters that were tokenized with character values, so we won’t change that setting on the WordTokenizer just yet. After the vectorization is complete, we now see 1479 attributes, or 1477 unique tokens excluding our nominal variables.

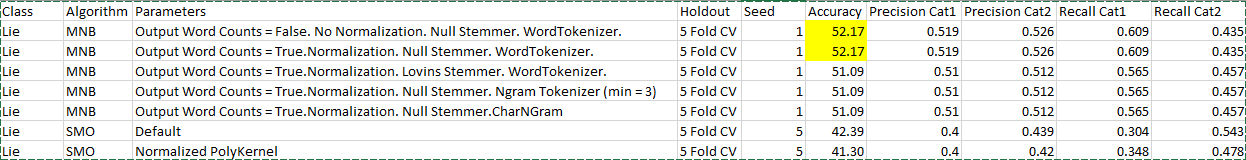
The first word shown as an attribute is #winning. A hashtag could be added as a delimiter, but we’d change the meaning and the origin of word itself. Additionally, it may be possible to remove certain attributes, but to keep our first result pure, we’ll run the classification model as is. Our last step in preprocessing will be to move our class variable to the end of our dataset, or the last column – Weka identifies the last column header as the class label.

\*weka.filters.unsupervised.attribute.Reorder -R 2-last,1

Lie Detection

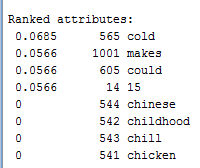
Starting off with lie detection, we use our original dataset with ‘Sentiment’ removed during feature selection. We could preprocess, or we could use the all in one mechanisms of a Filter Classifier – I prefer this method due to flexibility in parameter tuning of our string to word vector filter. We are tasked with using the Multinomial Naïve Bayes as well as Support Vector Machines to determine the validity and accuracy of each model. We know, from our coursework, that there is empirical evidence that MNB is best used for text mining problems. It could be intuitively argued that because Naïve Bayes is parametric and performs induction and deduction with the assumption that each of the features, in the case word tokens, are conditionally independent of one another, the model should struggle. But, as we’ve seen working with other datasets, variables that break that assumption rarely cause inaccurate predictions. Multinomial Naïve Bayes also appears to be a parametric algorithm, as it acts under the assumption that there is a multinomial distribution whereas Naïve Bayes classifiers that we had tested were generally using a normal distribution. Looking at a comparison of these two different algorithms in relation to accuracy. One of the positives about SVMs is that they can use kernel trick to put their inputs/instances into a multidimensional space allowing for a hyperplane to be drawn that wasn’t possible in a two-dimensional space. From reading, it appears that Naïve Bayes is often a better choice, over SVMs, particularly when the size of the training dataset is limited: “If you have fairly little data and you are going to train a supervised classifier, then machine learning theory says you should stick to a classifier with high bias, as we discussed in Section [14.6](https://nlp.stanford.edu/IR-book/html/htmledition/the-bias-variance-tradeoff-1.html#sec:secbiasvariance) (page [[*]](https://nlp.stanford.edu/IR-book/html/htmledition/the-bias-variance-tradeoff-1.html#p:secbiasvariance)). For example, there are theoretical and empirical results that Naive Bayes does well in such circumstances ([Forman and Cohen, 2004](https://nlp.stanford.edu/IR-book/html/htmledition/bibliography-1.html" \l "forman04learning), [Ng and Jordan, 2001](https://nlp.stanford.edu/IR-book/html/htmledition/bibliography-1.html" \l "ng01discriminative)), although this effect is not necessarily observed in practice with regularized models over textual data ([Klein and Manning, 2002](https://nlp.stanford.edu/IR-book/html/htmledition/bibliography-1.html#klein02conditional)). At any rate, a very low bias model like a nearest neighbor model is probably counter indicated. Regardless, the quality of the model will be adversely affected by the limited training data” (<https://nlp.stanford.edu>).

Starting with lie detection model generation, we remove the sentiment variable from our data. Below is a table and a graph showing results, and parameterization:



As we can see, there was very little significance to the prediction capabilities of detecting lies based on frequency. Intuitively, it does make sense that a machine may struggle with detecting lies, as it requires a certain level of intuition and is more easily picked up through social queues rather than diction. Also, we have to consider that words, on their own, aren’t particularly meaningful in classifying lies, as it relies more on context and undertone, which is hard to read, even for a human, when looking at text. What may have been more beneficial is if we had user profiles of each reviewer, and we could look at the past levels of truthfulness, as verified by professional in the space, and include that data within our model.

As it stands, our ML algorithms were either able to guess slightly above or below random guess, which is 50/50 in a case where the class labels are evenly distributed. Looking at the information gain of the features, only 4 unique words were deemed significant in the model:



Sentiment Analysis

Know that we know that machine learning tends to struggle with subjective classes, or classes that may require past intuition on human characteristics, we have a more straightforward ML task. Sentiment analysis is the idea of taking a group of words, or a string, and scoring it based on the executed language. So, in our case, we will be looking at the sentiment, be it positive or negative, for the restaurant reviews within our dataset.

To do this, we reload our original dataset and remove the ‘lie’ class. We can reorder the features to make it easier, moving our class label to the end of the dataset.



=== Confusion Matrix ===

a b <-- classified as

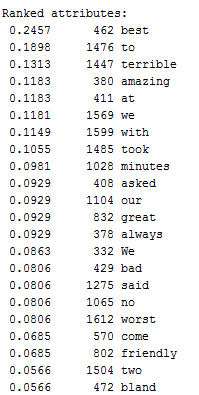
37 9 | a = negative

2 44 | b = positive

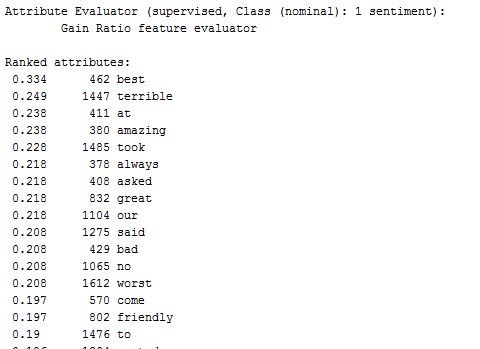
Many of our prediction errors involved false negatives , which in this case can be defined an instance that of negative sentiment which was classified as positive. Looking at all of our models built for sentiment classification, our lowest average metric came from Recall regarding category 1.

SVMs outperformed Multinomial Naïve Bayes in every test that we ran. Using the default PolyKernel produced the best results. We noticed that changing the Stemmer or adding in a stopwords handler, or even switching TDIDF on, didn’t move the result at all.

Using infogain on the sentiment attribute yielded more robust results:



We can see many adjectives (features) that helped the algorithm in making a class decision. ‘Best’ was far and away the most useful for the models, and seemingly helped to classify positive sentiment.



Work Cited:

https://nlp.stanford.edu/IR-book/html/htmledition/choosing-what-kind-of-classifier-to-use-1.html