Preprocessing Egyptian Dialect Tweets for Sentiment Mining

Amira Shoukry, Ahmed Rafea
Department of Computer Science and Engineering
The American University in Cairo
Cairo, Egypt
am_magdy@aucegypt.edu, rafea@aucegypt.edu

Abstract

Research done on Arabic sentiment analysis is considered very limited almost in its early steps compared to other languages like English whether at document-level or sentence-level. In this paper, we test the effect of preprocessing (normalization, stemming, and stop words removal) on the performance of an Arabic sentiment analysis system using Arabic tweets from twitter. The sentiment (positive or negative) of the crawled tweets is analyzed to interpret the attitude of the public with regards to topic of interest. Using Twitter as the main source of data reflects the importance of the system for the Middle East region, which mostly speaks Arabic.

Keywords-component; Sentiment; Feature; Tweets; Polarity, Stop-words, Stemming, Normalization

1. Introduction

Sentiment analysis has recently become one of the growing areas of research related to text mining and natural language processing. Due to the increasing availability of online resources and popularity of rich and fast resources for opinion sharing like news, online review sites and personal blogs, several parties such as customers, companies, or even governments started to analyze and explore these opinions. Generally, we can say that determining the writer's attitude regarding some topic or the overall tonality of the text is considered the main task of sentiment analysis. In this paper, we are interested in the effect of the preprocessing stage on the performance of the sentiment classification process for the Arabic language at the sentence level in which the aim is to classify a sentence whether a blog, review, tweet, etc… as holding an overall positive or negative attitude concerning the given topic. It is important to mention that this work is part of a project that will include extracting sentiment topic and other features.

The fields of text mining and information retrieval for the Arabic language had been the interest of many researchers and various studies have been carried in these fields resulting in diverse resources, corpora, and tools available for implementing applications like text classification (Duwairi, 2009) or named entity recognition (Shaalan and Raza, 2009). However, most of the research done in these fields was focused on English texts with very limited research done for other languages such as Arabic (Elhawary and Elfeky, 2010), particularly the Egyptian dialect which is the language of interest for this research. Although Arabic is considered from the top 10 languages mostly used on the Internet based on the ranking carried out by the Internet World State rank in 2010[1] and it is spoken by hundreds of millions of people, there exist very limited annotated resources for sentiment analysis such as labeled corpora, and polarity lexica. This could be considered the main reason which had motivated the generation of an opinion corpus for Arabic in this work.

The majority of the text produced by the social websites is considered to have an unstructured or noisy nature. This is due to the lack of standardization, spelling mistakes, missing punctuation, nonstandard words, repetitions, etc… (Al-Shammari, 2009). That is why the importance of preprocessing this kind of text is attracting the attention these days especially with the presence of several websites producing noisy text. There are mainly three steps in the preprocessing process: 1) normalization, 2) stemming, and 3) stop words removal. Normalization is the process of transforming the text in order to be consistent, thus putting it in a common form. On the other hand, stemming is the process of reducing words to

[1] http://www.internetworldstats.com/

their uninflected base forms. Sometimes the stem is different from the root, but it is useful as usually related words map to the same stem even if this stem is not in itself a valid root. And finally, the stop words removal which is the process of removing those words which are natural language words having very little meaning, such as "في" (in), "علي" (on), "انت" (you),"من" (of), and similar words.

The approaches for sentiment classification are: machine learning (ML) and semantic orientation (SO). The ML approach is a supervised approach where data marked with its class (positive or negative) are used as training data by the classifier implying that a combination of particular features yields a particular class (Morsy and Rafea, 2012) using one of the supervised categorization algorithm like Naïve Bayesian Classifier, Support Vector Machine (SVM), Maximum Entropy, etc… In contrast, the SO approach is mainly an unsupervised approach in which a sentiment lexicon is built with the class of each word is inferred by a number indicating its semantic intensity. Then, all the sentiment words in the sentence are extracted using this lexicon and their polarities are summed up to determine if the sentence has an overall positive or negative sentiment (Morsy and Rafea, 2012). In this study we will be testing the effect of the proposed preprocessing steps on both ML and SO approaches.

The remaining of the paper shows in more details our achieved work in the preprocessing of the Arabic tweets for analyzing and extracting their sentiments. Section II summarizes the work done in the preprocessing stage of most Arabic sentiment mining systems, which is our focus in this study, while section III proposes the system architecture and discusses the system implementation details. Section IV describes the experiments conducted and their results. Finally, Section V talks about the challenges, conclusion and future work.

2. Related Work

Firstly is the normalizing stage which is putting the Arabic text in a consistent form. A normalizer[1] is implemented for doing this job using Ruby. This normalizer performs several tasks such as removing diacritics from the letters, removing 'ء' (Hamza), making both 'ي' and 'ى' change to 'ي'(y), etc…

[1] http://arabtechies.sourceforge.net/projec/ normalization _ruby

Secondly is the stemming stage which is considered one of the most important stages in any Arabic information retrieval or text mining systems. Stemming Arabic terms has proven in several researches that it is not an easy task because of its highly inflected and derivational nature (Larkey. 2007). There are mainly two classes of stemmers for the Arabic language: aggressive stemmers (reducing a given word to its root) and light stemmers (identifying a set of prefixes and suffixes that will be removed). The authors in (Khoja and Garside, 1999) developed an aggressive stemmer which reduces the words to their roots. Their stemmer removes all the punctuation marks, diacritics, numbers, the article "ال" (the), and the inseparable conjunction prefix"و" (and). Additionally, they have built a large prefixes' and suffices' list which is used to check all the input words if they include any of them, and the longest of these is stripped off, if found. Finally, the produced word is compared against a list of patterns and if a match is found, the root is produced. Also, the authors in (Taghva et al., 2005) developed an aggressive stemmer similar to the one described in (Khoja and Garside, 1999) aiming at deriving the root of the word, but they have tried to overcome three issues in that stemmer which they believe are weaknesses in it. The three issues they have identified were: (1) the produced roots are sometimes not related to the original words, (2) the root dictionary which they uses can be difficult to maintain , and (3) the inability of the stemmer to remove affixes that should have been removed. In general, it is noticeable that the problem with aggressive stemmers is that as they reduce the words to their roots, most of the time it results in losing the specific meaning of the original words. This fact has caused this type of stemmers to be poor candidates for systems involving high accuracy in matching between similar words. On the other hand, the authors in (Beltagy and Rafea, 2011) extended one of the existing light stemmers, light10 stemmer, as it is considered to be one of the most accurate available stemmers. Also, they have proposed a set of rules in order to be able to handle broken plurals and transform them to their singular forms. The approach they have used allowed the stemmer to satisfy accuracy requirements by employing text within a corpus concept to verify whether to carry out such transformation or not. The transformed word is checked to see whether the word resulting from the suggested transformation is present in the corpus or not. So, if a word resulting from applying a transformation rule on an input word (a potential stem), or from removing certain prefixes or suffixes, is found to have appeared in the corpus, then this word is considered as a stem for the input word.

Similarly, the authors in (Nwesri, 2005) compared and proposed a set of techniques for stripping prepositions and conjunctions present at the beginning of a word, after which the result is checked against a lexicon to decide whether that certain prefix should be stripped from the input word or not. And finally, the authors in (Goweder et al., 2004) dealt with the problem of identifying broken plurals and stemming them to their singular forms. In all the experiments they have performed, the input words were first lightly stemmed using the stemmer proposed in (Khoja and Garside, 1999). As a result of observing that this method resulted in very low precision and aiming at improving the results, they have employing one of the machine learning approaches to add restriction rules automatically. But it is noticeable that the best results of all were obtained using a dictionary-based approach.

And finally the stop words removal stage. There is not one definite list of stop words for Arabic. Depending on the type of the application they are implementing, authors use different stop words list. Some authors build lists that consist mainly of the most common and short function words like "في" (in), "من" (of), "علي" (on), etc…[1]. On the other hand, some authors build list that contains the most common words including lexical words like "مثل" (like), "يريد" (want), "يقول" (say), etc…[2].

3. Conceptual Overview

The main aim of this research is to investigate how preprocessing of tweets written in Egyptian dialect could improve the results of sentiment analysis of these tweets. As stated before the preprocessing stage consists mainly of three stages:

3.1 Normalizing the Annotated Tweets

We have used the normalizer [3] as it is very efficient and there is not much work that can be done in this area. Table 1 defines language normalization rules:

Rule	Example
Tashkeel	حَدَّثَنَا -> حدثنا
Tatweel	اللــــــــــه -> الله
Hamza	ؤ or ئ or ء -> ء
Alef	آ or أ or إ -> ا
lamalef	لا or لآ or لأor لإ -> لا
yeh	ي or ى -> ي
heh	ه or ة -> ة

Table 1. Normalization Rules

3.2 Stemming the Normalized Tweets

Due to the complexity of the Arabic language, several studies with various complexity levels were carried out to address stemming because of its significance in informational retrieval and text mining systems. However, most of these studies were mainly for modern standard Arabic (MSA) and so they can't handle the different dialect specific rules like the Egyptian dialect. For example the word "علشان" (because) if we tried the MSA stemmer the word would become "عش" (hut) since in MSA when a word ends in "ان" it reflects duality, however this word should not have been stemmed originally. The fact which forced us to implement our own customized stemmer. The main objective of the stemmer is to reduce the input word to its shortest possible form without compromising its meaning. That is why we have adopted the light stemming methodology using dialect specific set of prefixes and suffixes because in aggressive stemmers reducing the word to its root can sometimes result in the mapping of too many related terms, each with a unique meaning, to a single root. Moreover, light stemmers are considered very simple to implement and have proven to be highly effective in several information retrieval systems. On the other hand, light stemmers are not applicable of handling some affixes and broken plurals which are very common in the Arabic language (Larkey, 2007). That is why in our

[1] http://www.ranks.nl/resources/stopwords.html
[2] http://arabicstopwords.sourceforge.net
[3] http://arabtechies.sourceforge.net/projec/ normalization _ruby

implemented light stemmer, we have combined some of the rules introduced in (Beltagy and Rafea, 2011), together with a set of rules we have introduced to handle broken plurals for Egyptian dialect which sometimes results in the addition of infixes to a word, as well as handling the removal of certain affixes. In our stemmer's implementation, we have built two lists: one for irregular terms (words that originally start or end by any of the prefixes or suffixes and should not be stemmed) and another one for irregular plurals and their singular forms. These lists are normalized and stemmed. Thus, the input word is first checked against these lists of irregulars if it is present then it won't be stemmed, otherwise the stemming rules will be applied.

The implemented stemmer consists mainly of three stages: 1) prefix removal, 2) suffix removal, and 3) infix removal which is mainly applying the rules for broken plurals. Generally, the prefix removal is the first stage attempted, followed by the suffix removal stage, and finally the infix removal stage. After each stage, the transformed word is checked against the dictionary to determine whether to continue with stemming it, or just stop. Figures 2 and 3 show the sets of prefixes and suffixes proposed for the Egyptian dialect, while figure 4 shows the set rules for handling broken plurals. Most of these rules were inspired from the ones introduced in (Beltagy and Rafea, 2011) with the new ones we propose are highlighted in red.

Prefix	Meaning
ال	The
وال	And the
بال	With the
كال	Like the
فال	Then the
لل	For
وبال	And with the
ولل	And for
وكال	And like the
وفال	And then the
بي	he is
بت	she is
هي	he will
هت	she will
بن	we are
بتت بنت بيت	it is

Prefix	Meaning
و	and
ك	like
ف	then
ل	For or because
ب	With or at

Figure 1: Set of compound and single prefixes with their meanings

Suffix set 1	يات , ات
Suffix set 2	ها , ون , وا , ين , ان , نا , هم , ت, ك , ي , ه , ة , و

Figure 2: Sets of suffixes

Rule ID	Condition	Rule	Example
R1	Length = 5 & 2nd char != و & 4th character = ئ & 3rd character = ا , & 5th char != ي or ه	Replace 3rd char with a (ي), delete 4th char, and add a (ة) at the end. If no match is found, attempt to find a match without the (ة)	حشائش
R2	Length = 5 & 2nd char = و & 4th character = ٶ & 3rd character = ا , & 5th char != ي or ه	Remove 2nd char and add a (ة) at the end. If no match is found, attempt to find a match without the (ة)	روائح
R3	Word ends with "ايا" and 2nd character != و	Remove last three chars, and append string (ية)	هدايا
R4	Length = 3 and 2nd character = 3rd character	Remove 3rd character	زمم
R5	Length = 4 and 3rd character = و and last character is not equal to ه	Remove 3rd character (و)	جذور
R6	Length = 5 and 3rd character is an ا	Remove 3rd character (ا)	مراكب
R7	Length = 5 and & 4th character is an ا & 5th character is ٶ	Remove 4th character (ا) & 5th character is ٶ and add an (ي) after the 2nd character	وزراء
R8	Length = 5 and 1st character is an ا and last character is a ة	Remove 1st character (ا) and last character (ة) and add an (ا) after the 2nd character	اتربة
R9	Length = 5 and 1st character is an ا and 4th character is an ا and second char != ي	Remove 1st and 4th characters(ا)	اشجار
R10	Length = 4 and 3rd character = و and 2nd character = 4th character	Remove 3rd and 4th characters	سدود
R11	Length = 5 && 2nd character = و and 3rd character is an ا	Remove 2nd character (و)	جوانب

Figure 3: Rules for broken plurals

3.3 Find a List of Egyptian Dialect Stop Words

Given the absence of any stop words list for the Egyptian dialect, we had to build this list from the beginning. The process started by identifying the words in the whole corpus (20,000 tweets) between different frequency ranges as shown in figure 5. The figure shows the number of the words in each frequency range, and it is clear from the graph that there is an inverse relationship between the frequency range and the number of words which complies with Zipf's law (Li, 1992). After that, we started by the first set of words consisting of 11 words which had the highest frequency range to be our list of stop words after removing all the sentiment words "جميل" (beautiful), "بشع" (ugly), etc.. , named entities like "فلول" (Followers), "مصر" (Egypt), "مبارك" (Mubarak), etc…, and verbs like "يحاكم" (Trial), "قتل" (kill), etc…, and tested its effect on the accuracy of the classifier. At the beginning there were drops in performance, means that there might be some important words that should not have been removed, or there some other stop words that still needs to be removed. So we worked on identifying these words manually. Then, this process continued accumulatively by adding lists from the following frequency ranges until we have reached a list of stop words consisting of 128 words that increases the accuracy by almost 1.5%. Figure 6 shows the frequency of each stop.

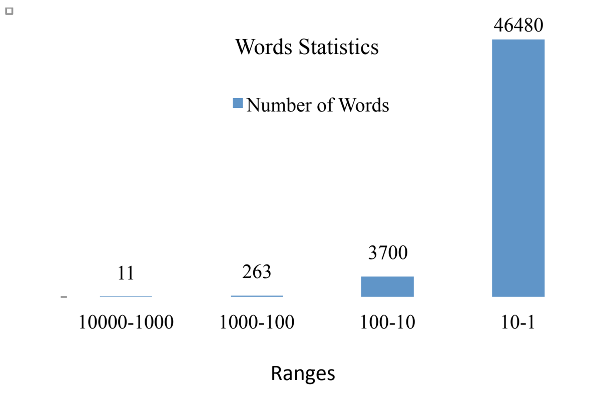

Figure 4. The Number of words in different Frequency Ranges

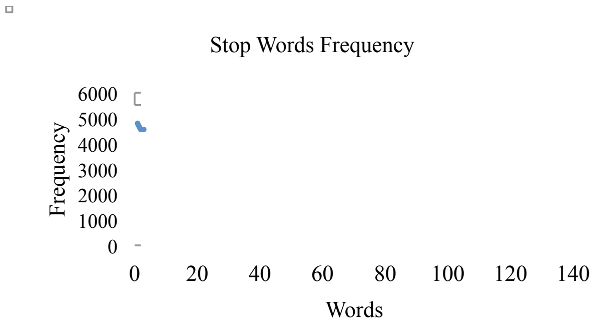

Figure 5. The frequency of each stop word

4. Sentiment Analysis Approaches

The effect of preprocessing on sentiment analysis performance was measured on the two approaches namely ML and SO approaches

4.1 Machine Learning Approach (ML)

This approach uses different feature sets (unigrams, bigrams, and trigrams), together with the Support Vector Machines (SVM) as the machine learning classifier. The preprocessing, features' extraction and the classification are done in three different components, creating the ability to try various arrangements of preprocessing, features and classifiers till reaching the one which yields the highest accuracy.

The methodology used for building the ML classifier consists mainly 5 stages: 1) crawling tweets from twitter to form a corpus, 2) cleaning this created corpus and annotating 1,000 tweets (500 positive tweets and 500 negative tweets), 3) normalizing, stemming and removing the stop words 4) identifying unigrams, bigrams, and trigrams to be used as candidates features in building the feature vectors, 5) using the most known classifier in sentiment classification; SVM. We have used the Weka Suite software (Hall et al., 2009) for the classification process.

4.1.1 Getting Data from Twitter (Arabic Tweets)

Despite the importance of the Arabic language, it is believed to be one of the languages with poor content over the web as very limited number of pages specializes in Arabic reviews. The fact which encouraged us to start using Twitter as the main source for getting vast amounts of data, especially that it provides a search API enabling the search for tweets in the language of interest (Twitter search API, http://search.twitter.com/search.atom?lang=ar& rpp=100&page={0}&q={1}). We were able to crawl more than 20,000 tweets from different news topics. The majority of these crawled tweets were in the Egyptian dialect with small number of tweets in standard Arabic. The size of the corpus was considered one of the main issues as the bigger the size of the training data, the more accurate the classifier will be in classifying any new supplied sentence.

4.1.2 Tweets Cleaning and Annotation

From the 20,000 crawled tweets, 1,000 tweets were annotated (500 positive tweets and 500 negative tweets). For the annotation process, two raters were working on labeling the tweets, and it was observed that they had a high degree of agreement in their classification (over 80%). For those tweets that they labeled differently, a third rater was used to determine its final sentiment. For those annotated tweets, all the user-names, pictures, non-Arabic hash tags, URLs and all non-Arabic words were removed. The tweets selected were chosen based on two assumptions: 1) the sentence represents the opinion of just one author, 2) the sentence holds the author's opinion about only one topic and not sarcastic.

4.1.3 Tweets Pre-Processing

In this stage we just apply the proposed preprocessing tool on the cleaned tweets. Each process is done accumulatively to produce at the end normalized, stemmed tweets with the stop words removed.

4.1.4 Feature Extraction and Feature Vector

Given that our work is mostly in word/phrase level sentiment analysis, we have chosen to work with unigrams, bigrams and trigrams (Khreisata, 2009). Unigrams are considered the simplest features to extract and they provide good courage for the data, while bigrams and trigrams provide the ability to capture negation or sentiment expression patterns. Therefore, the process starts by extracting all the unigrams, bigrams, and trigrams in the 1000 annotated tweets. Then for each of these candidate features, its frequency in the 20,000 tweets was calculated, creating a dictionary for all the candidate features with their corresponding frequencies. Finally for each Tweet, if any of these candidate features is present in it, then this candidate feature frequency is fetched from the dictionary and it is placed in the feature vector representing this tweet. Therefore, the feature vector built for each tweet used term frequency: ({word1:frequency1, word2:frequency2 …}, “polarity”)

4.1.5 Weka Suite Software

For the classification, the Weka Suite Software version 3.6.6 is used as it is a collection of ML algorithms such as NB, SVM, etc… as well as feature selection methods such as IG. Also, various test options exists like configurable number of fold cross validation, test set and percentage split. When the dataset size is large, it is possible to run it directly by inserting the dataset into the program or from the command line.

4.2 Semantic Orientation Approach

The methodology used to build the SO classifier consists mainly of 3 steps: 1) using 600 sentiment annotated tweets (300 positive and 300 negative) to build the sentiment words list, 2) normalizing, stemming and removing the stop words and 3) classifying the remaining 400 tweets (200 positive and 200 negative) as positive or negative using the sentiment word found in the tweet, and building a confusion matrix for the tweets classified as positive and another matrix for the tweets classified as negative to measure the accuracy of classification.

4.2.1 Building the list of Sentiment Words

Given the limited work done for Arabic text in the field of sentiment analysis, especially for the Egyptian dialect, we had first to start by manually building two lists: one for the most occurring positive sentiment words, and one for the most occurring negative sentiment words. Then for each word in these lists a weight is given to it based on its frequency in 300 positive tweets and its frequency in 300 negative tweets.

4.2.2 Tweets Pre-Processing

The same steps (normalizing, stemming, stop words removal) are done in the same order as in the ML approach. Both the tweets and the sentiment words list are processed.

4.2.3 Classifying the Test Set of Tweets

To determine the class of each tweet, a cumulative *score* is calculated using the sentiment words in the tweet to determine its class. For each sentiment words present, its score is added to the total in the following way:

$$score = \sum_{i=1}^{n} (w_{pi} - w_{ni})$$

where w_{pi} is the positive weight of the word, w_{ni} is the negative weight of the word, and they are calculated based on the number of times this word appeared in the positive tweets, and the number of times this word appeared in the negative tweets. The weights assigned to the sentiment words are used to determine how close it is to positive “1” or to negative “-1”. The final value of the score (score > 0 or score < 0) determines polarity of the whole tweet. Since, in this stage we are only dealing with two classes building a binary classifier, positive and negative, the neutral class, where either no sentiment words were found or both numbers of positive and negative sentiment words are equal, is not acceptable. Thus for each class a classifier is built determining whether the tweet belongs to its corresponding class, or it belongs to the class named “other”. Then, the accuracy, the precision, the recall, and the F-measure of each classifier will be calculated, which will be averaged at the end to reach a final unified accuracy.

5 Experimentation and Evaluation

5.1 ML Results and Discussion

5.1.1 Results

To test the performance of our proposed preprocessing stages, we have applied our 3 stages accumulatively meaning that the normalized tweets will be then stemmed, and finally the stop words will be removed from these stemmed tweets. Four experiments were carried out: 1) using raw tweets, 2) after applying the normalizer, 3) after applying the stemmer, and 4) after removing the stop words; and their results are shown in tables 2, 3, 4 and 5. The SVM classifier was first trained using the frequency of the unigrams only; secondly it was trained using a combination of both unigrams and bigrams with an attempt to capture any negation or sentiment switching phrases; and finally it was trained using a combination of unigrams, bigrams and trigrams to capture any sentiment expression or idioms. The results were as follows using 10-fold validation:

	SVM			
	Accuracy	*Precision*	*Recall*	*F-Measure*
Unigrams	**0.740**	**0.740**	**0.740**	**0.740**
Unigrams + Bigrams	0.739	0.740	0.739	0.739
Unigrams + Bigrams + Trigrams	0.737	0.738	0.737	0.737

Table 2. SVM results using raw tweets

	SVM			
	Accuracy	*Precision*	*Recall*	*F-Measure*
Unigrams	**0.756**	**0.756**	**0.756**	**0.756**
Unigrams + Bigrams	0.754	0.755	0.754	0.754
Unigrams + Bigrams + Trigrams	0.753	0.754	0.753	0.753

Table 3. SVM results using normalized tweets

	SVM			
	Accuracy	*Precision*	*Recall*	*F-Measure*
Unigrams	0.774	0.774	0.774	0.774
Unigrams + Bigrams	0.784	0.784	0.784	0.784
Unigrams + Bigrams + Trigrams	**0.787**	**0.787**	**0.787**	**0.787**

Table 4. SVM results using stemmed tweets (1)

	SVM			
	Accuracy	*Precision*	*Recall*	*F-Measure*
Unigrams	0.738	0.739	0.738	0.738
Unigrams + Bigrams	0.775	0.775	0.775	0.775
Unigrams + Bigrams + Trigrams	**0.779**	**0.779**	**0.779**	**0.779**

Table 5. SVM results using stemmed tweets (2)

	SVM			
	Accuracy	*Precision*	*Recall*	*F-Measure*
Unigrams	0.777	0.777	0.777	0.777
Unigrams + Bigrams	0.788	0.788	0.788	0.788
Unigrams + Bigrams + Trigrams	**0.788**	**0.788**	**0.788**	**0.788**

Table 6. SVM result after stop words removal

Tables 2 shows the results obtained in the classification process for SVM classifier using term frequency scheme respectively before applying any preprocessing, then Table 3-6 show the results obtained after applying each process accumulatively. Tables 4 and 5 show the result of applying two stemmers: 1) our implemented stemmer, and 2) light stemmer [1]. It is important to note that the performance measures of both the

[1] http://pypi.python.org/pypi/Tashaphyne/

positive and the negative classifiers were first calculated using the average of the 10-fold validations, then these measures were averaged to produce the numbers presented in the tables.

5.1.2 Discussion

Comparing the results of SVM, it was clear better results were produced after applying the preprocessing stages. The improvement between the best accuracy results before and after applying preprocessing is almost 4.5%. The same goes with the precision, recall and the F-measure. This behavior was observed in more than one study as preprocessing usually tries to reduce the noise in the text, thus eliminating part of the distortions in the features space. Also an important observation was noticed is that the number of features was reduced dramatically from 6622 features in case of best result using unigrams before applying preprocessing to 4893 features in case of best result using trigrams after applying preprocessing. That is because the more steps we apply from the preprocessing stage, the more related features converge together reducing the problem of features over-fitting and increasing the rate of the learning scheme.

We have tested our implemented stemmer against one of the light stemmers available. Analyzing the results in tables 4 and 5, it is noticeable that our implemented stemmer produces better results because dialect specific issues that we have addressed in our implementation. For example, the word "علشان" and "عشان" both forms of the words are right and they mean "because", in our stemmer we have included them in the irregular list and so they won't be stemmed, however in the light stemmer they will be stemmed to "علش-not a word" and "عش-hut" which are completely two different words.

Regarding the n-gram model, we can note clearly that after applying the stemming, adding the bigram model to the unigram model greatly improves the performance. However, there were not big differences in the performance by adding the trigram model to the combined unigram and bigram model. It should be noted that we have used only the 1000 annotated tweets to build the unigram, bigram and trigrams models, may be using more tweets could result in more unigrams, bigrams and trigrams, thus further improvements in the results.

5.2 SO Results and Discussion

5.2.1 Results

To test the effect of the preprocessing on the SO performance, 3 experiments were carried out one at each stage with the preprocessing applied to both the sentiment words and the tweets. Before carrying the experiments, we have removed stop words as their removal should not have any impact on enhancing the results but their removal will accelerate the classification process. In the first experiment we normalized both the tweets and the sentiment words, and then in the second experiment both were also stemmed. We didn't test the effect of stop words removal on SO performance as there is no intersection between the sentiment words and the stop words, thus removing the stop words won't affect the performance of the SO, it is only the sentiment words which affect it.

	Positive	Negative	Average
Accuracy	0.725	0.653	0.689
Precision	0.768	0.714	0.741
Recall	0.725	0.653	0.689
F-measure	0.746	0.682	0.714

Table 7. SO results using raw tweets

	Positive	Negative	Average
Accuracy	0.728	0.658	0.693
Precision	0.767	0.711	0.739
Recall	0.728	0.658	0.693
F-measure	0.747	0.683	0.715

Table 8. SO results using normalized tweets

	Positive	Negative	Average
Accuracy	0.760	0.758	0.759
Precision	0.761	0.770	0.765
Recall	0.760	0.758	0.759
F-measure	0.760	0.764	0.762

Table 9. SO results using stemmed tweets (1)

	Positive	Negative	Average
Accuracy	0.753	0.755	0.754
Precision	0.758	0.763	0.760
Recall	0.753	0.755	0.754
F-measure	0.755	0.759	0. 757

Table 10. SO results using stemmed tweets (2)

Tables 7-10 calculate the performance results for the classification of the binary classifiers at each stage of the preprocessing. Tables 9 and 10 test the result of applying two stemmers: 1) our implemented stemmer, and 2) light stemmer.

5.2.2 Discussion

Regarding the effect of the preprocessing on the SO performance, we can note that there was an improvement of 7% in the accuracy and the recall, while there was an improvement of 2% in precision and 5% in the F-measure. That is because in SO it is only the form of the sentiment words which affect the performance, thus after preprocessing, the sentiment words in the tweets were almost converted to the same form of the sentiment words in the lists and they were easily extracted. However not all tweets contain sentiment words and even if there exist they represent a very small percentage of the words in the tweet. Hence, building more comprehensive lists of sentiment words could be considered a possible solution to further enhance the performance.

Analyzing the results in tables 9 and 10, it is noticeable that both stemmers produce almost the same results with very minor changes. This behavior is somehow expected as the stemming of most of the sentiment words is expected to be the same because there are less dialect specific sentiment words.

Comparing the results of the positive and the negative binary classifiers, it was clear that the performance of the positive classifier was improving over the performance of the negative classifier until we have applied the stemmer they started to become very close. This behavior reflects the fact that the positive tweets are less noisy than the negative tweets; therefore with minimal preprocessing (just normalizing) it has almost reached the best result.

6. Conclusion and Future Work

In this paper, we have demonstrated the effect of the preprocessing on enhancing the sentiment classification of 1000 Arabic tweets (positive or negative) written in Egyptian dialect from Twitter. As a first step, we believe that the results obtained are very promising. We have used two stemmers (our implemented stemmer and light stemmer) for the aim of comparing their performance in both approaches, and it was noticeable that in ML approach our stemmer produced an improvement of 1% over the light stemmer, while in the SO approach our stemmer produced an improvement of 0.5% over the light stemmer due adding Egyptian dialect prefixes, suffixes and rules for broken plurals. In the ML approach, we have applied the feature vectors to the SVM classifier once before applying the preprocessing and once after applying each stage of the preprocessing to test its effect on the system's performance, and at the end we have reached an improvement in the performance of almost 4.5% in all measures. While in the SO approach, we have applied each stage of the preprocessing to both the tweets and our created sentiment words lists, and at the end we have reached an improvement between 2-7% for the different performance measures.

It is important to note that from the possible causes behind the improvement of the ML approach (78.8%) over the SO approach (75.9%) given that the SO depends only on the sentiment words: 1) the tweet originally contains no sentiment words, 2) the sentiment word in the tweet is not present in the lists, 3) the sentiment word even after applying the preprocessing is written in a different form from the one stored in the list. For example, the word "خير - good" and "خيرا - good", in meaning they are the same but here the suffix "ا" present after the stemming makes them two different words. However, this is considered a defect in the normalization program we are using as "ا" is considered a diacritic that should have been removed.

For future work, we believe that our developed stemmer could be further improved by closely monitoring the performance of each applied rule, thus increasing the probability that more related words will be reduced to the same stems. Also our developed stop words list needs to be further investigated as the performance increased by only 0.1% which means that there are some other stop words that still need to be removed. Moreover, we will be trying to include the semantic to build a hybrid approach combining both ML and SO approaches and testing the effect of preprocessing on this hybrid approach. Accordingly, a more comprehensive list of all Egyptian dialect positive and negative sentiment words needs to be built since there doesn't exist any of them.

Finally, improving the performance of this preprocessing component with all its stages is currently considered our main aim as it is part of a bigger system for determining sentiment of the Arabic tweets, extracting hot topics, and identifying influential bloggers (Shoukry and Rafea, 2012).

Acknowledgment

The authors would like to thank ITIDA for sponsoring this project entitled "Semantic Analysis and Opinion Mining for Arabic Web", and the Egyptian industrial company LINK-Development and its team for their help in developing a tool for collecting and annotating the tweets.

References

Rehab Duwairi, Mohamed N. Al-Refai, and Natheer Khasawneh. 2009. Feature reduction techniques for Arabic text categorization. Journal of the American Society for Information Science and Technology, 60(11), 2347–2352.

Khaled F. Shaalan and Hafsa Raza. 2009. NERA: Named entity recognition for Arabic. Journal of the American Society for Information Science and Technology, 60(8), 1652–1663.

Sara Morsy and Ahmed Rafea. 2012. Improving Document-Level Sentiment Classification Using Contextual Valence Shifters Natural Language Processing and Information Systems Lecture Notes in Computer Science Volume 7337, 2012, pp 253-258

Leah S. Larkey, Lisa Ballesteros, and Margaret E. Connell. 2007. Light stemming for Arabic information retrieval. In Arabic Computational Morphology, A. Soudi, A. van der Bosch, and G. Neumann, Eds. 221–243.

S. Khoja, and R. Garside. 1999. Stemming Arabic text. Tech. rep. Computing Department, Lancaster University, Lancaster, U.K.

Kazem Taghva, Rania Elkhoury, and Jeffery Coombs. 2005. Arabic stemming without a root dictionary. ITCC 1, 152–157.

Samhaa R. El-Beltagy and Ahmed Rafea. 2011. An accuracy-enhanced light stemmer for arabic text. ACM Trans. Speech Lang. Process. 7, 2, Article 2 (Feb. 2011), 22 pages.

Abdusalam Nwesri, S. M. M. Tahaghoghi, and Falk Scholer. 2005. Stemming Arabic conjunctions and prepositions. In Proceedings of the 12th International Symposium on String Processing and Information Retrieval (SPIRE'05). Lecture Notes in Computer Science, vol. 3772, Springer, 206–217.

Abduelbaset Goweder, Massimo Poesio, and Anne De Roeck. 2004. Broken plural detection for arabic information retrieval. In Proceedings of the Annual ACM Conference on Reseurch and Development in Information Retvieval (SIGIR'04).

Abduelbaset Goweder, Massimo Poesio, Anne De Roeck, and Jeff Reynolds. 2004. Identifying broken plurals in unvowelised Arabic text. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP).

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The WEKA Data Mining Software: An Update; SIGKDD Explorations, Volume 11, Issue 1.

Mohamed Elhawary, and Mohamed Elfeky. 2010. Mining Arabic Business Reviews, Google Inc., Mountain View, CA, USA, 2010 IEEE International Conference on Data Mining Workshops.

Bo Pang and Lillian Lee. Thumbs up? Sentiment Classification using Machine Learning Techniques, Department of Computer Science, Cornell University, Shivakumar Vaithyanathan, IBM Alma den Research Center.

Laila Khreisata. 2009. A machine learning approach for Arabic text classification using N-gram frequency statistics, Journal of Informatics, Vol 3, Issue 1, January 2009, Pages 72-77.

Eiman T. Al-Shammari. 2009. A Novel Algorithm for Normalizing Noisy Arabic Text, Computer Science and Information Engineering, WRI World Congress on , vol.4, no., pp.477-482, March 31 2009-April 2 2009 doi: 10.1109/CSIE.2009.952

Amira Shoukry, and Ahmed Rafea. 2012. Sentence-level Arabic sentiment analysis, Collaboration Technologies and Systems (CTS), 2012 International Conference on , vol., no., pp.546-550, 21-25 May 2012 doi: 10.1109/CTS.2012.6261103

Wentian Li. 1992. Random texts exhibit Zipf's-law-like word frequency distribution, Information Theory, IEEE Transactions on , vol.38, no.6, pp.1842-1845, Nov 1992 doi: 10.1109/18.165464