A FRAMEWORK FOR THE RAPID DEVELOPMENT OF LIST BASED DOMAIN SPECIFIC ARABIC STEMMERS

Samhaa R. El-Beltagy

Faculty of Computers and Information,
Computer Science Department
Cairo University
5 Dr. Ahmed Zewail Street, 12613 Orman
Giza, Egypt
samhaa@computer.org

Ahmed Rafea

Computer Science Department
American University in Cairo
113 Kasr El Aini St.,
P.O. Box 2511,
Cairo, 11511, Egypt
rafea@aucegypt.edu

Abstract

Increased interest in the field of text mining, has also witnessed increased interest in the development of more accurate stemmers required by various text mining applications. The goal of this work is to present an approach for stemming that falls somewhere in between aggressive and light stemming. Not only does the presented work address the removal of suffixes and prefixes, it also introduces a set of rules for removing infixes without the use of a morphological analyzer. The basic premise on which this work is based, is that in any reasonably sized corpus, a word and its stem are likely to both appear in the corpus. By capitalizing on this observation, the work aims to present a method for rapidly building stem lists from a small set of documents as well as make use of the local context of a document when carrying out stemming. The evaluation of the work shows that it significantly improves stemming accuracy. It also shows that by improving stemming accuracy, a task such as automatic annotation can also be significantly improved.

Introduction

Stemming is a very common operation in any information retrieval system as well as almost all text mining applications. The importance of building a good stemmer lies in the fact that stemming can directly affect the performance of any application of which it is a component. Research has shown that stemming of Arabic terms is a particularly difficult task because of its highly inflected and derivational nature (Larkey et al., 2002). Most work in this area either tries to reduce a given word to its root (aggressive stemmers), or to identify a set of prefixes and suffixes the removal of which can have a positive impact on a given task such as information retrieval (light or simple stemmers). The main advantage of using a light stemmer is that it is very simple to apply and though not very accurate in its stemming performance, has proved effective for the task of information retrieval. There are instances however when light stemmers do not offer the needed accuracy. Tasks such as semantic annotation, information extraction, and segmentation using lexical chaining, all exemplify such cases. In this report, accuracy in stemming is emphasized through the proposal of an approach that extends the notion of light stemming. Not only does the presented work investigate the removal of suffixes and prefixes, it also introduces a set of rules for removing infixes without the use of a morphological analyzer. A stem in the context of this work, is the singular, and whenever applicable, masculine form of a word which does not necessarily map to its root (a lema). The basic premise on which this work is based, is that in any reasonably sized corpus, a word and its stem are likely to both appear in the corpus. This is a similar assumption to that adopted by (Xu and Croft, 1998) in their work on corpus based stemming. But unlike their work, focus is placed on building stem lists through the application of heuristic transformation rules on words in order to reduce each to its singular form. If a word resulting from applying a transformation rule on an input word (a potential stem), is found to have appeared in the corpus, then this word is considered as a stem for the input word. It is important to note that this procedure is not error free. For example, the same rule that would reduce "دروس" to "درس " can reduce "الغروب" to "غرب" which is not its stem, but if both words are in the corpus "غرب" will be assumed to be the stem for "غروب". By maintaining a stem list containing words that should not be further conflated (such as "غروب"), mistakes like this can be avoided. Building such lists from scratch is very expensive in terms of human effort and is not feasible in general non domain specific applications. The presented work however, introduces a methodology for rapidly building lists like this through a semi-automatic interface and shows how building such lists can pay off within a domain specific application. and shows how building such lists can pay off within a domain specific application. The rest of this paper is organized as follows: section 2 presents an overview of the presented work, section 3 details transformation rules that can be applied to a word in order to obtain its stem, section 4 describes the procedure for rapidly building a stem list, section 5 describes experiments carried out to evaluate the presented work and their results, and section 6 concludes this work and provides future research directions.

Building and using the stemmer: an overview

In this work, building a stemmer is a two phase process. In the first phase (training phase), a number of documents from an input corpus can be selected for the stem list building process. Through a user interface, a human can then verify the resulting words and their stems making any corrections (if needed) to ensure that words that should not be conflated are stored in the stem list. In this step also, the user can specify stems for irregular words

which are then stored in a dictionary. In the second phase (operational phase), given any word or document, stemming can be carried out by checking whether its potential stem (obtained by applying appropriate rules) exists in the stem list or not. Depending on the required strictness of the stemmer, the original word whose conflated form cannot be matched to an entry in the stem list or the input text can be returned, or the corresponding lightly stemmed version of the word may be returned. The presented work aims to maintain the semantic of the word when stemming it as much as possible, but without a semantic analyzer this is not always possible. Whenever there is ambiguity regarding whether or not to stem a word because a proper stem will not be obtained without understanding the meaning of the word, the shorter form of the term is favoured. For example, the word "مدرسة" may mean school in which case according to our definition of a stem, it should not be further reduced. The same term can also mean teacher the stem of which is "مدرس" and since this is the shorter form, it is the one that will be kept. This feature of course can be controlled through the dictionary itself, so if the user of our stemmer adds the term مدرسة" to the stem list, then every time this term is encountered, it will not be further reduced. There are other cases when the removal of the suffix "ة" should be avoided. For example, the equivalent of the term "minute" in Arabic is "دقيقة",. Most light stemmers will remove the "ة" at the end which will convert the word to "دقيق" which means "flour" (a totally different word). By maintaining a list of word stems that contain words such as "دقيقة" we would know that this word does not need to be further conflated. Like most light stemmers, prefixes and suffixes commonly attached to verbs, are not included in transformation rules presented by this work.

Stemming Steps

The presented stemmer works on three levels: prefix removal, suffix removal and infix removal. Each of these levels utilizes a different set of rules. In general, prefix removal is first attempted, then suffix followed by infix removal. Each of these steps is described in the following sub-sections.

Prefix Removal

Prefix removal is the first step to be carried out in either training or operational mode. In our work, we've divided prefixes into two classes, compound prefixes which consist of more than one letter and singular prefixes or particles that are made up of just a single character.
Our compound prefix set covers the following patterns:

لا ,ال,وال,بال,كال, فال, لل,**وبال**, **ولل**, **وكال**, **وفال**

Underlined prefixes, denote the set of prefixes not removed by the light10 stemmer which is the baseline stemmer in this work. For the removal of any prefix, the length of the term to be stemmed minus the length of the prefix has to be greater than or equal to two, otherwise no prefix removal is carried out. A small experiment was carried out to test the effect of removing prefixes in the compound prefix set from any word that starts with them if they satisfy the length requirements mentioned above (except for prefix لا). In this experiment 9 documents making up a total of 19291 words, were used. From these, a total 3851 unique words (excluding stop words, single characters and numbers) were extracted. The number of words on which prefix removal was applied was 1708 (44% of input words). This resulted in the generation of 1345 unique terms (a reduction of 21 %). Words that started with a prefix, along with their altered forms were examined to assess the accuracy of compound prefix removal step. After identifying all cases from which prefixes should not have been removed, but which met the only requirement which is the length condition, it was found that these cases are quite rare and that the overall accuracy of this step is about 98.7%. For that reason, is was decided not to impose any further rules on the removal of a prefix in the compound set (i.e if a word matches with a prefix in the compound prefix set, we simply remove it).

Prefix لا is considered as a special case and is handled a bit differently than all the other prefixes in the compound set. The reason for this, is that a word might have this prefix to denote negation or simply because the causation prefix "ل" happened to be attached to a word starting with an "ا". So validation similar to that carried out prefixes in the singular prefix set (is detailed below) is first carried out before removing this prefix.

The singular prefix set includes the following conjunctions and prepositions:
و, ك, ف, ل, ب
Underlined particles, denote the set of prefixes not removed by the light10 stemmer. Removing these conjunctions and prepositions from a word without validating the removal can often result in mistakes. This is because this removal means removing the first letter of any word that starts with any of this prefixes. So, for example "وليد" (baby) would be reduced to "ليد" which has no meaning and "فيل" would also be reduced to the meaningless word "يل". This is perhaps why the only prefix from this set that a light stemmer removes is the "و" which occurs often as a preposition attached to a word. But as stated before, the likely hood of error in removal is quite high. So, to validate the removal of these in training mode, the prefix is first removed and then a check is made to see if the resulting word has appeared at least once as part of the input corpus. If the term is found, then it is assumed that the prefix can be removed safely. In stemming mode, the dictionary is first checked for the resulting word then if not found, terms in the input document are checked as well.

Suffix Removal

The next step after prefix removal, is suffix removal. Two suffix sets have been identified as follows:

Suffix set 1	**يات** ,ات, **اته**
Suffix set 2	ها, ون, وا, ين, ان, ية, يه, **هم**, ي, ه, ة

Table 1: Set of suffixed handled by the stemmer

Each of these sets is treated differently. Again, for the removal of any suffix in either of the suffix sets, the length of the resulting term must be at least 2 characters in length. If this condition is not met, the input term is returned as is. If an input term ends with any of the suffixes in suffix set 1, the suffix is removed and validation is carried out in a similar manner as described

before (in training mode this means checking that the word resulting from the removal of the suffix has appeared at least once in the input corpus. In stemming operational mode, this means checking that the resulting word either appeared in the dictionary or in the input document). If a match is found, the resulting word is returned. If no match is found, the "ة" character is added to the resulting word and validation is carried out again. In case a match is found, the resulting word is returned. If no match is found suffixes in suffix set 2 are checked.

Handling of suffixes in the second suffix set is carried out in a similar manner. Here, the original word is checked for each of the suffixes in the suffix set in the denoted order. If the word ends with the suffix, the suffix is removed and the resulting word is validated. If found, it is returned. Otherwise, a check is made to see whether the resulting word ends with a "ت". If it does, then the "ت" is replaced with a "ة" and the resulting word is validated again. If validated, it is returned but otherwise the original word is checked for the next suffix until all are exhausted. If no match is found in this procedure, the original word is returned as is.

Infix removal

Infixes usually occur as a part of broken or irregular plurals. While there is no straight forward way for handling irregular plurals except through the use of dictionaries, some of the commonly used broken plurals exhibit well defined patterns that can be detected and transformed. However, just because a word conforms to any of the detected patterns, does not mean that a transformation from the pattern to its possible stem is valid which is why stemming of broken plurals is not usually an easy task. [Gowder et al, 2004], show that simply matching a word to its broken plural pattern, results in very low precision. In their work, they experimented with various ways for reducing broken plurals to their singular forms. In all experiments, input words are firstly lightly stemmed using a modified version of the aggressive Khoja stemmer [Khoja & Garside, 1999]. In the first and least accurate experiment, all forms that fit the pattern of a broken plural were detected and analyzed to see whether or not words that fit these patterns are in fact broken plurals. Having found that this technique results in very low precision, an alternative method which adds further restrictions based on the author's observations to existing patterns, was adopted. Using this method precision increased significantly. In the third variation, a machine learning approach was adopted to automatically add restriction rules which further improved the results, but the best results of all were obtained using a dictionary based approach. Like this work, focus in the work of [Gowder et al, 2004] was on ways of correctly identifying patterns within the text and developing heuristics in order to determine with accuracy that a certain word fits a certain pattern. Their work confirms the effectiveness of using stem dictionaries, while this work illustrates how to rapidly build these. Unlike their work however, even without the use of a dictionary, this work aims to increase the precision of a match, by checking that a transformation from a word that matches a pattern to its potential stem results in a word that has already appeared within the context in which the word has originated. In case the input to the stemmer, is a document, the context (or local context) is all unique words that have appeared in that document. When the input is a set of documents, then the context is the aggregate of all words in all documents. As the size of the local context increases, the better the chance is for finding a match between a word and its potential stem. This has the effect of increasing recall, but also decreasing precision, as it is also likely that an incorrect match will be found. Examples of patterns detected and handled by the developed stemming algorithm are shown in table 2.

PCode	Examples	Stems
P1	سدود, خطوط,حدود	سد, خط,حد
P2	بذور ,جذور ,سطور ,دروس	بذرة , جذر , سطر, درس
P3	اشجار, أمراض, اقوال	شجرة,مرض ,قول
P4	أشهر, أحرف	شهر, حرف
P5	مراكب,مدارس	مركب ,مدرسة
P6	هدايا,وصايا	هدية, وصية
P7	جوانب , عوائل, مواشي	جانب, عائل, ماشية
P8	دول , ورد, شجر	دولة, وردة, شجرة
P9	حشائش,قصائد, دلائل	حشيشة, قصيدة, دليل
P10	روائح, فوائد, قوائم, سوائل	رائحة, فائدة, قائمة, سائل
P11	اجهزه, اتربة	جهاز, تراب
P12	حديثا, نوعا, نجاحا	حديث, نوع, نجاح

Table 2: Examples of patterns handled by the system

For each of the above patterns, a transformation rule is defined to transform the broken plural pattern to its singular form. Each time a pattern matches and a transformation occurs, validation takes place as previously described. If validation results in a match being found, then the transformed pattern is assumed to be the stem of the original word.

Procedure for rapidly building a stem list

In order to rapidly build a stem list which as stated before can have the effect of boosting stemming accuracy, a set of documents from the input corpus need to be selected as a training set. The following steps are then carried out:

1. read all unique terms in a corpus of documents and place in initially empty set allTerms. This step includes removal of dialectics and normalization of letters.
2. for each unique term t_i
   a. generate term t_i' by removing prefixes as detailed above from t_i
   b. generate term t_i'' by removing suffixes as detailed above from t_i'
   c. generate term t_i''' by removing infixed as detailed above from t_i''
   d. store t_i and the result potential stem t_i''' in a stem table.
3. Display all term pairs (t_i , potential_Stem_ t_i) stored in the dictionary to the user for validation. (user interface is shown in fig 1). What is not shown, is that in case the user indicates that a term is

incorrectly stemmed (s/he can then enter the correct stem for that word).

4. For each validated entry, store ONLY the validated stem in a text file. This now constitutes the stemming stem list.

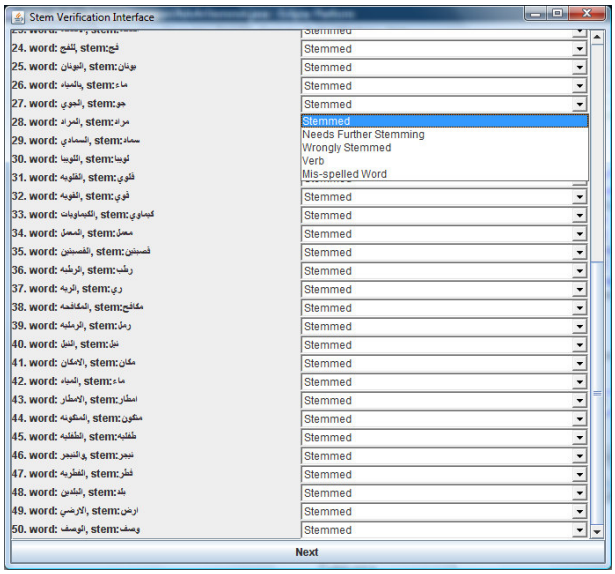

Figure1: Stemming Validation Interface

Evaluation

When evaluating the proposed approach, the main concern was on assessing the stemming accuracy achieved by the approach as well as comparing the improvement to a baseline system in order to determine the significance of the improvement. It was also important to establish that improvement in accuracy can have a contribution to real life applications. To address each of these evaluation concerns, separate experiments were carried out. These are detailed in the following sub-sections.

Determining improvements in accuracy

The experiment presented in this sub-section was carried out in order to measure and compare the stemming accuracy achieved by the proposed approach with that of a baseline system The baseline stemmer with which comparisons were made, is the light10 stemmer [Larkey et al, 2002]. For experimentation purposes, the light stemmer was developed in Java using the rules outlined for the light10 stemmer in [Larkey et al, 2002]. The domain chosen for experimentation was the Agricultural domain. To build an Agricultural stem list, a total of 9 Agricultural extension documents were used as a training set. The total number of words in these documents was 19291, but unique nonstop words amounted to only 3869 terms. From these, a stem list consisting of 1268 terms was built as well as a dictionary for irregular terms consisting of 68 terms. The whole process of reviewing the stems and building the stem list took a little less than 2 hours. After building the stem list, stemming accuracy was tested by applying the built stemmer on a set of 4 previously unseen, agricultural documents. Each document was stemmed separately using the proposed stemmer as well as using a light stemmer. The total number of words in these document was: 9818 while the average length of each document was 2455 ± 405 words. In order to facilitate the manual review of the resulting stems, only unique non stopword terms and their stems were shown to the evaluator. Stems resulting from the proposed approach were shown side by side to stems resulting from the light stemmer. Entries such as verbs, , English words and misspelled terms were excluded from the evaluation process. However, words that can be either verbs or nouns depending on the context, were included and were treated as nouns. A stem was considered correct only if it followed our definition of a stem, which is that it represents the singular, and whenever possible, masculine form of a words. After manually reviewing all resulting stems, accuracy for each method on each document was calculated using the following formula:

$$accuracy(d_i) = all\_correct\_stems(d_i) / total\_stemmable(d_i)$$

where $all\_correct\_stems(d_i)$ is the sum of all correctly stemmed words in document d_i and $total\_stemmable(d_i)$ is total number of valid words (that are not stopword, verbs, etc) in document d_i. The results of this step are shown in table 3.

	Proposed Stemmer	**Light Stemmer**
Document 1	90.4%	50.9%
Document 2	89.4%	45.8%
Document 3	93.1%	55.3%
Document 4	90.5%	47.2%
Average Accuracy	**90.8% ± 1.59%**	**49.8±4.25%**

Table 3: Examples of patterns handled by the system

From table 3 it can be seen that on average when using the proposed stemmer in conjunction with a stem list there is a 82.3% improvement in the accuracy of stemming. Despite the fact that the used dataset is a very small one, using a t-test to calculate the significance of the difference between the results, showed that the difference in accuracy between the proposed stemmer and a light one is extremely statistically significant (with $p < 0.0001$).

Evaluating the stemmer with real life applications

Have establishing that using the proposed stemmer, better stemming accuracy can be achieved, the goal of the second experiment was to investigate whether such an improvement can in fact have an implication for real applications. One of the motivations for developing the proposed stemming approach was to use it as part of an Arabic semantic annotation system as well as for an ontology learning system. For both of these systems it was very important to maximize matches between concepts in an ontology and their corresponding forms, but not with non matching terms as this would directly affect the performance of the semantic annotation system, as well as the automatic evaluation of the ontology learning system. Using existing stemmers either resulted in missing certain important matches as for example for the concepts

"امراض, مرض" and "اسمدة, سماد" as well as "حشائش, حششة" or conflated independent concepts such as "رية and ري". However, the former case was the more common one. So, it was only natural to test the developed stemmer on a simplified version of a system that carries out automatic annotations. A description of the full system can be found in [El-Beltagy et al , 2007]. Towards this end, an experiment was setup to annotate section headings extracted from agricultural documents using concepts from an agricultural ontology. The used ontology consisted of 322 concepts. The section headings were automatically collected from 90 Agricultural extension documents and amounted to 3192 headings. Each heading was tagged with zero or more concepts in an ontology if a match between phrases in that heading and an entry in the ontology was found. Stemming of concepts and headings is an essential pre-processing step in the matching process. Using a light stemmer in this step resulted in the generation of 3260 correct concept annotations, which means that an average of 1.02 tags were generated per heading with a standard deviation of 0.98 tag per heading. Substituting the light stemmer with the proposed one and using the dictionary generated in the first experiment, a total of 3934 correct concept annotations were generated (no incorrect tags were generated in either experiment). So, on average 1.23 tags were generated per heading with a standard deviation of 1.04 tags. Comparing these results, it can be observed that using the proposed approach resulted in an increase of 20.7% correct tags. When the average number of tags were used to establish the significance of the difference between the two results, the difference was found to be extremely statistically significant with $p < 0.0001$. Another experiment was carried out to determine the effect of stemming on the task of keyphrase extraction. In this experiment the KP-Miner system was used for keyphrase extraction [El-Beltagy and Rafea, 2009]. The used dataset consisted of one hundred randomly collected articles from the Arabic Wikipedia [Wikipedia, 2008]. Keywords for each article were obtained from the keyword meta-tag associated with each, but numeric entries (mostly denoting year numbers) were ignored and so were Wikipedia related tags (such as article seed for example) . The average number of words per document in this dataset is 804 ± 934 and the average number of keyphrases is 8.1 ± 3.2. The percentage of author assigned keyphrases actually appearing within the body of associated articles in this dataset is 81.8%. The KP-Miner system allows the user to specify the number of keypharses to extract for each input document. Setting this number to 10, a comparison was held to see how many keyphrases would be correctly extracted when using a light stemmer as opposed to using the proposed stemmer. Four different configurations for the proposed stemmer were used:

C1: Proposed stemmer is used with no stem list

C2:Proposed stemmer is used with no stem list, but in conjunction with a light stemmer (i.e stemming is enforced). What that means is that after carrying out transformations outlined before, a light stemmer is invoked on the resulting word.

C3: Proposed stemmer is used with the stem list obtained from agricultural documents

C4: Proposed stemmer is used with the stem with a stem list in conjunction with a light stemmer.

The results of comparing these configurations with each other and with a light stemmer are shown in table 4.

	Total # of matches	**Average Precision**	**Average Recall**	**Avg # of matches / document**
Light Stemmer	175	0.175 +/- 0.088	0.292 +/- 0.244	1.75 +/- 0.88
C1	180	0.181 +/- 0.082	0.297 +/- 0.240	1.80 +/- 0.82
C2	187	0.188 +/- 0.090	0.305 +/- 0.239	1.87 +/- 0.90
C3	192	0.192 +/- 0.092	0.312 +/- 0.241	1.92 +/- 0.83
C4	194	0.194 +/- 0.093	0.314 +/- 0.241	1.94 +/- 0.93

Table 4: Comparison between a light stemmer and different configurations for proposed stemmer

As can be seen from table the best result was obtained using the proposed stemmer in conjunction with a stem list and a light stemmer. These results show an approximately 11% improvement in keyphrase extraction over the basic light stemmer. However, when using the t-test to compare average precision values, the difference between these results turned out to be statistically not significant with $p = 0.1394$. Further experimentation with a larger dataset, and with other configurations for the proposed stemmer are planned.

Conclusion and Future Work

This paper has presented an approach for making use of text within a corpus or a document for verifying whether or not to strip a word from certain prefixes, suffixes or infixes. The approach can be utilized for rapidly building stem lists that can greatly improve stemming accuracy. For the removal of infixes, a set of transformation rules were proposed. Evaluation of the proposed stemmer has shown that it does in fact achieve significantly higher accuracy when compared to a light stemmer. It has also shown, that improved stemming accuracy leads to significant improvement in the task of automatic annotation.

Future work will mainly focus on means of making the stemmer even more accurate. Experimenting with modifying the stemmer and testing it on the task of information retrieval is also planned. In addition, some modifications to the stem building application so as make the stem list building process even faster, are intended. One of the important modifications is to offer alternative stems for each word in case a word matches with more than one pattern.

Acknowledgements

This work was supported by the Center of Excellence for Data Mining and Computer modeling within the Egyptian Ministry of Communication and Information (MCIT).

Bibliographical References

El-Beltagy, S. R., Hazmam, M., and Rafea, A. (2007). Ontology Based Annotation of Web Document Segments. In proceedings of the 22nd Annual ACM Symposium on Applied Computing (pp. 1362-1367), Seoul, Korea.

El-Beltagy, S. R., and Rafea, A. (2009). "KP-Miner: A Keyphrase Extraction System for English and Arabic Documents, Information Systems, 34(2009), 132 -144.

Goweder, A., Poesio, M. De Roeck, A. Reynolds, J. (2004). Identifying broken plurals in unvowelised Arabic text. In proceedings of EMNLP, Barcelona, Spain.

Khoja, S. and Garside, R. (1999). Stemming Arabic text. Computing Department, Lancaster University, Lancaster, United Kingdom.

Larkey, L. S., Ballesteros, L., and Connell, M. E. (2002). Improving Stemming for Arabic Information Retrieval: Light Stemming and Co-occurrence Analysis. In proceedings SIGIR'02. Tampere, Finland.

Larkey, L. S. and Connell, M. E. (2001). Arabic information retrieval at UMass in TREC-10. In TREC 2001. Gaithersburg: NIST.

Wikipedia (2008). Wikipedia, the free encyclopedia. http://ar.wikipedia.org/wiki/Main_Page

Xu, J., and Croft, W. B. (1998). Corpus, based stemming using co-occurrence of word variants. ACM Transactions on Information Systems , 16(1), 61- 81.