Assignment 2: Sentiment Analysis in Twitter Messages Report

CSI4107: Information Retrieval and the Internet

For: Diana Inkpen

By: Martin Moncion 6146324

Jeremy Sivaneswaran 3116318



## Division of Work

|  |  |
| --- | --- |
| Train a Classifier | Martin |
| Add more features & classifiers | Martin & Jeremy |
| Report | Martin & Jeremy |
| Results file | Martin |

1.

For the first step we created our own arff file. We filtered out all of the stopwords with the stopwords text file from the first assignment as well as single letters. We manually went through the stopwords text file to remove any words that would obviously categorize a sentence to be positive or negative such as win or lose. Removing stopwords greatly reduced the size of the vector space. We managed to reduce the vector space from around 4000 attributes to 1050. To tokenize the words we used the StringToWord attribute filter in Weka to extract them. We also modified the delimiter to only extract .,'"?/-@%&=#<>[]\* from the data set in order to tokenize them. To improve our results we also removed rare words by setting the minimum frequency value to 2. So that way the word must appear a minimum of 3 times in the data set in order for it to tokenize. We kept the!:;() characters in order to search for emoticons in step 2.

2.

For part two of the assignment we used a couple of features to help the classification results. We had a list of positive and negative words and we counted each time they would occur in a twitter message. We used the LIWC resource available on the assignment website for our list of positive and negative words. We also counted the number of positive and negative emoticons as well as “!” marks. With all of this we also created another feature that would categorize our results into positive, negative, objective and neutral, where we would take the total number of positive words and emoticons found and subtract them by the total number of negative words and emoticons. If the sum of positive and negative words and emoticons were both 0 then the result is a neutral statement. If the subtraction of words resulted to zero then it is a neutral statement, otherwise it’s positive or negative based on which occurs more often. We found when we use the classification on our custom feature (feelingcategory) our results were much better since it was directly based off of our features and the data presented.

Step 1 results:

Base case for comparison without stopword removal: Naïve Bayes

=== Summary ===

|  |  |  |
| --- | --- | --- |
| Correctly Classified Instances | 3271 | 45.242 % |
| Incorrectly Classified Instances | 3959 | 54.758 % |
| Kappa statistic | 0.2198 |  |
| Mean absolute error | 0.2973 |  |
| Root mean squared error | 0.4345 |  |
| Relative absolute error | 85.842 % |  |
| Root relative squared error | 104.4266 % |  |
| Total Number of Instances | 7230 |  |

=== Detailed Accuracy By Class ===

TP Rate FP Rate Precision Recall F-Measure ROC Area Class

0.784 0.547 0.544 0.784 0.642 0.726 positive

0.309 0.086 0.439 0.309 0.362 0.726 negative

0.058 0.041 0.278 0.058 0.095 0.625 neutral

0.448 0.126 0.39 0.448 0.417 0.771 objective

Weighted Avg. 0.492 0.292 0.445 0.492 0.441 0.711

Base case for comparison without stopword removal: SMO

=== Summary ===

|  |  |  |
| --- | --- | --- |
| Correctly Classified Instances | 3801 | 52.5726 % |
| Incorrectly Classified Instances | 3429 | 47.4274 % |
| Kappa statistic | 0.2909 |  |
| Mean absolute error | 0.3176 |  |
| Root mean squared error | 0.4027 |  |
| Relative absolute error | 91.7119 % |  |
| Root relative squared error | 96.7841 % |  |
| Total Number of Instances | 7230 |  |

=== Detailed Accuracy By Class ===

TP Rate FP Rate Precision Recall F-Measure ROC Area Class

0.727 0.362 0.625 0.727 0.672 0.718 positive

0.391 0.101 0.458 0.391 0.422 0.731 negative

0.317 0.162 0.348 0.317 0.332 0.598 neutral

0.344 0.085 0.423 0.344 0.379 0.73 objective

Weighted Avg. 0.521 0.23 0.505 0.521 0.51 0.696

Base case for comparison without stopword removal: J48 Trees

=== Summary ===

|  |  |  |
| --- | --- | --- |
| Correctly Classified Instances | 3285 | 45.4357 % |
| Incorrectly Classified Instances | 3945 | 54.5643 % |
| Kappa statistic | 0.1863 |  |
| Mean absolute error | 0.291 |  |
| Root mean squared error | 0.462 |  |
| Relative absolute error | 84.0483 % |  |
| Root relative squared error | 111.0354 % |  |
| Total Number of Instances | 7230 |  |

=== Detailed Accuracy By Class ===

TP Rate FP Rate Precision Recall F-Measure ROC Area Class

0.668 0.393 0.586 0.668 0.624 0.671 positive

0.334 0.127 0.364 0.334 0.349 0.62 negative

0.233 0.162 0.281 0.233 0.255 0.55 neutral

0.288 0.114 0.313 0.288 0.3 0.626 objective

Weighted Avg. 0.457 0.253 0.439 0.457 0.446 0.629

Step 2 results: on the category classifier by implementing our filter modifications

Naïve Bayes

=== Summary ===

|  |  |  |
| --- | --- | --- |
| Correctly Classified Instances | 3644 | 50.4100% |
| Incorrectly Classified Instances | 3586 | 49.5989% |
| Kappa statistic | 0.2917 |  |
| Mean absolute error | 0.2688 |  |
| Root mean squared error | 0.4104 |  |
| Relative absolute error | 77.6305% |  |
| Root relative squared error | 98.6279% |  |
| Total Number of Instances | 7230 |  |

SMO

=== Summary ===

|  |  |  |
| --- | --- | --- |
| Correctly Classified Instances | 3770 | 52.1438% |
| Incorrectly Classified Instances | 3460 | 47.8562% |
| Kappa statistic | 0.2941 |  |
| Mean absolute error | 0.3189 |  |
| Root mean squared error | 0.4047 |  |
| Relative absolute error | 92.1047% |  |
| Root relative squared error | 97.2633% |  |
| Total Number of Instances | 7230 |  |

J48

=== Summary ===

|  |  |  |
| --- | --- | --- |
| Correctly Classified Instances | 3587 | 49.6127 % |
| Incorrectly Classified Instances | 3643 | 50.3873 % |
| Kappa statistic | 0.2326 |  |
| Mean absolute error | 0.2881 |  |
| Root mean squared error | 0.41 |  |
| Relative absolute error | 83.212 % |  |
| Root relative squared error | 98.5261 % |  |
| Total Number of Instances | 7230 |  |

However our best results came from creating our own classifier based on the data given. Those results are saved in the results directory they are called Results\_Bayes.txt, Results\_SMO.txt, Results\_J48.txt. Results\_J48.txt gave us the best results of the 3.

## Discussion

What led us to our best results was creating our own classifier called feelingcategory based on the data given to judge the results. We were able to get 97.704 for naïve bayes, 97.9391 % for SMO and 99.9308 % for j48. Feelingcategory used all of the feaures we created(positiveW, positiveE, negativeW, negativeE) that combined with the string to word vector filter that we modified also helped. We reduced the number of words to keep in the string to word to 1000. This reduced our vector space to about 1100 attributes. It also greatly increased the runtime and produced slightly better results. We were able to produce results in 15minutes rather than an hour and 15minutes.

## Program Details

Our program runs similarly to assignment 1. It’s written in java we downloaded the semeval\_twitter\_data.txt and saved as defaultMessages.txt in our data directory. Our program would read the file line by line and process the information. It starts by extracting the category and twitter message string and discards the rest. Once that is done the “ ’ ” character is remove in order to create a proper ARFF file, every single letter is removed as well.

The following snippet of code was used to remove stopwords.

**for** (String s : example.toLowerCase().split("\\b"))

{

**if** (!StopWordsList.contains(s)) result.append(s);

}

The following snippet of code demonstrates how we were able to find out how many positive and negative words and emoticons were in the string.

**for** (String s : wordexample.toLowerCase().split("\\s"))

{

//System.out.println(s);

**if** (PositiveWords.contains(s)) poscount++;

}

**for** (String s : wordexample.toLowerCase().split("\\s"))

{

**if** (NegativeWords.contains(s)) negcount++;

}

**for** (String s : emoticonexample.split("\\s"))

{

//System.out.println(s);

**if** (PositiveEmoticons.contains(s)) posemoticoncount++;

}

**for** (String s : emoticonexample.split("\\s"))

{

**if** (NegativeEmoticons.contains(s)) negemoticoncount++;

}

With this information we just needed to reconstruct are string into a format that Weka can accept. Here’s an example.

**int** positivefeeling = poscount + posemoticoncount;

**int** negativefeeling = negcount + negemoticoncount;

**if**(positivefeeling > negativefeeling)

{

example = "'" + example.toLowerCase() + " '," + classifier + "," + poscount+ "," + negcount+ ","+posemoticoncount+","+negemoticoncount+",positive";

}

## How to Run

Our program is an eclipse project file. If you have eclipse installed you will be able to build and run the code. The .arff file we used is saved in the Data folder called parsedData.arff. Just open the file in Weka run the StringtoWordVector filter with the settings we specified in step 2 and run the classifiers.