UNIVERSITÀ DI PISA

Sentweet

Business Intelligence project

A Weka based twitter sentiment analysis tool

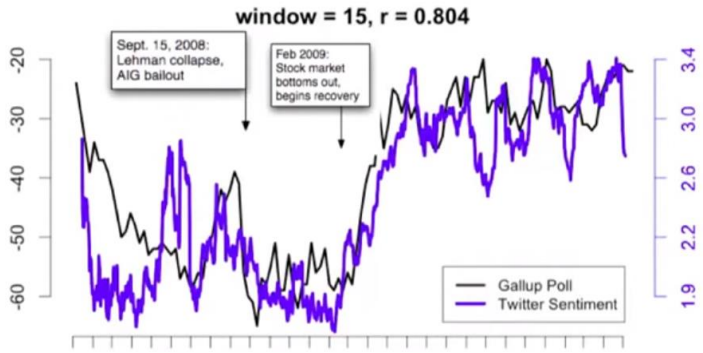
Egidi Sara A.A. 2015/16

1. Introduction

This project encompasses the analysis of text classification in a social context. The aim of this study is to classify a short text into either positive or negative, depending on the emotion of the writer. More specifically, the texts taken into analysis are tweets retrieved directly from Twitter.

Motivation

Sentiment analysis can be applied in many contexts. For example, regarding specific products, one may want to know facts about a particular printer by knowing what the customers say about it. Thus, this process may be aided by providing the user with some form of aggregation of reviews that shows how the people *feel* about that printer, easing her from manually reading all the feedbacks before taking a decision. Other works such as [2] show how twitter sentiment can be used to predict and measure consumer confidence over time (Fig 1), suggesting that we can use twitter to measure public opinion. Moreover, in [3] Bollen shows how Twitter may also be used to predict the stock market, and how their calmness index predicts the Dow Jones with a 3 days offset.



This document reports all the analysis phases carried out during the project, and those steps can be thought as sequential ones. In chapter 2, a brief description of the dataset chosen and how it was preprocessed prior to the analysis. Chapter 3 describes the filters applied to the training set ant their optimization, while chapter 4 contains a survey of the classifier tested and their comparison. Finally, in chapter 5 and 6 the implementation of the actual application is reported and commented.

2. Dataset and preprocessing

Many corpus are available online, each of which consists of labeled tweets. However, most of the datasets are not hand labeled, the label is based merely on the smiles used in the tweet body. The chosen dataset is the STS (Stanford Twitter Sentiment), which contains 40000 classified tweets, each row is marked as 4 for positive sentiment and 0 for negative sentiment.

The raw data consisted of a csv of 2 columns: Id, polarity id, source, tweet.

```
4;;1898408556;;Sat May 23 18:06:34 PDT 2009;;edrinaperez;;Millersfield. Watchin game with Coco :)
4;;1898408606;;Sat May 23 18:06:34 PDT 2009;;Velouette;;back with my babes!  spent a sunny day playing in the
park. tomorrow we attack the green jungle in the back garden! hugs to ya'll...
4;;1898408634;;Sat May 23 18:06:34 PDT 2009;;alcoholharmony;;@OrigSupawoman girl, now you know that's a lie. I
would never ^_^
0;;1835338065;;Mon May 18 05:45:29 PDT 2009;;;VioletsCRUK;;Im goin 2 hane 2 have a marathon tweeting session
this week if im goin to be Twitter-barred all weekend! From Fri 1/time till Mon Evenin!
0;;1835338356;;Mon May 18 05:45:32 PDT 2009;;;gwennibee;;is trying to revise for her maths exam but it isn't
going well :(
```

The preprocessing of the dataset was pretty straightforward, since the dataset is already balanced and somehow clean.

The only adjustment needed was some noise reduction, like the removal of the URLs, mentions and quotes. Since many (more than half) of the tweets contain smiles, which are strongly related to the polarity of the tweet, while preprocessing the dataset they were removed to not mislead the classification accuracy: since both the training and the test sets are composed by mainly tweets with smiles, and the accuracy results in a deceiving 95%.

During the analysis, it came to light that many of the users make use of exaggerating sentences by repeating many times the same letter. (e.g. “happppppyy”) For this reason, another adjustment was made, by “trimming” the letters that are repeated more than 3 times to only 2 times using a regex expression.

```
result = item.replaceAll("(.)\\1{2,100}", "$1$1");
```

Following the guidelines of ThinkNook[1], also a list of stopwords was used. Weka provides a default stopword list, which was modified and adapted to better fit the classification problem. For example the keywords “_USERNAME_” “_URL_” “rt” were added. In the figure below, an example of some final preprocessed tweets.

```
4,' back with my babes spent a sunny day playing in the park tomorrow we attack the green jungle in the back
garden hugs to ya\'ll'
4,'  girl now you know that\'s a lie i would never'
0,' im goin  hane  have a marathon tweeting session this week if im goin to be twitter-barred all weekend from
fri 1/time till mon evenin'
```

Anyways, the Stanford dataset is extracted from tweets from year 2010. This brings some inaccuracy, since the language people use, especially on social network, and the opinion and phrases they post actually evolve with time. Trends or even new products change the way we communicate, and thus the prediction made on old data may not be entirely reliable.

For this reason a further analysis was carried out, by actually generating a dataset from scratch. This was done by using the smiles contained in the retrieved tweets as labels, and extracting tweets concerning different subjects. The resulting corpus was then cleaned and tested with the same classifiers used with the STS Dataset, with good results. As described by [4] and [5], emoticons can be safely used as a good estimation of the class labels for this kind of analysis.

3. Filters

Once the dataset has been preprocessed into an ARFF file, the text of each tweet needs to be transformed in to a word vector as a first step towards feature selection. To do so, the *StringToWordVector* filter in Weka comes to help, embedding many other powerful tools for NLP.

3.1. StringToWordVector

This filter basically tokenizes a string of text into a set of attributes representing word occurrence information from text contained in the strings. This set of attributes is determined by the first batch filtered, which usually consists of the training data (See section 4.1).

As previously stated, this filter has many capabilities. Each available parameter was analyzed and tuned, resulting in the following configuration

Tf-idf

> This allows TF-IDF weights to be taken into account to reflect how important a word is to a specific tweet in the corpus. One issue encountered while performing tf-idf weighting on tweets is the short, constrained nature of tweets. This creates an upper limit on the Term Frequency, reducing the importance of that portion of the weighting scheme. Another issue is the enormous vocabulary size of Twitter users. New words are generated all the time, hashtags and handle references are thrown about, making the representation of Tweets in the form of tf-idf vectors problematic. At the same time, it's much harder to gauge word salience when the length of a tweet is constrained to 140 characters. This limit doesn't allow for much repetion of words, thereby drastically reducing the variation on the term frequency portion of tf-idf. For those reasons both the options were set as **false**.

OutputWordCount

> This option allows to keep track of how many times a word appears rather than keeping a Boolean. It was kept as **false**, since a tweet is a rather small message and thus the words are seldom repeated many times inside the same one.

Stemmer

The stemmer is used to reduce inflected words to their word stem (root). After a few experiments, **IteratedLovinsStemmer** proved to be the best performing one, increasing by 3% the overall accuracy.

Stopwords

Not all the words are meaningful when it comes to sentiment analysis. For this reason the StringToWordVector filter also allows to add a predefined list of words to ignore when extracting the attributes. The list comes by default with weka, but the classifier performed better by leaving the stopword list enabler as **false**. This may be justified by the fact that many tweets are composed with quite a small number of words, sometimes even 2. Since the tokenizer also takes bi and trigrams, removing words can actually penalize the filter.

Tokenizer

As a last option, several tokenizers have been tested. When using the **1 word tokenizer** the attributes consist of just one word and the classifier proved to have a good accuracy. The accuracy decreased when using bigrams and stayed the same when using unigrams and trigrams together.

3.2. Attribute Selection

A major issue with text categorization problems is the high dimensionality of the feature space. The feature space resulting from the above filter consists of around 1000 unique terms that occur in the tweets. To achieve dimensionality reduction, the filter *AttributeSelection* was applied. This filter can use different evaluators and search algorithms. For this analysis, *InformationGain*, *CfsSubsetEval* and *ChiSquare* were used, setting as search policy both *BestFirst* and *Ranker*. The chosen evaluator is **InformationGain**, with **ranker** as search method.

4. Classifiers

Once the filters have been accurately tuned, the dataset may actually be classified. To do so, the filters should be applied just to the training set and not on test one, and the filtered classifier can be used for this purpose.

4.1. FilteredClassifier

The filtered classifier is a meta-classifier that allows to use only the training set class info when learning with cross validation, so that for each fold the test set doesn't affect the learning process. Without using this method, the chosen classifier would learn and tokenize using also the test set and this would be like "cheating". Weka's implementation of this classifier encompasses a classifier and a filter: different classifiers have been tried, and the filter used is a **MultiFilter** that chains the StringToWordVector and AttributeSelection with the parameters discussed earlier.

In the table below, a report of each classifier accuracy. Each classifier was tested with a 10 fold cross validation using the same filtered classifier, except for the Naïve Bayes Multinomial text which doesn't need it (Section 4.2).

Classifier	TP	FP	Accuracy	Roc Area
Naïve Bayes	Neg 0.717	0.339	66.88%	0.756
	Pos 0.661	0.283		
Naïve Bayes Multinomial	Neg 0.788	0.277	75.72%	0.834
	Pos 0.723	0.212		
Naïve Bayes Multinomial Text	Neg 0.769	0.231	76.89%	0.839
	Pos 0.769	0.231		
SMO (SVM)	Neg 0.737	0.203	76.69%	0.767
	Pos 0.797	0.263		
J48	Neg 0.723	0.271	72.06%	0.762
	Pos 0.729	0.277		
K-NN	Pos 0.634	0.313	66.05%	0.698
	Neg 0.687	0.366		

NaiveBayesMultinomialText (only on Weka 3.8)

This classifier has been introduced with only recently with Weka 3.8. Many tests have reported impressive results, although it doesn't use any kind of attribute selection. This classifier includes the string to word vector filter, thus simplifying the process.

As a final test for assessing the best classifier, using the Weka's experimenter, the t-test was carried out using both datasets. The selected classifiers are: Naïve Bayes Multinomial, Naïve Bayes Multinomial Text and SVM. The significance level used is 0.05.

NBMT proved to be better than the SVM, considering both AUC and accuracy.

```
Tester:     weka.experiment.PairedCorrectedTTester -G 4,5 -D 1
Analysing:  Percent_correct
Datasets:   1
Resultsets: 3
Confidence: 0.05 (two tailed)
Sorted by:  -
Date:       4/27/16 10:40 PM

                              (SVM)        (MNB)     (MNBT)
Dataset                   (1) meta.Fil | (2) meta. (3) bayes
----------------------------------------------------------
Preprocessed1             (20)   76.36 |   74.41 *   77.12 v
----------------------------------------------------------
                              (v/ /*) |   (0/0/1)    (1/0/0)


Tester:     weka.experiment.PairedCorrectedTTester -G 4,5 -
Analysing:  Area_under_ROC
Datasets:   1
Resultsets: 3
Confidence: 0.05 (two tailed)
Sorted by:  -
Date:       4/27/16 10:45 PM

                              (SVM)        (MNB)     (MNBT)
Dataset                   (1) meta.Fi | (2) meta (3) baye
----------------------------------------------------------
Preprocessed1             (20)   0.76 |   0.82 v   0.84 v
----------------------------------------------------------
                              (v/ /*) |   (1/0/0)   (1/0/0)
```

By using the generated dataset, the accuracy of the model reaches its best at 78%, thus this is the model chosen for the final implementation.

```
                                -
       10.0     <laplace=1>     i wish w
<laplace=1>            3.0     find out how
        3.0     <laplace=1>     am pl
        3.0     <laplace=1>     she need
        3.0            11.0     if you don
        4.0     <laplace=1>     i wish m
        3.0     <laplace=1>     i wish n
        3.0     <laplace=1>     wish i liv
<laplace=1>            25.0     now for som


Time taken to build model: 7.77 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances       15625               78.125  %
Incorrectly Classified Instances      4375               21.875  %
Kappa statistic                          0.5625
Mean absolute error                      0.2418
Root mean squared error                  0.4169
Relative absolute error                 48.368  %
Root relative squared error             83.3723 %
Total Number of Instances            20000

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.844    0.281    0.750      0.844    0.794      0.567    0.851     0.828     0
                 0.719    0.156    0.821      0.719    0.767      0.567    0.851     0.865     4
Weighted Avg.    0.781    0.219    0.786      0.781    0.780      0.567    0.851     0.846

=== Confusion Matrix ===

    a    b   <-- classified as
 8436 1564 |    a = 0
 2811 7189 |    b = 4
```

5. Implementation

Before making use of the filtered classifier with a 10-fold cross validation, all the single steps of the classification were tested in sequence.

First of all a pre-processing class takes care of the cleaning of the raw txt file, extracting just the useful information (i.e. polarity and text) and saves the output as an ARFF file.

```
public void preprocessRaw(String path, String outputPath){
       Preprocessor pre = new Preprocessor();
       String strLine = null,str,pol;
       List<String> result = new LinkedList<String>();
       try{

      FileInputStream fstream1 = new FileInputStream(path);
      DataInputStream in = new DataInputStream(fstream1);
      BufferedReader br = new BufferedReader(new InputStreamReader(in));;

      while ((strLine = br.readLine()) != null) {
       String[] items = strLine.split(";;");
       str = items[5].toLowerCase();
             pol = items[0];
         result.add(pol+","+Utils.quote(pre.preprocessDocument(str)));
      }
      in.close();
  }catch (Exception e){
      System.err.println("Error while parsing the input raw file: " + e.getMessage());
  }
       try {
             PrintWriter writer = new PrintWriter(new FileWriter(outputPath));
             ListIterator<String> itr = result.listIterator();
             String temp;
             writer.print("@relation tweets\n\n");
             writer.print("@attribute classPol {0,4}\n");
             writer.print("@attribute text String\n\n");
             writer.print("@data\n");
             while (itr.hasNext()) {
          String element = (String) itr.next();
          temp = element.replaceAll("_SMILEHAPPY_", "");
          temp = element.replaceAll("_SMILESAD_", "");
          temp = element.replaceAll("USERNAME", "");
          writer.print(temp + "\n");
      }
             System.out.println("Saved preprocessed dataset "+ outputPath);
             writer.close();
       }
       catch (IOException e) {
             System.out.println("Problem found when writing " + outputPath);
       }
       System.out.println(Arrays.toString(result.toArray()));

       System.out.println("Preprocessing completed");
}
```

When parsing the file, the document is also pre-processed by another preprocessor class method. The pre-processor carries out all the steps defined in the previous chapter:

```
public String preprocessDocument(String item) {
	String result_fin = "";
	String result = item;
	StringTokenizer st1 = new StringTokenizer(result, " ,?![]");
	while (st1.hasMoreTokens()) {
		String str = st1.nextToken();
		result = removeUrl(str);
		if(!_opt.isRemoveEmoticons())
			result = replaceEmoticons(result);
		else
			result = removeEmoticons(result);
		StringTokenizer st2 = new StringTokenizer(result, ".:#)(_");
		String tmp = "";
		String tmp2 = "";
		while (st2.hasMoreTokens()) {
			tmp = st2.nextToken();
			tmp = recognizeLaugh(tmp);
			tmp2 = tmp2 + " " + removeUsername(tmp);
		}
		result = tmp2;
		result = result.replaceAll("lu+v+", "love");
		result = removeRepeatedCharacters(result);
		result = removeSymbols(result);
		result_fin = result_fin + result;
	}
	return result_fin;
}
```

The FilteredClassifierBuilder class method actually builds the classifier using the string to work vector and attribute selection

```
public void buildClassifier() {
		try {
			inputInstances.setClassIndex(0);
			// string2WordVector - tokenizer
			NGramTokenizer tokenizer = new NGramTokenizer();
			tokenizer.setNGramMinSize(1);
			tokenizer.setNGramMaxSize(1);
			tokenizer.setDelimiters("\\W");

			StringToWordVector s2wv = new StringToWordVector();
			s2wv.setTokenizer(tokenizer);
			s2wv.setWordsToKeep(1000000);
			s2wv.setDoNotOperateOnPerClassBasis(true);
			s2wv.setLowerCaseTokens(true);
			s2wv.setUseStoplist(true);
			s2wv.setStopwords(new File("stopwords.txt"));
			s2wv.setInputFormat(inputInstances);

			// Filter the input instances into the output ones - used for debug
			//outputInstances = Filter.useFilter(inputInstances,s2wv);
			//System.out.println("Filtering finished");

			// attribute selection
			AttributeSelection attSel = new AttributeSelection();
			InfoGainAttributeEval eval = new InfoGainAttributeEval();
			Ranker search = new Ranker();
			search.setThreshold(0);
			attSel.setEvaluator(eval);
			attSel.setSearch(search);
```

```
                // classifier
                Classifier cModel = (Classifier)new NaiveBayes();
                //cModel.buildClassifier();

                // filter
                MultiFilter mf = new MultiFilter();
                Filter Filters[]={s2wv,attSel};
                mf.setFilters(Filters);

                // meta-classifier
                FilteredClassifier fc = new FilteredClassifier();
                fc.setFilter(mf);
                fc.setClassifier(cModel);

                // train and make predictions
                fc.buildClassifier(inputInstances);

                // save the built model to file
                Debug.saveToFile("/data/WEKA_Models", fc);
        }
        catch (Exception e) {
                System.out.println("Problem found when training"+e);
        }
}
```

The classifier is finally built and its model saved

```
long time1, time2;
FilteredClassifierBuiler fcb = new FilteredClassifierBuiler();
fcb.preprocessRaw("train.txt","outputparser.arff");
fcb.loadARFF("outputparser.arff");
time1 = System.currentTimeMillis();
System.out.println("Started building model at: " + time1);
fcb.buildClassifier();
time2 = System.currentTimeMillis();
System.out.println("Finished building model at: " + time2);
System.out.println("Total building time: " + (time2-time1));
System.out.println("Model saved at /data/WEKA_Models ");
```

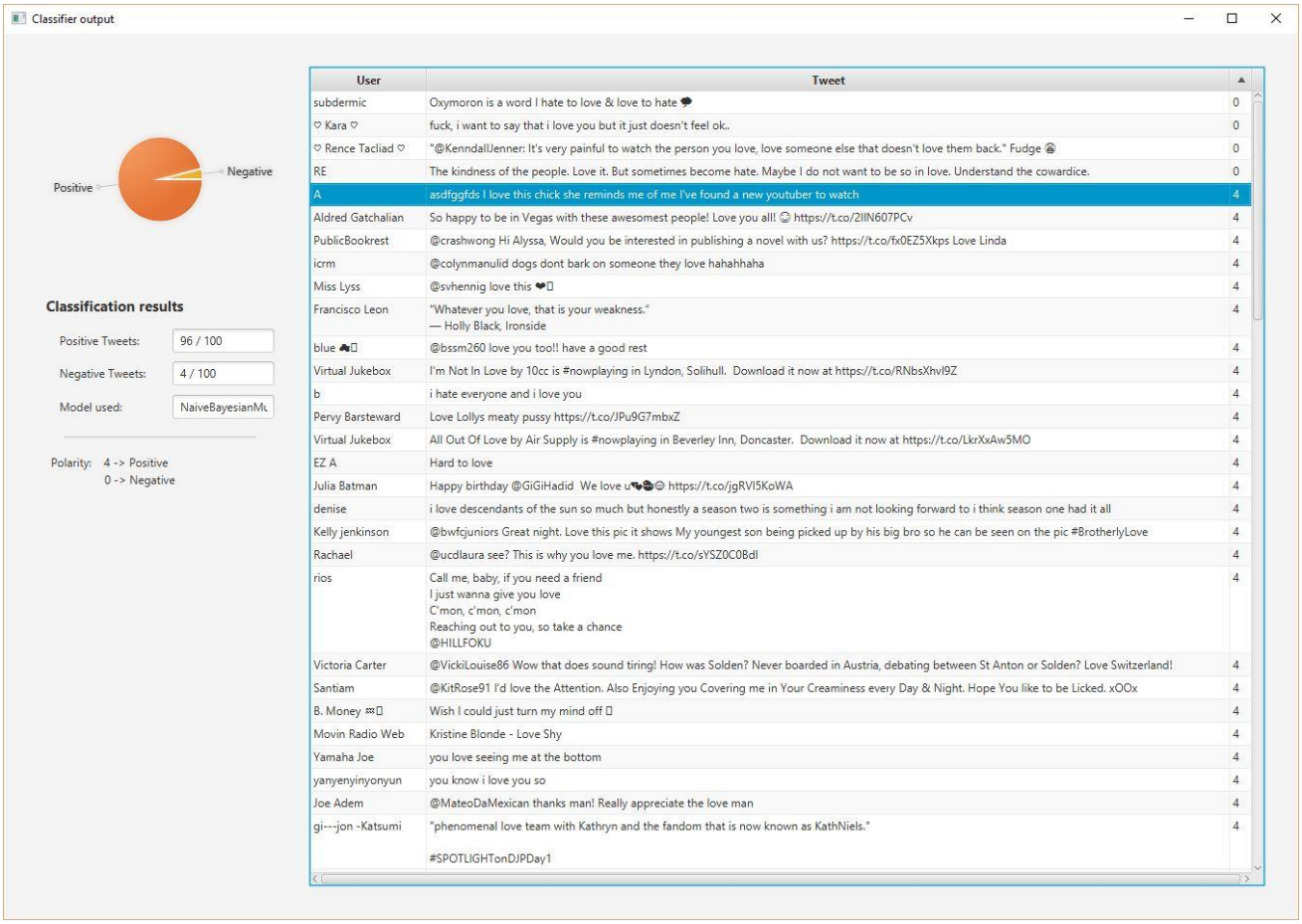
External libraries and tools were used apart from the Weka ones, to allow a user friendly interface and tweet retrieval.

JavaFX: tool that allows to use scenes as forms to create a user friendly GUI.

Twitter4J & TwitterAPI: java library that integrates java application with Twitter Service

6. Application

The final application consists of two main screens, in the first one the user inputs his search term and retrieves the tweets, which will also be preprocessed. Once the ARFF has been created, the user can visualize the results.



The code is publicly available on GitHub: https://github.com/egidisa/Sentweet

References

[1] – ThinkNook, Improving text classification algorithm - http://thinknook.com/10-ways-to-improve-your-classification-algorithm-performance-2013-01-21/

[2] – Brendan O'Connor et al, From Tweets to Pools: Linking Text Sentiment To public Opinion Time Series, 2010

[3] – Johan Bollen et al, Twitter mood predicts the stock market, 2011

[4] - Twitter sentiment classification using distant supervision, A Go, R.Bhayani, L.Huang, 2009

[5] - Sentiment analysis of twitter data, A. Agrawal, 2011

Kathy Lee, Twitter Trending Topic Classification, 2011

Twitter sentiment analysis: The good the bad and the omg!, E. Kouloumpis, T.Wilson, JD Moore, 2011