Assignment #2: Sentiment Analysis in Twitter Messages

CSI 4107 – Information and Retrieval from the Internet

Presented to:
Dr. Mohamad Hoda

By:
Alexandre Billard (6812210)
Qufei Chen (6771326)

School of Electrical Engineering and Computer Science
Faculty of Engineering

Introduction and Classifier Descriptions

For this assignment, decided to run a number of different classifiers to run our dataset through. The classifiers had the responsibility of predicting if a tweet was positive, negative, neutral or objective. The following classifiers were the chosen algorithms used to determine the category of the tweets.

A decision tree (J48)
This classifier will iteratively match with the best attribute to split on by using the method of gain ratio in order to overcome the bias to multi-valued attributes that tend to happen with information gain. The algorithm recurses and essentially splits the subsets by the next attribute with the greatest information gain to produce the following subsets. This algorithm also handles pruning the tree to limit overfitting, this can be illustrated with words such as "bad" which would be a good split point for determining a classification which would then decide whether or not to
branch further.

K-Nearest neighbor (IBk)
The K-Nearest neighbor classifier stores all available cases and classifies new cases using a distance function such as Euclidean, Hamming, Minkowski or Manhattan. This classifier is specified as lazy learning approach since it spends more time during testing than it does during training. It works by computing the k closest neighbor to each testing instance and assigns it to the class with the highest number of the closest neighbor. This classifier also holds the property of having k as an odd integer to avoid ties.

Naïve Bayes (NaiveBayes)
The Naïve Bayes classifier is based on Bayes Theorem which assumes class conditional independence. In this regard, we refer to the idea that this classifier assumes total independence
from its attributes and uses a probabilistic learning to classify instances. It calculates the probability that an instance is in a certain class given a certain feature, and does this for all features.

Support Vector Machines (SVM)
Support Vector Machines, or SVM for short is a classifier based on regulating the data following decision planes designed by decision boundaries. The Support Vector Machine classifier attempts to minimize the expected empirical loss on the training data, it works under the probabilistic assumption taken from previous examples. Essentially, SVMs create a maximum margin separator in the attempt of creating a decision boundary with the largest possible distance to the previous examples in order to generalize the next state.

Configuration and Techniques

For this lab, we ran all of our tests using Weka Explorer.

For **step 1**, we used the given data file (`semeval_twitter_data.arff`), and performed modifications such as tokenization, stop word removal, and stemming using the Weka Explorer interface. All tests use the StringToWord attribute filter in Weka to tokenize and extract words, and are evaluated using 10-fold cross validation. For the stop words list, we used the MultiStopWords list in Weka. For the Stemmer, we applied the Lovins Stemmer, also through the Weka interface.

For attribute selection in **step 2**, we added the attribute “`positive_negative_score`”, which we determined to be the number of positive words minus the number of negative words. We wrote a script to take the included subjectivity clues file (`subjclueslen1-HLTEMNLP05.txt`) to determine if a word in a tweet is positive or negative, and then took the difference of the number of positive words and negative words as a single attribute. In this step, we also converted emojis to their Unicode values and used them as tokens for the classification as well. The script `attribute_pos_neg.py` counts the number of positive and negative words, and convert emojis to Unicode. To run this program, simply modify the `data_path`, and `subjectivity_clues_path` variables to point to the correct path of the files on your system, and modify `output_file_name` to the path you wish to store the resulting arff file. Then run `python attribute_pos_neg.py` in the terminal. The resulting arff file (which we then used as an input for Weka Explorer) is named subj_data.arff.

For each of these data modifications (tests), we ran the data through all four of the classifiers listed above, and observed their accuracy, precision, recall, f-measure, and confusion matrix (see tables 2 and 3).

Results

We performed various tests on the data to determine which combination of methods resulted in the highest accuracy. Table 1 contains a breakdown of the tests and their corresponding parameters. The results can be found in Tables 2 and 3.

Table 1. Tests and their corresponding parameters

	Parameters (Yes/No)			
Test #	**Tokenization**	**Stop Word Removal (MultiStopWords)**	**Stemming (LovinsStemmer)**	**Attribute Selection**
1	Yes	No	No	No
2	Yes	Yes	No	No
3	Yes	Yes	Yes	No
4	Yes	Yes	Yes	Yes

Table 2. Tests and their accuracy with different classifiers

	Accuracy of Tests (%)			
Test #	**Naïve Bayes**	**SVM/SMO**	**Decision Trees/ J48**	**KNN**
1	45.1452	52.1577	45.4357	41.964
2	43.9004	50.4149	45.4357	41.964
3	45.2006	50.0277	45.4357	41.964
4	46.7773	49.7095	47.2337	41.2448

Complete results can be found in the following files: (where x = {1,2,3,4})

- test[x]_naivebayes.txt
- test[x]_svm.txt
- test[x]_decisiontrees.txt
- test[x]_knn.txt

Confusion Matrices + Precision, Recall, F-measure

The confusion matrix, precision, recall, and f-measure for the best result of each classifier are listed below. The complete results for each test can be found in the attached data folder.

Table 3: Confusion matrices, precision, recall, and f-measure of classifiers calculated by Weka

Classifier	**Confusion Matrix**	**Precision**	**Recall**	**F-measure**
Naïve Bayes (Test 4)	a b c d <-- classified as	0.438	0.468	0.440
	2289 377 186 432 | a = positive			
	600 437 97 159 | b = negative			
	761 267 170 349 | c = neutral			
	380 108 132 486 | d = objective			
SVM (Test 1)	a b c d <-- classified as	0.506	0.522	0.511
	2386 276 424 198 | a = positive			
	469 509 226 89 | b = negative			
	606 219 491 231 | c = neutral			
	352 101 268 385 | d = objective			
Decision Trees (Test 4)	a b c d <-- classified as	0.445	0.472	0.454
	2324 364 367 229 | a = positive			
	538 477 189 89 | b = negative			
	738 252 308 249 | c = neutral			
	415 107 278 306 | d = objective			
KNN (Test 1)	a b c d <-- classified as	0.373	0.420	0.380
	2368 216 505 195 | a = positive			
	835 139 242 77 | b = negative			
	935 127 325 160 | c = neutral			
	589 99 216 202 | d = objective			

Discussion

We ran a series of four tests, each with a different selection of features, on each of our four chosen classifiers. We obtained our best results through SVM on the original data set in test 1 (no stemming, no stop word removal, no additional attributes) with an accuracy of 52.1577%.

The range of the rest of our results are very similar (41.964% - 52.157%), across all the classifiers and variation of parameters/attributes. We also observed the scores to be quite low in comparison to acceptable standards. Stemming, removal of stop words, and even the inclusion of emojis and the number of positive and negative words (which should serve to increase the accuracy) seem to have no effect on raising the accuracy of the tests. Possibilities for these results will be discussed further in the Error Analysis section below.

The complete results of our test1 using SVM are stored in `results.txt`.

Error Analysis

One possible explanation for why the scores are so low is that we used the built-in tokenization from the StringToVector attribute in Weka Explorer instead of tokenizing the data ourselves. Perhaps the default tokenizer in Weka does not tokenize the words in the most ideal way, which affected the accuracy. We could have tokenized the tweets ourselves for a better result.

Another explanation could be the usage of the Lovins Stemmer and the stop word removal using MultiStopWords. These functions are built into Weka, but we could not verify the performance of them before usage. If we had implemented our own stemmer or used a hand generated stop words list, the results could have been improved.

Another possibility for error is our usage of Weka Explorer. After running our tests, we realized that there are multiple ways/places to apply StringToVector, stemming, and stop word removal. There may have been an error in usage where we applied the functions at the wrong place, or used a method that affected the accuracy.

Division of Tasks

Alex	Qufei
- Introduction - Classifier Descriptions - Decision Tree Tests and Results - KNN Tests and Results	- Configuration and Techniques - Discussion - Naïve Bayes Tests and Results - SVM tests and Results - attribute_pos_neg.py script