Parin Patel

Lie Detection: Through Support Vector Machine and Multinomial Naive Bayes

Introduction:

Imagine being able to know exactly when a person is lying or being deceptive. More than decade ago, this was a concept only reserved for movies and tv shows, like *Minority Report and Persons of Interest.* Both of which had similar plots where a device was used that could predict when a crime or incident was going to occur. From Hollywood to Silicon Valley, everyone seems to wants to be able to predict human cognition and intention better. Over the years, many machines that detect lies have been built. These traditional devices, often referred to as "lie detectors" rely on a person physiological changes, like blood pressure, pulse rate, and skin conductivity. Since this method detects abnormalities in these physiological indicators, a major flaw of this system is that it can be cheated by anyone with professional training. Additionally, in today's digital age where we communicate online via text so frequently, traditional methods of lie detections cannot be utilized.

Since we cannot physically see whoever is posting or writing online, other methods of lie detection had to be developed. This is where people claim machine learning algorithms can figure out whether a person is lying or not. We will test this claim by analyzing customer reviews through the use of Support Vector Machine and Multinomial Naive Bayes in order see if we can detect the fake reviews.

Analysis:

Parin Patel

Initially, we attempted to try to conduct this analysis in R. However, our initial models' accuracy was extremely low and was error-prone/had low reproducibly. We believe this is mainly due to the small size of the dataset. We chose to complete the homework in Weka for times-sake; however, due to personal interest, we will pursue fixing trying to create a lie detection model using R.

Data Preparation:

Using the .arff version of the data, first it was read into weka. The data has three different columns labeled lie, sentiment, and review. All three were initially read in to be Nominal data types. The lie and sentiment columns were kept as nominal. However, the review column was adjusted to Word Vector using the unsupervised filter called "StringToWordVector". Using this filter, we also made some adjustments to the parameters.

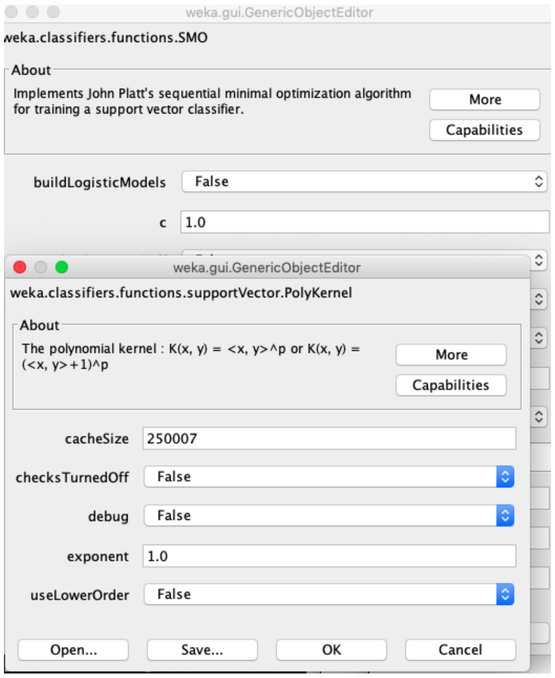
First, we used the WordTokenizer to return the individual word tokens. Additionally we added the "-" symbol within the word delimiters. Additionally, we turned off stemming by adjusting the NullStemmer to FALSE. We turned on the output WordCount to TRUE to provide the raw term frequency instead of the Boolean values. Finally we normalized the term frequency by turning on IDFTransform. We left TFTransform to False we do not want to return a log value for the term frequency. We then defined the attribute indices to only "last" so that it shows which specific attribute in the last "review" column we want to apply the vectorization to. Finally the lowercase Token was set to True so that we can merge any upper and lowercase by converting all the words to lowercase. This would then be added to a dictionary. Next, we wanted to remove words that maybe typos or errors, so we kept the minimum term frequency at 1. This will remove any frequencies that are less than one, and therefore, may be simply the result of a typo. Finally, we maintained a value of 1000 for the wordsToKeep parameter. This would mean that we want Weka to pick the top 1000 words in each category, which are first sorted by frequency, and then merged together after.

Support Vector Machine (SVM)

Parin Patel

Used this website as resource: https://machinelearningmastery.com/use-regression-machine-learning-algorithms-weka/

We chose to tune our support vector machine first. Under the classifiers.meta.FilteredClassifier we chose the SMO classier. Opening up the ObjectEditor we chose to use PolyKernal and then chose to calibrate it for Linear Regression by keeping our exponent to 1 (Figure 1). And kept our attributeIndices to "last". We kept our filter as Discretize, as well, since we would want to bin our attributes for review, which are listed as numeric currently, into nominal attributes.


weka.gui.GenericObjectEditor
weka.classifiers.functions.SMO
About
Implements John Platt's sequential minimal optimization algorithm for training a support vector classifier.
buildLogisticModels: False
c: 1.0

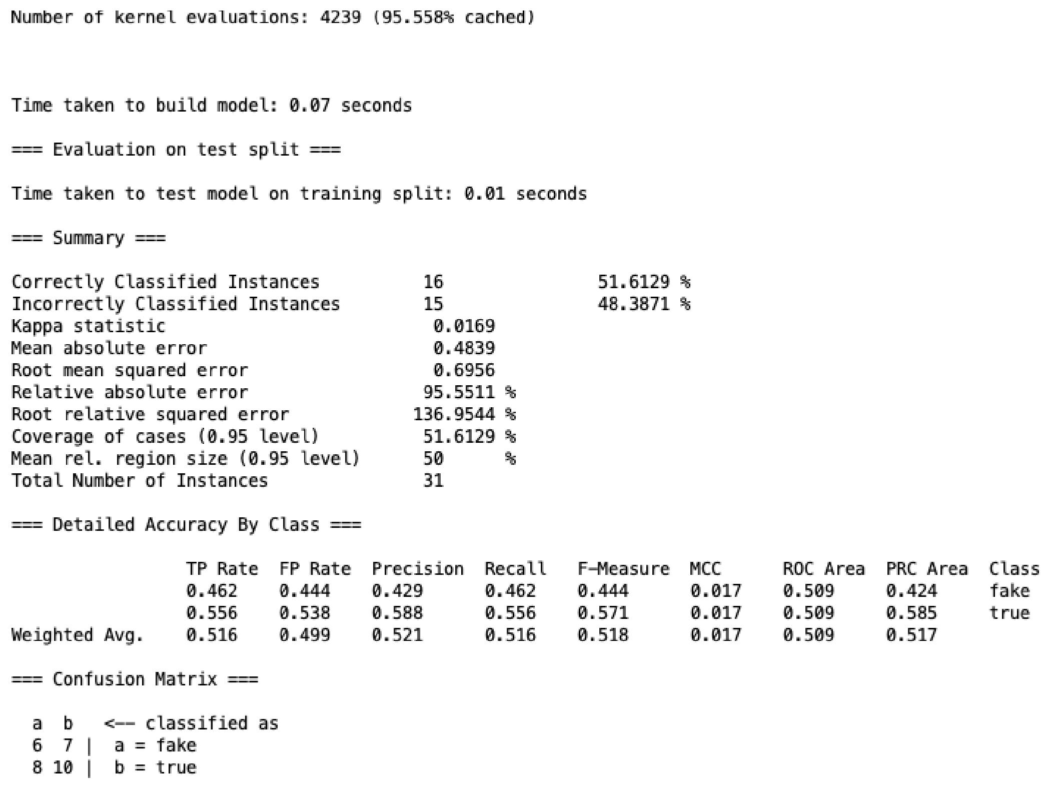
weka.gui.GenericObjectEditor
weka.classifiers.functions.supportVector.PolyKernel
About
The polynomial kernel : K(x, y) = <x, y>^p or K(x, y) = (<x, y>+1)^p
cacheSize: 250007
checksTurnedOff: False
debug: False
exponent: 1.0
useLowerOrder: False

Figure 1

Our output for lie detection is listed below in Figure 2. Figure 3 is the output for sentiment. Additionally, we have chosen to report the results in the table below. Results will be discussed in the Results section.

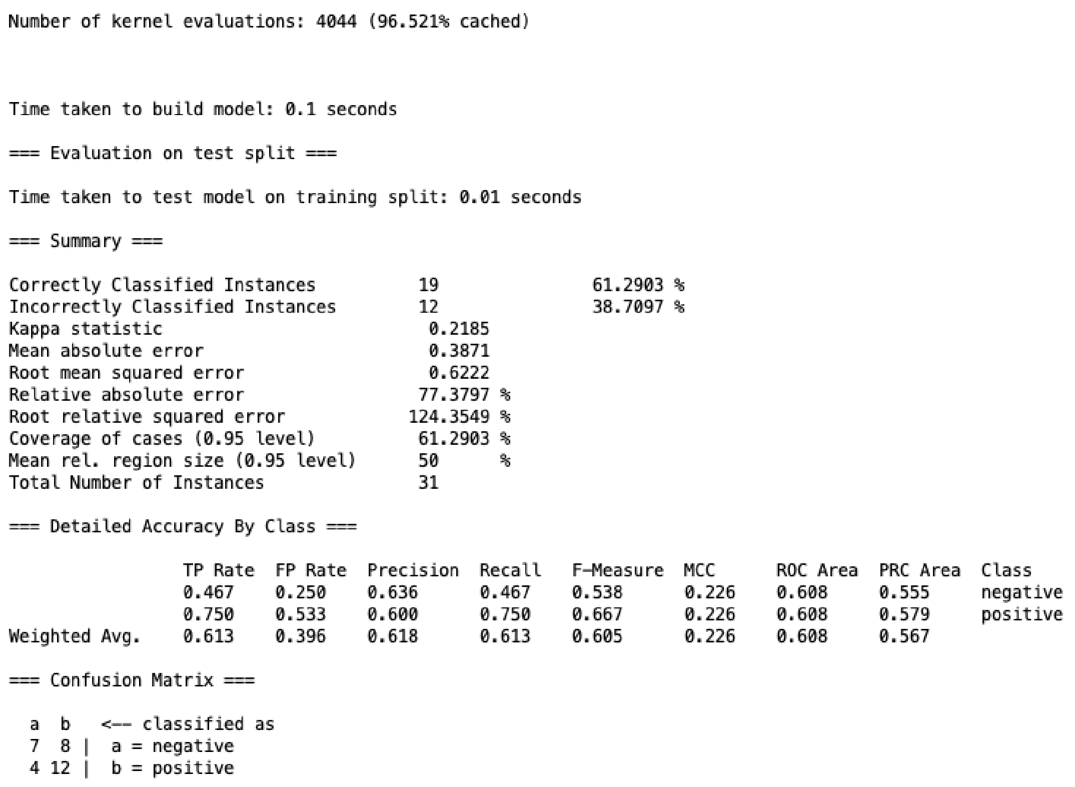
Parameter Setting	Overall Accuracy	Precision in Category I	Recall in Category I	Precision in Category II	Recall in Category II

Lie Detection	51.6129 %	0.429	0.462	0.588	0.556
Sentiment	61.2903 %	0.636	0.467	0.600	0.750



```
Number of kernel evaluations: 4239 (95.558% cached)



Time taken to build model: 0.07 seconds

=== Evaluation on test split ===

Time taken to test model on training split: 0.01 seconds

=== Summary ===

Correctly Classified Instances          16               51.6129 %
Incorrectly Classified Instances        15               48.3871 %
Kappa statistic                          0.0169
Mean absolute error                      0.4839
Root mean squared error                  0.6956
Relative absolute error                 95.5511 %
Root relative squared error            136.9544 %
Coverage of cases (0.95 level)          51.6129 %
Mean rel. region size (0.95 level)      50      %
Total Number of Instances               31

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.462    0.444    0.429      0.462    0.444      0.017    0.509     0.424     fake
                 0.556    0.538    0.588      0.556    0.571      0.017    0.509     0.585     true
Weighted Avg.    0.516    0.499    0.521      0.516    0.518      0.017    0.509     0.517

=== Confusion Matrix ===

  a  b   <-- classified as
  6  7 |  a = fake
  8 10 |  b = true
```

Figure 2

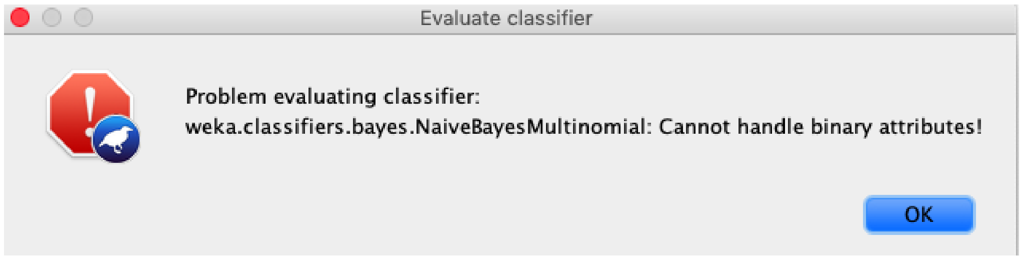


```
Number of kernel evaluations: 4044 (96.521% cached)



Time taken to build model: 0.1 seconds

=== Evaluation on test split ===

Time taken to test model on training split: 0.01 seconds

=== Summary ===

Correctly Classified Instances          19               61.2903 %
Incorrectly Classified Instances        12               38.7097 %
Kappa statistic                          0.2185
Mean absolute error                      0.3871
Root mean squared error                  0.6222
Relative absolute error                 77.3797 %
Root relative squared error            124.3549 %
Coverage of cases (0.95 level)          61.2903 %
Mean rel. region size (0.95 level)      50      %
Total Number of Instances               31

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.467    0.250    0.636      0.467    0.538      0.226    0.608     0.555     negative
                 0.750    0.533    0.600      0.750    0.667      0.226    0.608     0.579     positive
Weighted Avg.    0.613    0.396    0.618      0.613    0.605      0.226    0.608     0.567

=== Confusion Matrix ===

  a  b   <-- classified as
  7  8 |  a = negative
  4 12 |  b = positive
```

Figure 3

Multinomial Naïve Bayes:

We chose to tune for multinomial Naiive Bayes . Under the classifiers.meta.FilteredClassifier we chose the NaiveBayesMultinomial classifer. When we ran this classifier with its default settings, we came across an error (figure 4). As we adjusted the parameters, we still kept getting this error message

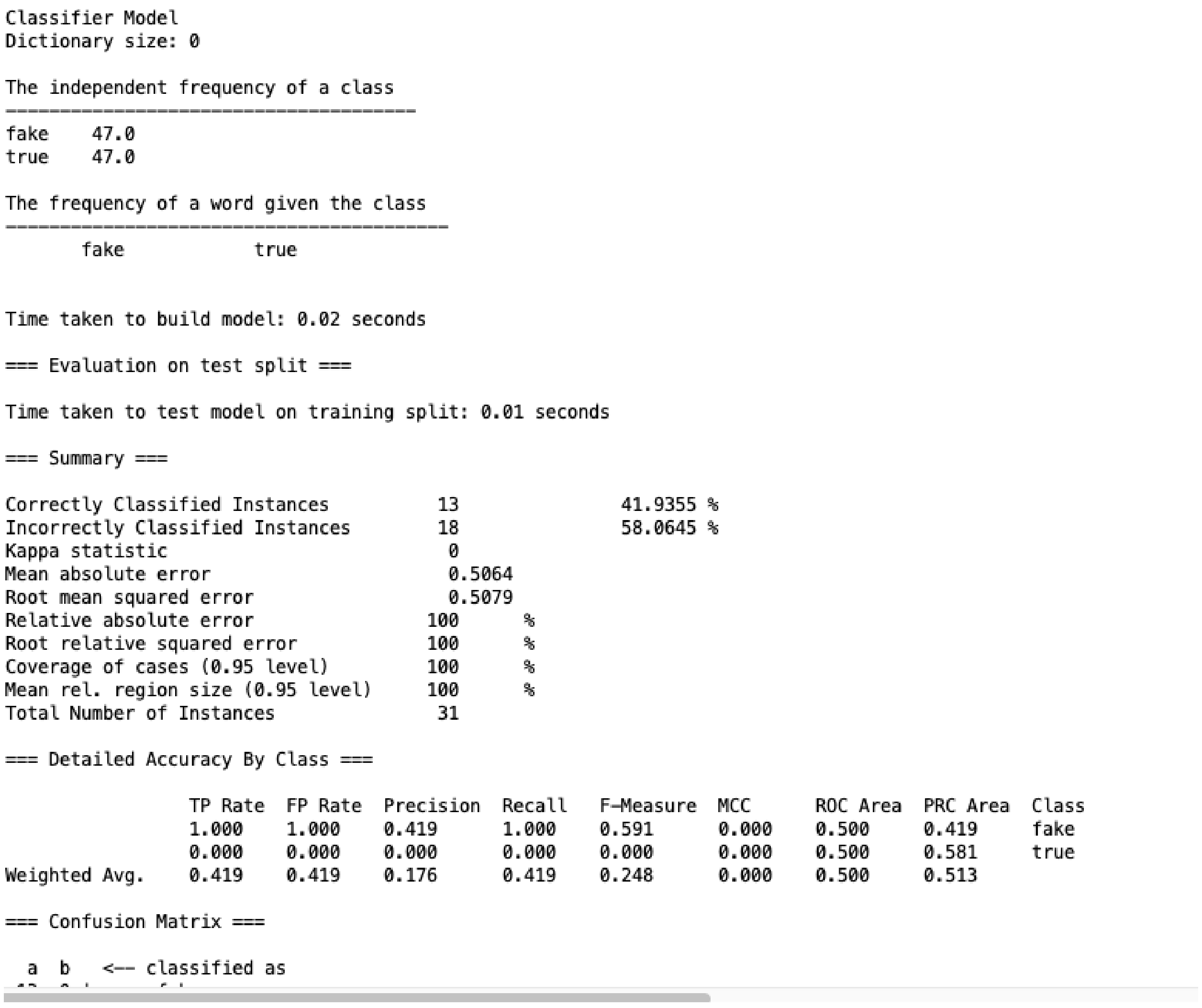


Evaluate classifier

Problem evaluating classifier:
weka.classifiers.bayes.NaiveBayesMultinomial: Cannot handle binary attributes!

Figure 4

We solved our problem by adjusting the classifier from NaiveBayesMultinomial to NaiveBatesMultinomilalText. Running it with its default parameters we got the following output for Lie (figure 5):

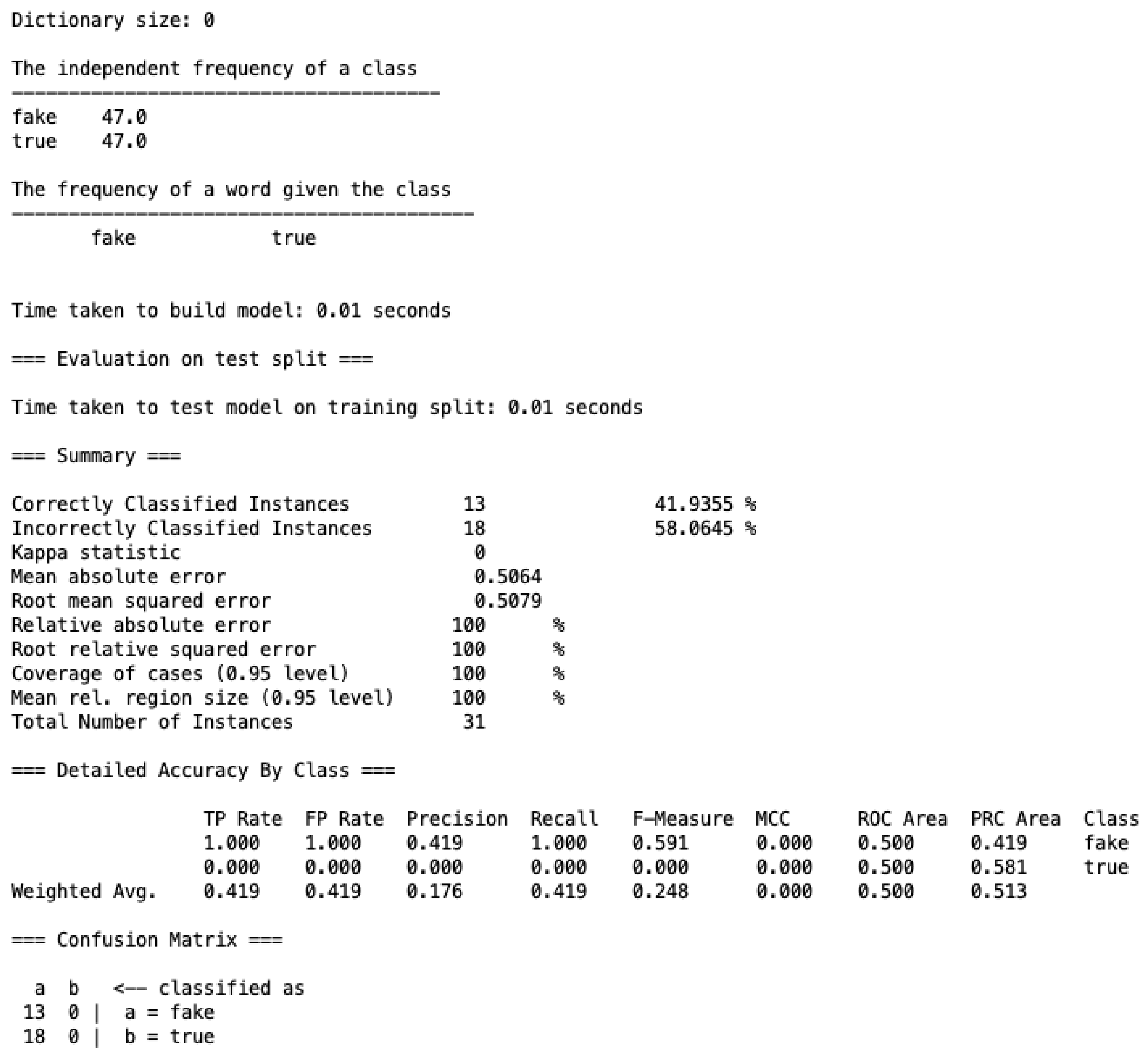


```
Classifier Model
Dictionary size: 0

The independent frequency of a class
--------------------------------------
fake    47.0
true    47.0

The frequency of a word given the class
-----------------------------------------
       fake            true


Time taken to build model: 0.02 seconds

=== Evaluation on test split ===

Time taken to test model on training split: 0.01 seconds

=== Summary ===

Correctly Classified Instances          13               41.9355 %
Incorrectly Classified Instances        18               58.0645 %
Kappa statistic                          0
Mean absolute error                      0.5064
Root mean squared error                  0.5079
Relative absolute error                100      %
Root relative squared error            100      %
Coverage of cases (0.95 level)         100      %
Mean rel. region size (0.95 level)     100      %
Total Number of Instances               31

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    1.000    0.419      1.000    0.591      0.000    0.500     0.419     fake
                 0.000    0.000    0.000      0.000    0.000      0.000    0.500     0.581     true
Weighted Avg.    0.419    0.419    0.176      0.419    0.248      0.000    0.500     0.513

=== Confusion Matrix ===

  a  b   <-- classified as
```

Figure 5

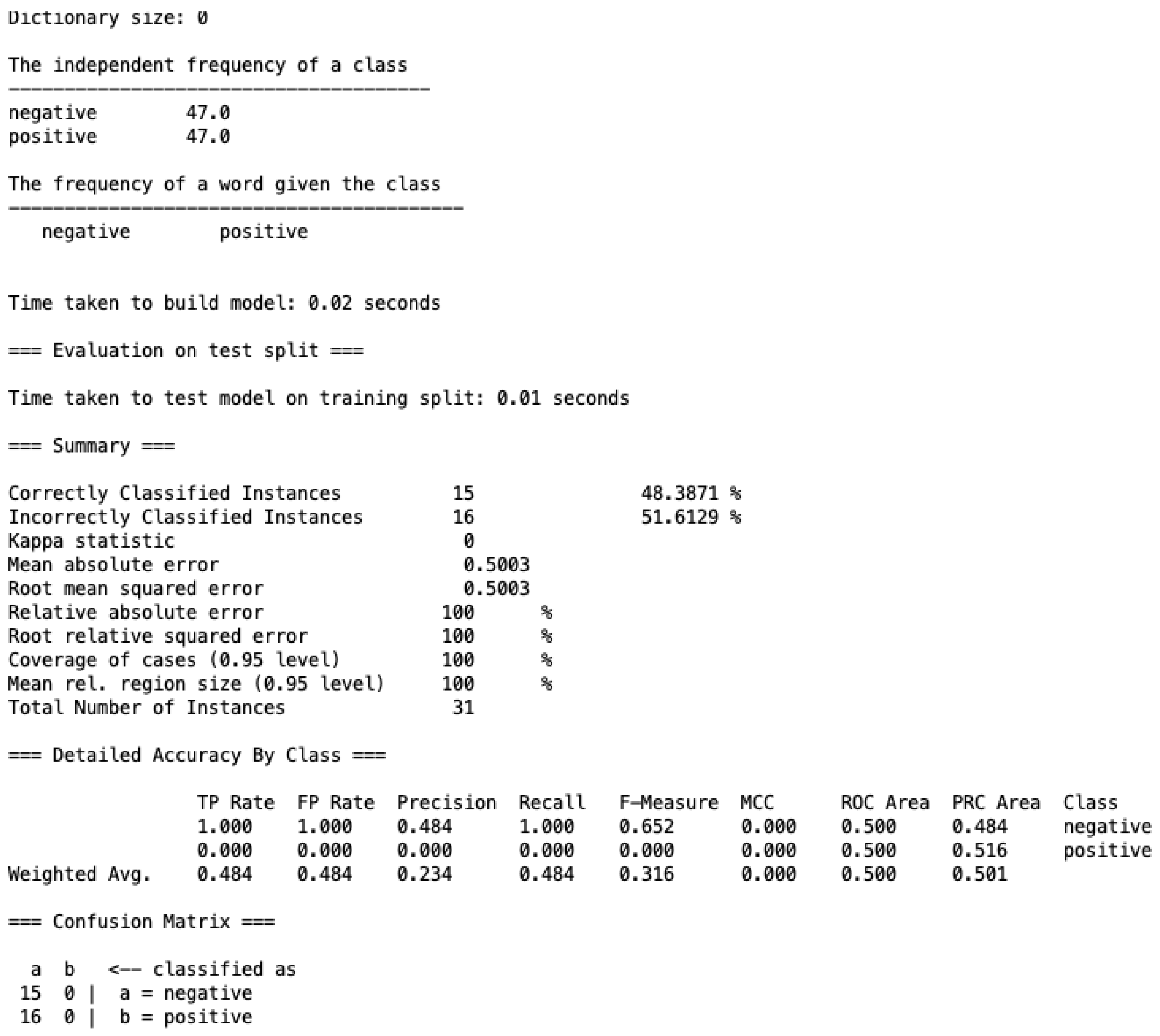
However, when we ran it a second time for Lie (figure 6) and sentiment (figure 7), we had tuned the parameters. We first chose to us the Word Tokenizer and add a “-“ to the delimiter.

Additionally we kept the stemmer to NullStemmer. Finally, we changed the MinWordFrequency to 1 and set the lowerCaseTokens to be TRUE.



```
Dictionary size: 0

The independent frequency of a class
--------------------------------------
fake    47.0
true    47.0

The frequency of a word given the class
-----------------------------------------
       fake            true


Time taken to build model: 0.01 seconds

=== Evaluation on test split ===

Time taken to test model on training split: 0.01 seconds

=== Summary ===

Correctly Classified Instances          13               41.9355 %
Incorrectly Classified Instances        18               58.0645 %
Kappa statistic                          0
Mean absolute error                      0.5064
Root mean squared error                  0.5079
Relative absolute error                100      %
Root relative squared error            100      %
Coverage of cases (0.95 level)         100      %
Mean rel. region size (0.95 level)     100      %
Total Number of Instances               31

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    1.000    0.419      1.000    0.591      0.000    0.500     0.419     fake
                 0.000    0.000    0.000      0.000    0.000      0.000    0.500     0.581     true
Weighted Avg.    0.419    0.419    0.176      0.419    0.248      0.000    0.500     0.513

=== Confusion Matrix ===

  a  b   <-- classified as
 13  0 |  a = fake
 18  0 |  b = true
```

Figure 6



```
Dictionary size: 0

The independent frequency of a class
--------------------------------------
negative        47.0
positive        47.0

The frequency of a word given the class
-----------------------------------------
   negative        positive


Time taken to build model: 0.02 seconds

=== Evaluation on test split ===

Time taken to test model on training split: 0.01 seconds

=== Summary ===

Correctly Classified Instances          15               48.3871 %
Incorrectly Classified Instances        16               51.6129 %
Kappa statistic                          0
Mean absolute error                      0.5003
Root mean squared error                  0.5003
Relative absolute error                100      %
Root relative squared error            100      %
Coverage of cases (0.95 level)         100      %
Mean rel. region size (0.95 level)     100      %
Total Number of Instances               31

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    1.000    0.484      1.000    0.652      0.000    0.500     0.484     negative
                 0.000    0.000    0.000      0.000    0.000      0.000    0.500     0.516     positive
Weighted Avg.    0.484    0.484    0.234      0.484    0.316      0.000    0.500     0.501

=== Confusion Matrix ===

  a  b   <-- classified as
 15  0 |  a = negative
 16  0 |  b = positive
```

Figure 7

Additionally, we have chosen to report the results in the table below. Results will be discussed in the Results section.

Parameter Setting	Overall Accuracy	Precision in Category I	Recall in Category I	Precision in Category II	Recall in Category II
Lie Detection	41.9355 %	0.419	1.000	0.000	0.000
Sentiment	48.3871 %	0.484	1.000	0.000	0.000

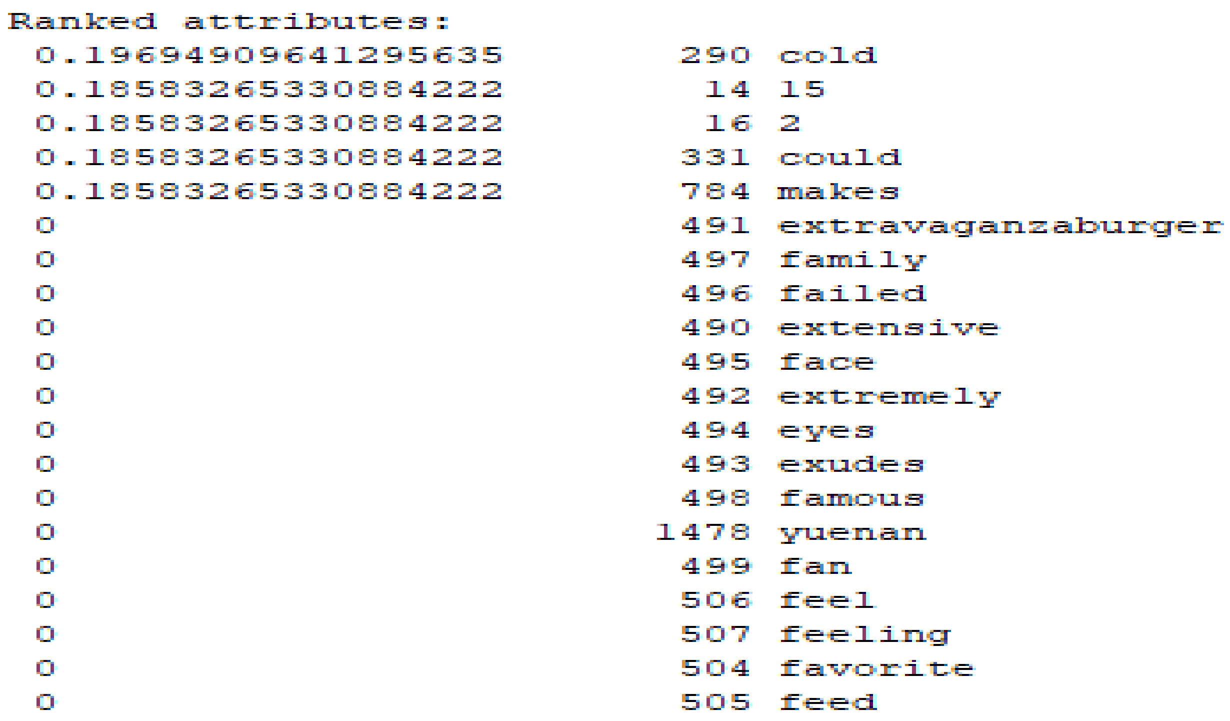
Parin Patel

GainRatio:

Used this website as resource:
http://weka.sourceforge.net/doc.dev/weka/attributeSelection/GainRatioAttributeEval.html

We used GainRatio attribute evaluator to rank the features. The higher the gain ratio for the attribute, the more useful the attribute will be for classification. Figure 8 is the output for lie while figure 9 is the output for sentiment.

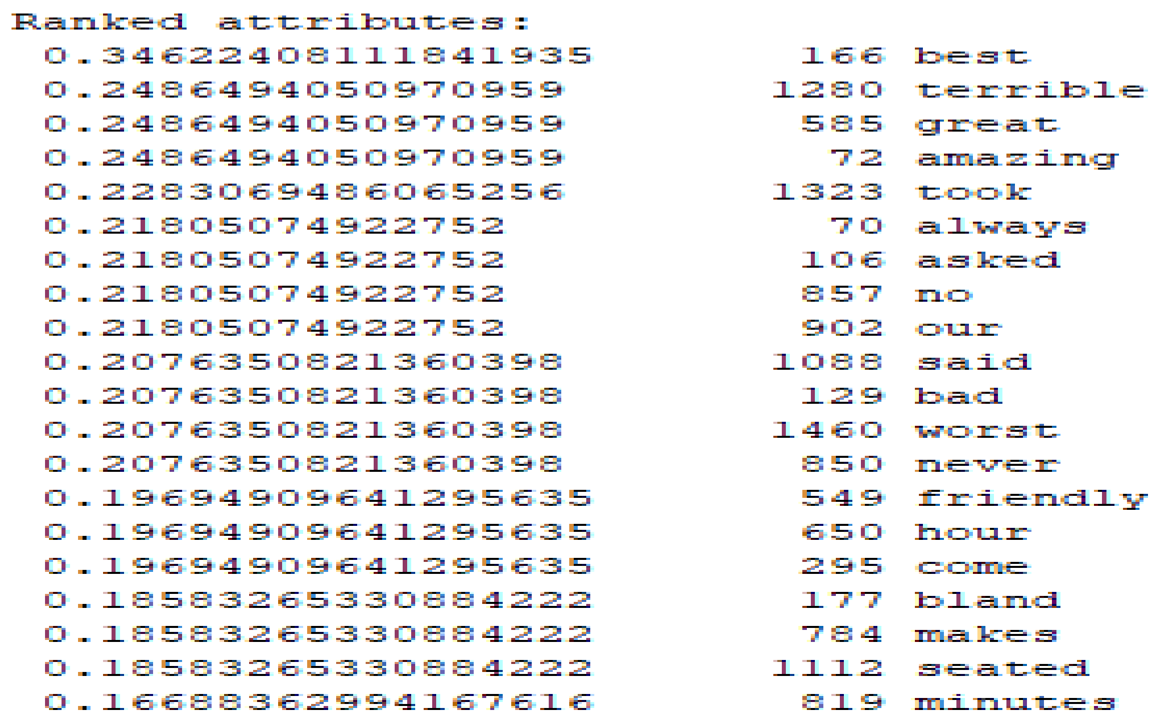
Lie Detection:



```
Ranked attributes:
 0.19694909641295635        290 cold
 0.18583265330884222         14 15
 0.18583265330884222         16 2
 0.18583265330884222        331 could
 0.18583265330884222        784 makes
 0                          491 extravaganzaburger
 0                          497 family
 0                          496 failed
 0                          490 extensive
 0                          495 face
 0                          492 extremely
 0                          494 eyes
 0                          493 exudes
 0                          498 famous
 0                         1478 yuenan
 0                          499 fan
 0                          506 feel
 0                          507 feeling
 0                          504 favorite
 0                          505 feed
```

Parin Patel

Sentiment:



```
Ranked attributes:
 0.34622408111841935         166 best
 0.248649405097095 9        1280 terrible
 0.2486494050970959          585 great
 0.2486494050970959           72 amazing
 0.2283069486065256         1323 took
 0.218050749227 52            70 always
 0.21805074922752            106 asked
 0.21805074922752            857 no
 0.21805074922752            902 our
 0.2076350821360398         1088 said
 0.2076350821360398          129 bad
 0.2076350821360398         1460 worst
 0.2076350821360398          850 never
 0.19694909641295635         549 friendly
 0.19694909641295635         650 hour
 0.19694909641295635         295 come
 0.18583265330884222         177 bland
 0.18583265330884222         784 makes
 0.18583265330884222        1112 seated
 0.16688362994167616         819 minutes
```

Additionally, we listed the top 10 attributes for sentiment below

Ranked attributes:

best
terrible
great
Amazing
Took
always
aske
no
our
said
bad
worst
never
friendly
hour
come

Parin Patel

Bland
Makes
Seated
minutes

Chi2

We used this website as a resource:
http://weka.sourceforge.net/doc.stable/weka/attributeSelection/ChiSquaredAttributeEval.html

Next, we ranked the features and listed the top20 results from using Chi2. Specifically we used the ChiSquaredAttribueEval, to measure the association between the word features and its category. If the attributes rank is the same, like “great”, “terrible”, and “amazing” then the classifier has learned to grouped them together.

Lie detection

```
Ranked attributes:
 6.419      290 cold
 5.287       14 15
 5.287       16 2
 5.287      331 could
 5.287      784 makes
 0          491 extravaganzaburger
 0          497 family
 0          496 failed
 0          490 extensive
 0          495 face
 0          492 extremely
 0          494 eyes
 0          493 exudes
 0          498 famous
 0         1478 yuenan
 0          499 fan
 0          506 feel
 0          507 feeling
 0          504 favorite
 0          505 feed
```

Sentiment

```
Ranked attributes:
25.556      166 best
19.403     1313 to
14.553     1416 we
13.538      864 not
12.494      585 great
12.494     1280 terrible
12.494       72 amazing
10.839      819 minutes
 9.976     1323 took
 8.762       70 always
 8.762      902 our
 8.762      857 no
 8.762      106 asked
 7.576     1460 worst
 7.576      850 never
 7.576      129 bad
 7.576     1088 said
 6.419      295 come
 6.419      650 hour
 6.419      549 friendly
```

Results:

Between our SVM and MNB models, the SVM had a higher overall level of accuracy and precision. The graph below compares the accuracy percentages between the two models. In both models the accuracy of lie detection was lower than that for sentiment. However, overall both the lie detection and sentiment accuracy levels were lower for MNB. This is likely because sentiment classification looks at the words and weights them on a scale that resembles negative to positive. This is unlike lie detection that looks at the word, as a whole, to detect if it's a lie. It is difficult to say if a word is a lie without external factors or additional attributes to help guide the model.

Additionally, we graphed precision below from category I to category II to show how both MNB models dropped to zero in category II. This was unlike the SVM precision rate that, interestingly, increased for lie detection. We think this increase, since it is very small, is likely due to error. The model could possibility have needed further tokenization and tweaks to see for sure if this upward trend is intentional or an error. After trying to have the models predict fake reviews, it is clear that the difficulty of the task is reinforced. I believe that spotting fake reviews from text alone is an immensely difficult task which would require much more work.

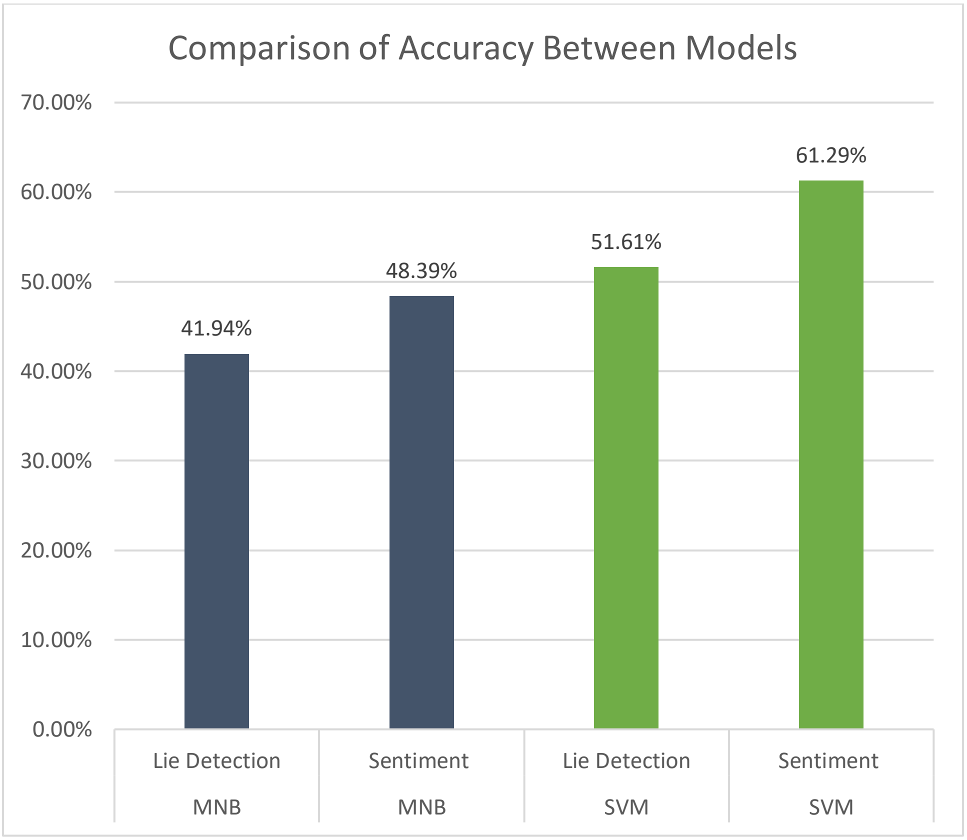


Figure 8

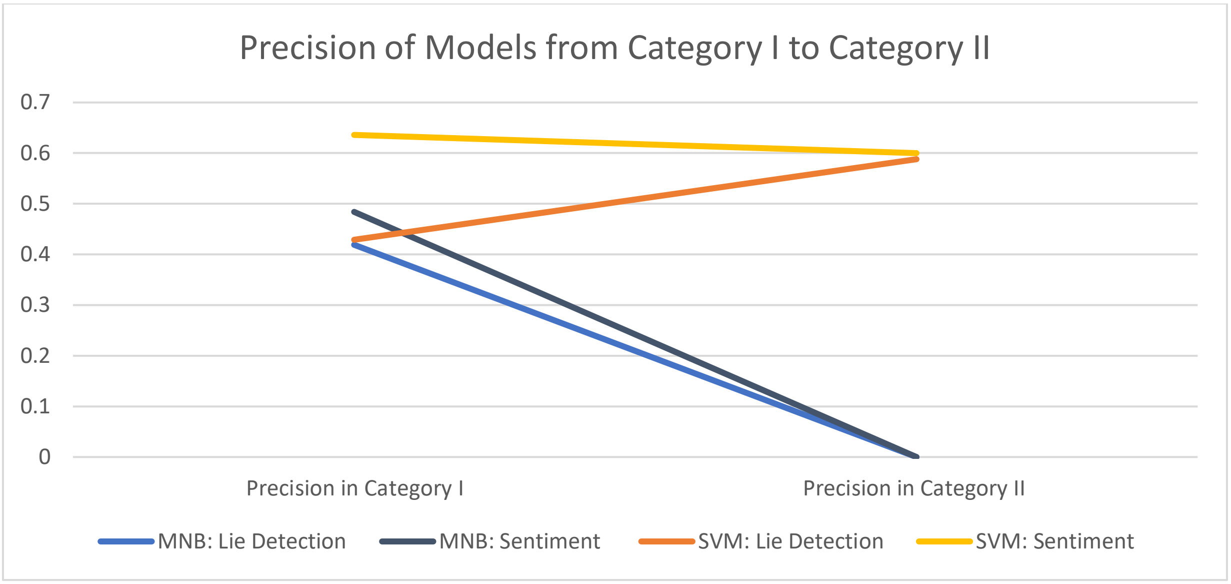


Figure 9

Additionally, it is important to note that for the MNB, tuning our parameters did not help us get better results. The default settings yielded the same results as our adjusted ones. However, SVM's outputs were more accurate we tuned the parameters accordingly.

When we looked at the Chi2 and GainRatio to rank the features, we noticed some interesting points. First, in the gain ratio, the lie detection seems to not have learned as much as the sentiment analysis. This is something we noticed earlier in our accuracy and precision outputs, however, we can see this trend again through our gain ratio attribute. This is important in highlighting the difficulty of detecting a lie by just looking at a word. Also, our sentiment analysis took into account opposite words like "best" and "terrible", and then it focused further on the negative words like "bad", "worst" and "never".

For our Chi2 attribute measure, we can see from our output that the attributes that have similar ranks, for example of12.495 for the attributes "great", "terrible" and "amazing", are likely to mean that they are classified together according to our model. This is likely because the model is grouping words with similar meanings.

Conclusion:

Overall, from our results, we can see a few major themes. First, it is more likely that we can use machine learning to try to detect sentiment rather than have it be used to detect lies. This is mainly because sentiment is easier for our algorithm to understand due to the nature of words and how they are structured in the English language. This is also why lie detection is more difficult, the algorithm is not able to take into consideration additional factors that usually occur when a person lies. For example, by just looking at a reviews word, the algorithm has no indication to the background or motive (maybe this reviewer is a competitor of the restaurant) of the person stating the words. It is possible that this could be beneficial when combined with the results of facial or phycological analysis of the review. However, right now, as a stand-alone lie detection algorithm, it is not probable.