Arabic Language Analysis Toolkit

HAJDER SHOUHANI RABIEE

KTH Computer Science
and Communication

Master of Science Thesis
Stockholm, Sweden 2011

Arabic Language Analysis Toolkit

H A J D E R S H O U H A N I R A B I E E

Master's Thesis in Computer Science (30 ECTS credits)
at the School of Computer Science and Engineering
Royal Institute of Technology year 2011
Supervisor at CSC was Viggo Kann
Examiner was Stefan Arnborg

TRITA-CSC-E 2011:133
ISRN-KTH/CSC/E--11/133--SE
ISSN-1653-5715

Royal Institute of Technology
School of Computer Science and Communication

KTH CSC
SE-100 44 Stockholm, Sweden

URL: www.kth.se/csc

Abstract

Arabic Language Analysis Toolkit

In this project report I investigate the possibility of creating a language analysis toolkit for Arabic by integrating existing tools and resources. The investigation shows that there are actually few tools and resources for public use and fit for this purpose. Integrating the "best" tools was not straightforward, a lot of problems exist in the area and complicated the integration process. These were mainly due to the lack of different input/output formats of every tool, granularity of tag sets and tokenisation. Despite the complications faced, I present an integrated solution consisting of a part-of-speech tagger and a morphological analyser. The toolkit was trained on classical Arabic and tested on a sample text of modern standard Arabic. The results of the experiments are not that impressive, however the report outlines the difficulties of integrating tools today and in the end the project achieved its main objective.

Sammanfattning

Språkanalysverktygslåda för arabiska

I detta examensarbete undersöker jag möjligheten att skapa en språkanalysverktygslåda för arabiska genom att integrera existerande verktyg och resurser som är baserade på öppen källkod. De verktyg och resurser som använts i undersökningen hade alla olika format av indata/utdata, detaljnivå av taggar och olika tokeniseringsmetoder. I undersökningen beskrivs att integrationen av de "bästa" verktygen inte var rättfram pga dessa olikheter mellan resurserna. Trots dessa svårigheter, presenterar jag en språkanalysverktygslåda bestående av en part-of-speech taggare (PoS) och morfologisk analysator (MA). PoS:n tränades på klassisk arabiska och användes för taggningsexperiment på texter tillhörande kategorin modern standardarabiska. Resultaten av experimenten är inte imponerande men projektet anses ha uppnått sitt främsta mål ändå.

Acknowledgments

This report was written as a MSc degree project for CSC, KTH within the field of Natural Language Processing. The major part of the work has been conducted at Leeds University under the supervision of Dr. Eric Atwell. I would like to thank him for his support during the whole project and always having ideas of "one more paper related to XYZ" that was useful. He had a major role in the admission process to Leeds University, for which I cannot thank him enough for. I would also like to thank all people who made the visit at Leeds a pleasant one.

My supervisor at KTH, professor Viggo Kann, whom I must thank for his support during the whole project and especially at a point when the project seemed to have failed and my heart was utmost filled with anxiety - thank you for helping me fulfill the project and reach the goal. Also my examiner Stefan Arnborg for his support in creating a consistent report and his enormous patience in spell checking the report.

Last but not least - my family whom I cannot thank enough.

Contents

Chapter 1

Introduction

Natural Language Processing (NLP) is a young branch of computer science, it combines computer science, linguistic theory and mathematics. The aim is to equip computers with the same linguistic knowledge as humans have, in order to simplify many mechanical tasks that humans are already capable of. Applications such as text summarization, machine translation and spelling correction are popular tools available today that utilise NLP. The transition to computers has also allowed for new applications such as text categorization. The future holds probably many new computer applications based on NLP.

It is estimated that about 220 million people(Lewis, 2009) are Arab speaking and that Arabic is the fourth most spoken language, thus it is a major international modern language. It is also recognized as one of the six major official languages of the United Nations. English on the other hand with 330 million native speakers(Lewis, 2009) has received an unproportional attention when it comes to the development of NLP tools and resources. The tools for Arabic are few and often miss certain features or do not live up to the same standard as their English counterpart (Atwell et al., 2004). There is a need for improvements in several areas of Arabic NLP and this degree project will focus on investigating existing tools in order to build a new toolkit and suggest ideas for future work in the area.

1.1 Motivation

The number of computing resources for the Arabic language are few, considering the amount of people it is spoken by. Thus far, the tools which have been developed all have their caveats, e.g. restricted input encoding (only Arabic Windows), licensing issues or a transliteration table that is often used. There is a need for more tools for the

Arabic language in order to start creating new and better NLP applications. Difficulties for researchers lie in Arabic being a morphologically complex language, the non-Roman script is also one of the things that non-native speakers might find difficulties working with (Atwell et al., 2004; Sawalha and Atwell, 2010).

A better set of tools will in the first place support further research into the area, for example support corpus linguistics researchers in analysis of Arabic texts. Secondly it lays the ground for creating new and better end-user application.

1.2 Problem

One of the fundamental parts of any linguistic application is the Part-of-Speech tagger (PoS tagger) which in turn is dependent on a morphological analyser which utilizes dictionaries for lookup. The three components all exist today and have been developed by previous researchers. However, many tools which have been developed are either in closed projects or not freely available.

The problem in this project is to investigate what open-source tools exist today for Arabic NLP, especially PoS taggers and morphological analysers, to compare them with regard to several aspects e.g. how easy it is to get hold of, which algorithm/model is used, how difficult it is to adapt into other tools, for which purpose it is suitable etc.

For the tools that lack certain features in accordance to today's standards or don't perform well, an improvement shall be suggested or made. For PoS tagger this may for example mean retraining with a different tag set, resulting in better performance. The ultimate goal is integration of a PoS tagger and a morphological analyzer for Arabic into an existing toolkit e.g. NLTK[1]

1.3 Outline of the report

The report is divided as follows: in the first part a technical background of the existing tools and resources is presented. In particular chapter two lists the existing PoS taggers and morphological analysers and their features. Evaluation methods and previous surveys of tools and resources is also covered in the same chapter. The second part describes my approach to solving the problem, the experiments and the results. Readers already familiar with the NLP area may skip the technical background.

[1]Natural Language Toolkit, http://www.nltk.org

Part I

Background

Chapter 2

Existing resources

In this chapter I present some of the most important tools that exist today for Arabic Corpus Linguistics (ACL). The chapter focuses on PoS taggers and morphological analysers. Important features are summarised as a table at the end of this chapter. The tools which have attracted the attention of this project are presented below. Each tool was examined in relation to (i) user experience, defined by integration into other applications (if applicable) (ii) algorithm used and model based on (if applicable) (iii) popularity amongst the community (iv) availability (v) what purpose it is suitable for (vi) which tag set is used (if applicable)

Readers not familiar with Arabic are encouraged to take a look at Appendix B first.

2.1 Earlier surveys

In (Atwell et al., 2004) an outline of some of the most important tools is presented, which has served as a starting point for my own survey. Furthermore (Al-Sughaiyer and Al-Kharashi, 2004) report in their survey findings that many tools are only described generally with no measures of effectiveness and provide little in-depth investigation of available techniques. Very few stemmers were available for them during the survey, they also claim many researchers don't acknowledge the efforts of other and no systematic approach of evaluating algorithms exists either. Additionally the lack of standards is criticised.

2.2 Morphological analysers

In a linguistic application, a morphological analyser (MA) analyses a word and outputs morphological information about the word in terms of class, number, gender etc. It

w	wa	Pref-Wa	and <pos>wa/CONJ+</pos>
f	fa	Pref-Wa	and;so <pos>fa/CONJ+</pos>
b	bi	NPref-Bi	by;with <pos>bi/PREP+</pos>
k	ka	NPref-Bi	like;such as <pos>ka/PREP+</pos>
wb	wabi	NPref-Bi	and + by/with <pos>wa/CONJ+bi/PREP+</pos>
fb	fabi	NPref-Bi	and + by/with <pos>fa/CONJ+bi/PREP+</pos>

Table 2.1. Sample entries for the prefixes

should also able to conjugate a word given its simplest form. E.g in English, producing the conjugation of the singular forms for the word "run": (they) run, (you) run, (I) run. For Arabic though, this is a more complex task as it is *morphologically complex* and has distinc forms for every number and gender.

In this section I briefly describe some of the popular MAs. Comparing MAs is not straightforward, as mentioned in (Al-Sughaiyer and Al-Kharashi, 2004) it is suggested to develop application oriented MAs, hence the requirements and evaluation grounds may be different.

The presentation in the following subsections starts out with a detailed explanation of the Buckwalter Morphological Analyzer, one of the first morphological analysers for Arabic. Only an overview is presented of the succeeding tools that have drawn inspiration from it.

2.2.1 Buckwalter Morphological Analyzer

The Buckwalter Morphological Analyzer (BAMA) 1.0 (Buckwalter, 2002) was released in 2002. It can be obtained by sending an inquiry to LDC. There is also a Java port versioned 1.2 written by Pierrick Brihaye available online called *Ararmorph*. The latest version BAMA 2.0 though is only available through LDC membership.

BAMA 1.0 comes with a short online documentation describing the design of the analyser. How to set up the analyser is not documented, I came across two different versions, one which was a CGI script that requires a web server to run and a second version which was a Perl script. Both packages come with lexicon files and the engine which is written in Perl, input is accepted in cp1256 (Windows Arabic) encoding only.

Aramorph takes care of the issues in BAMA 1.0 and comes with enough documentation on how to get started with the tool.

The lexicon is made up of three files, the first one contains all Arabic prefixes and different concatenations of the prefixes, see Table 2.1. The columns represent: the prefix without short vowels or diacritics; the prefix with short vowels and diacritics; the prefix's morphological category (for controlling the compatibility); its English gloss(es) including PoS data within tags. The second lexicon file contains all Arabic suffixes and their different concatenations (not shown). The third lexicon file contains all Arabic stems, see Table 2.2.

ktb	katab	PV	write
ktb	kotub	IV	write
ktb	kutib	PV_Pass	be written;be fated;be destined
ktb	kotab	IV_Pass_yu	be written;be fated;be destined

Table 2.2. Sample entries for the stems

INPUT STRING:	والغاز	
LOOK-UP WORD:	wAlgAz	
SOLUTION 1:	(wa>alogAz)	wa/CONJ+>alogAz/NOUN
(GLOSS):	and + mysteries/enigmas/riddles +	
SOLUTION 2:	(waAlgAz)	wa/CONJ+Al/DET+gAz/NOUN
(GLOSS):	and + the + gas +	

Table 2.3. Analysis report

Algorithm Many of the decisions are actually already in the lexicon and compatibility tables, therefore the algorithm is quite simple. Its steps include: tokenisation, word segmentation, dictionary lookup, compatibility check, analysis report, second lookup (orthographic variants). Table 2.3 shows the word segmentation and analysis report from the MA for an Arabic word, transliterated as "wAlgAz" using the Buckwalter Transliteration Table (appendix C). In the last two columns, the two analysis solutions, together with their morphological tags (conjunction, noun and determiner) are shown.

The Arabic words are segmented into their affixes and stem according to the following rule:

- the prefix can be 0 to 4 characters long
- the stem does not have any length limit.
- the suffix can be 0 to 6 characters long

The dictionary is then queried for the existence of each prefix, stem and suffix. If all three are found the next step is to determine if they are compatible with each other. This is done by asking the following questions:

- is prefix category A compatible with the stem category B?
- is prefix category A compatible with suffix category B?
- is stem category B compatible with suffix category C?

If the answer is yes to all questions, the word is valid. The analysis report outputs the different solutions for a word, see example below of the word.

If no analysis is found, the orthographic variants are looked up to check for different spellings of the word.

The first version of BAMA has several shortcomings, as witnessed by (Altabba et al., 2010). They report that since all derivations are hard coded instead of relying on rules, it makes the runtime processing long. Furthermore, the authors state that it has a spelling problem in conversions between Aleph and Hamza, the word Hadramout (حضرموت) (city in Yemen) is not analysed correctly, and problems when dealing with acts in the past tense where the pronoun is absent, e.g. حاول,اضرب . Their experiments also show that some problems occur with the past tense passive voice.

Many of the shortcomings mentioned by (Altabba et al., 2010) can probably be remedied if the lexical files would not apply a coarse representations of the affixes; collecting clitics together with prefixes or suffixes is not the best way. As argued by (Sawalha and Atwell, 2010) a more fine-grained representation of words is needed.

Despite all this, the analyser definitely fills a purpose, perhaps not for machine translation, but information retrieval where the requirement of exact analysis is not as important.

2.2.2 Alkhalil Morphological Analyzer

The Alkhalil Morphological Analyzer is written in Java. The lexical resource consist of several classes that handle voweled and unvoweled patterns for both verbal and nominal words; Arabic word roots and affixes. Analysis is carried out in the following steps: preprocessing, removal of diacritics; segmentation, each word is considered as (proclitic+stem+enclitic) and aims to identify them; analysis of the stem; result screening; display of result (Boudlal et al.)

According to (Altabba et al., 2010) the Alkhalil analyzer is the best one, although it has some problems with its database. It won the first prize at a competition by The Arab League Educational, Cultural Scientific Organization (ALESCO) in 2010. It has some limitations such as it does not provide PoS tags in good reusable format. Neither does it differentiate between clitics and affixes fully.

The software and its documentation has and can be obtained from the developers (Boudlal et al.).

2.2.3 Xerox Arabic Morphological Analysis and Generation

This is a web based analyzer[1], queries are easily entered in the interface and supports several encodings, such as cp1256 (Windows Encoding) and the Buckwalter transliteration, see Appendix C.

It is a large-scale system using finite state lexical transducer, which basically is a mathematical model used to design programs that can be represented by states and the transition between them. The machine is adapted to the Xerox finite-state format which is claimed to be more suitable for the purpose of doing morphological analysis, it is further described in (Beesley, 2001).

[1] http://open.xerox.com/Services/arabic-morphology/Consume/218

2.2.4 ElixirFM

This functional morphology analyzer was written as a part of a PhD thesis at Charles University, Prague by Otakar Smrž, it is further described in (Smrž, 2007). It is open-source, the engine uses Haskell Functional Morphology(Forsberg and Ranta, 2004) library and the web interface is written in Perl, therefore if one is not comfortable with these languages then a higher learning curve should be expected.

ElixirFM uses paradigms, grammatical categories, lexemes and word classes to model inflection and derivation. The lexicon reuses the Buckwalter Lexicon(Buckwalter, 2002) enhanced with the Prague Arabic Dependency Bank. In Haskell a domain-specific language is specified, combining lexical definitions as source code, data structure for independent processing and a document resembling the printed dictionaries.

2.3 Part-of-Speech taggers

In part-of-speech (PoS) tagging the goal is to assign each word in a text a word class (part of speech / tag). Enriching a text with tags fills several purposes and is a very important resource for finding certain patterns in a language, analysing word frequencies, syntactical structures and other analysis.

Every PoS tagger uses a tag set, which is different for every tagger. In order to compare taggers objectively, there must be a common ground, in this case the same tag set. Moreover, they should be evaluated on the same piece of text. The tag set plays a key role; a tagger which uses a smaller tag set, has less tags to choose from during assignment, hence the probability of assigning a "wrong" tag is lower.

Below I present some of the most known taggers that have been encountered during the background research.

2.3.1 Stanford Part-Of-Speech tagger

This tagger was originally developed for English at Stanford University and is described in (Toutanova and Manning, 2000). The improved version, which is described in (Toutanova et al., 2003) adds support for other languages together with speed and usability improvements.

The tagger is based on the maximum-entropy model.

> *the intuition of maximum entropy is to build a distribution by continously adding features*. (Jurafsky and Martin H, 2009)

In this context, 'features' refers to the constraints that are activated when training the tagger, e.g morphological and syntactic features. Each feature adds a constraint to the total distribution. For further information refer to (Jurafsky and Martin H, 2009; Berger et al., 1996).

The latest version comes with trained models for Chinese, German and Arabic, the authors claim in the README file a 96.50% accuracy on Arabic. The tagger was trained on the training part of the Arabic Penn Treebank (ATB). It uses augmented

Bies (Bies) mapping of ATB tags, which is not so fine-grained, as the authors also state. As an example it does not tokenize clitics when tagging, e.g. the word بسم is tagged as noun, while it should be separated into the proclitic and noun as ب+سم; tagging it as *preposition* and *noun* respectively. As stated above, a smaller tag set makes it harder to assign a "wrong" tag, thus it is one factor contributing to the high accuracy.

The documentation for the tagger is clear and to start tagging is just a a matter of a command. It is also possible to retrain the tagger.

2.3.2 Brill Tagger

This tagger combines the ideas of rule-based tagging with a general machine-learning approach which is *transformation-based*. The idea behind is to initially let the text pass through a annotator, in part-of-speech context this might be assigning each word its most likely tag. Then the text is compared to the gold standard (the "best" tagging"), in order to create *transformations* that can be applied to improve the initial text as much as possible. A transformation is described in (Brill, 1995) it consists of two parts, defined as:

a rewrite rule - e.g. change the word from modal to noun
a triggering environment - e.g. preceding word is a determiner

An *objective function* is used to rank a transformation, e.g the number of errors made. Thus, the transformation with the *best score* is chosen. This stage is repeated until a condition is fulfilled; the result being a transformation list. The tagger is available for the public and freely and it's easy to get started with.

2.3.3 Khoja Arabic Part-Of-Speech Tagger

This tagger also combines a statistical and rule-based approach as it is proven to produce the best results(Khoja, 2001). The tagger uses a tag set of 131 tags, which is derived from traditional Arabic grammatical theory.

Four different corpora were used for testing, the largest consisting of roughly 59,000 words was used to train the tagger and to build various lexicons which is used when tagging the test set. One version of a lexicon lists each word together with all tags it has received in the corpus. This was used in the initial stage of the tagging by looking up each word in the lexicon and give all possible tags as specified in the lexicon. Stemming was used for words which were not found in the lexicon. For disambiguating tags, the Viterbi algorithm was used together with lexical- and contextual probabilities. In all, the tagger achieved an accuracy of around 90%. For further detail, see (Khoja, 2001).

2.3.4 TreeTagger

This is a language-independent tagger by (Schmid, 1994) and is based on decision trees. The tagger has been successfully trained and tagged many European languages and it's

adaptable to other languages if a lexicon and a manually tagged training corpus are available.

The tagger comes with code for different architectures such as PC, Solaris and Macintosh. This gives an indication that it's likely written in C/C++. Wrappers for Java and Python are available by the community.

2.3.5 Other taggers

There do exist other taggers for Arabic, some novel ideas are described in (Algrainy et al., 2008; Yousif and Sembok, 2008) which report accuracy an of 91% and unbelievably high as 99.99% respectively. The first one uses a pattern-based approach to tag a word, without using a huge manually annotated lexicon. This is an interesting approach but could not be tried in this project.

The second one uses a Support Vector Machines (SVM) approach and the tag set described in (Khoja et al., 2001) with 177 tags in total, 103 nouns, 57 verbs, 7 residual and 1 punctuation.

2.4 Evaluation methods

There exist different ways of evaluating a tagger, among the most common are precision, recall and accuracy/success rate. The success rate is defined over all tags and is evaluated by comparing the tags assigned by a tagger with a gold standard - a file with each word assigned its 'best' tag.

$$\text{Success rate} = \frac{\text{Correctly tagged tokens}}{\text{Total number of tokens}}$$

For a better understanding of how well a tagger performs, one can use tag-wise evaluation. Tag-wise measurement is a good way of evaluating a tagger, because by measuring one tag at a time one can get a better picture of what tags are harder to distinguish than others. The error measures are *precision* and *recall*. Precision is the fraction of tokens tagged T in the gold standard of those tagged T by the tagger. Recall is the fraction of tokens tagged T by the tagger of those tagged T in the gold standard.

In formulas we write them as following:

$$\text{Precision}_T = \frac{\text{TP}}{\text{TP} + \text{FP}} \qquad \text{Recall}_T = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- True positive (TP): words tagged as T both in the test set and by the tagger.
- True negative (TN): words tagged as non-T both in the test set and by the tagger.
- False positive (FP): words tagged as non-T in the test set and as T by the tagger.
- False negative (FN): words tagged as T in the test set and as non-T by the tagger.

One can also combine the two measures and calculate something called F-measure.

$$\mathrm{F-Measure}_T = \frac{2 \cdot \mathrm{Precision}_\mathsf{T} \cdot \mathrm{Recall}_\mathsf{T}}{\mathrm{Recall}_\mathsf{T} + \mathrm{Precision}_\mathsf{T}} = \frac{2TP}{2TP + FP + FN}$$

Baseline Usually as an initial performance measure, a so called *baseline* is calculated and compared against new results from the same tagger but with improved functionality. A typical baseline is assigning each word the most frequent tag.

Evaluating MAs on the other hand, is quite different, since the requirements of an analyser may differ for every application. How many solutions should a MA output? What is a good solution? How fine-grained should it be? These are questions whose answers may vary depending on application and will therefore not be analysed in this project. For the interested reader Ripper (Cohen, 1996) is a method to determine whether the analysis from a MA is "good" or "bad".

2.5 Toolkits

A NLP toolkit usually incorporates several tools for various computational linguistics problems such as a tokenizing, part-of-speech, named entity recognition and parsing. Here I briefly present the three popular NLP toolkits.

2.5.1 Natural Language Toolkit - NLTK

The NLTK is a toolkit written in Python and consists of several modules and data sets, and covers both symbolic and statistical NLP. The former type is based on rules and a deterministic approach for performing analysis and tagging while the latter is a probabilistic approach. It is a widely used toolkit, used in over 200 countries and has become very popular among teachers and researches around the world. The documentation is comprehensive and is supported by a wide global community as well. Currently it only supports NLP tasks for English (Bird and Loper, 2004).

2.5.2 Stanford CoreNLP

This is another suite for English which includes all tools from the Stanford NLP group in one toolkit and it consists of a tokenizer, PoS tagger, named entity recognizer, parser, and coreference module - all in Java. The design is aimed at flexibility and extensibility but it makes heavy use of RAM, recommended amount is at least 3GB (!), depending on the size of the text being processed.(CoreNLP)

2.5.3 General Architecture for Text Engineering - GATE

GATE is also open source and over 15 years old, it is suitable for all types of computational tasks as the designers claim. It has the largest community and more than 5 million euro have been invested in the project. GATE Developer is the IDE which integrates an

information extraction system with several plugins e.g. WordNet and Arabic tokeniser. The GATE architecture is written in Java.(GATE)

2.6 Other resources

If we come to look at the situation of corpora and stemmers, the situation is similar (Al-Sughaiyer and Al-Kharashi, 2004), or even worse in the case of corpora. Not a single tagged MSA corpus exists freely or publicly. The only exception is Shereen Khoja who distributes her 50000 word tagged corpus for research purposes, however no reply has been received at the moment.

The Holy Quran though is the only open domain Arabic text, however it represents classical Arabic text, which is not exactly the same as MSA when it comes to vocabulary and grammar.

The above background survey is summarized in Table 2.4. The last column Testing indicate how easy it was to get started with the tool. In terms of MA this means just generating the output from it. In terms of PoS taggers it means how easy it was to retrain and testing on sample texts.

2.7 Difficulties

The background research shows that difficulties in trying to integrate the tools lie in the way each tool represents the texts. The morphological analysers use different encodings, e.g. CP-1256, UTF-8, ISO-8859-6 or different alphabet, e.g. Buckwalter transliteration (ASCII) (see Appendix C and Arabic alphabet. The tokenisation algorithms are also different for each tool, leading to a different analysis granularity, hence a different tag set. As this is a basis for evaluation, one can understand where the problem lies when trying to evaluate taggers on a common ground. Also this is a problem when it comes to integration of the tools. In this project where my objective is to integrate a morphological analyser with a PoS tagger there is a direct problem when the output of one tool does not fit the input of the other.

The morphological analysers use a tokenisation based on the prefix+stem+affix but with slight modifications between them. In (Habash and Rambow, 2005) the following is pointed out

> There is not a single possible or obvious tokenisation scheme: a tokenization scheme is an analytical tool devised by the researcher.

Which basically explains the difference between the tools, also summarised in Table 2.5, although the same type of input/output appears to be used, there is always another task of adjusting the text into the correct format.

Name	License	How obtained	Language	Portability	Complexity of portability	Testing
BAMA 1.0	GPL	Contacted LDC twice (one reminder)	Perl	Perl interpreter needed	Integration with Java through API	Easy, good documentation. Only cp-1256 encoding accepted though.
BAMA 1.2 (Aramorph)	GPL	Available from homepage	Java	Java Virtual Machine needed only	Easy to integrate with other Java apps.	Additionally supports UTF-8 encoded files and input.
BAMA 2.0	LDC membership	LDC membership	Perl	Perl interpreter	Same as BAMA 1.0	N/A
Alkhalil	GPL	Email the authors	Java	Java Virtual Machine only	Including into other Java apps is fine	Easy interface, short and concise documentation.
Xerox MA	?	Online, can be used as web service	PHP	SOAP API	Web interface	Accepts input in several encodings, output good. Documentation on creating clients.
Stanford PoS tagger	GPL	Stanford NLP group	Java	JVM	N/A	English easy. Arabic retraining easy but exception raised when testing, author contacted.
BrillTagger	Free for non-commercial use	Online	C/C++	Not platform-independent	Architecture modifications needed	Not possible to build without modifying for my system. Skip
TreeTagger	GPL	Online	C/C++	Wrappers for Java and Python exists	Java wrapper OK, Python not tested	Problems during retraining but good support from the author.
Khoja APT	GPL	Waiting for reply still	?	?	?	No reply
SVM Tagger	?	Waiting for patent until release	?	?	?	In the meantime, sending encoded data and wait for reply - bad !

Table 2.4. Comparison table of existing resources

Name	Input	Tokenisation and output	Tagset
BAMA 1.0	CP-1256 encoding, Arabic input only	English, Transliteration Prefix+Stem+Suffix	Buckwalter morphological tags
Xerox MA	UTF-8, Buckwalter, ISO-8859-6	Finite State Automata, similar to, Prefix+stem+Suffix	Buckwalter similar
ElixirFM	UTF-8,Buckwalter,ArabTeX	Prefix+Stem+Suffix but in detail may be in contrast to other as it claims	Buckwalter morph tags
Stanford	UTF-8	UTF-8, Assumes already tokenised according to Penn ATB, i.e. only does white-space	ATB collapsed tags - "Bies tags"
BrillTagger/TreeTagger	Independent	Whitespace	Depends on training data

Table 2.5. Input and output of tools

Part II

My contributions

Chapter 3

Method

In this chapter I describe the methods I have used in trying to solve the problem of building a toolkit. The definition of an analysis toolkit for languages is not entirely precise and is open for interpretations. But it indicates the inclusion of resources that can be used for the development of NLP applications. Some of the existing toolkits are referenced in section 2.5. All three toolkits referenced in that section include fundamental resources such as analysers, i. e. PoS taggers and corpora.

3.1 The method

As mentioned in section 1.2 the focus of this project has been to look at the fundamental tools, i. e. PoS tagger and MA, with the aim of integrating the two into an existing toolkit.

With the table presented in Table 2.4 in mind, the first decision on what tools the toolkit should incorporate was made. The decision was based foremost on the tool's usability aspects, both in terms of usage and integration with other tools. For the taggers, this is further motivated by comparing the accuracy for the ones mentioned in section 2.3, all lie within the 95-97% (except for the pattern tagger with 91%) accuracy range, therefore the usability and portability were the best criteria for eliminating candidates.

After the elimination phase a thorough inspection of the underlying algorithm and source was conducted.

3.2 Solution - building the toolkit

The goal of finding tools that were written in an interpreted language or were platform-independent led me to the Alkhalil morphological analyser and the Stanford PoS tagger,

the first one was picked because of its availability, portability and good support from the authors. The Stanford PoS tagger additionally seemed good as it belongs to a renowned NLP group and as the authors claim performs very well on Arabic. Furthermore it is written in the same language as the morphological analyser (Java). I anticipated assembling the two would make a perfect portable toolkit.

The main aim of the toolkit would be to see how well a tagger can perform on unseen texts, i.e. tagging texts from a different lexicon than the tagger was trained on. Only integrating the two into a bundle and calling them separately via an API did not pose a big challenge. I was motivated by (Habash and Rambow, 2005) who reported positive results on using a morphological analyser during the tagging process, their work is based on (Hajič, 2000) who argues that a morphological analyser aids the morphological disambiguation process during tagging. One of the ideas initially was trying to create plugins of the tools and integrating them into GATE, which is also written in Java. However GATE is slow when processing Arabic and lacks a good user interface. The first impression was not satisfactory - GATE was an option I had to forego as a toolkit candidate to integrate into. Integrating tools into NLTK was not realistic either at this stage, since it is written in Python, it did not fit with the choice of the PoS tagger and MA I had made, which are written in Java. Finally, adapting the NLTK to Arabic seems to be a long way, according to a response on the NLTK mailing list regarding this:

> But the only time I've tried reading an Arabic text into the NLTK it didn't work too well...NLTK will get a huge boost in terms of its linguistic applicability once people rewrite it for Python 3, since Python 3.x is natively unicode.[1]

3.2.1 Training corpus

The first thing I set out to do was finding a corpus for retraining the tagger for Arabic. This was not an easy task but was a part of the background research. As explained in the technical background chapter 2, there are not many resources publicly available. The only corpus option I had was the Quranic Arabic Corpus (Dukes) for retraining the tagger. The corpus has 77430 words each annotated with tag, prefix, lemma and is fully diacritized. I only performed a whitespace tokenisation, this has the drawback of the tagging not being very fine-grained. For a language as Arabic that has many affixed words, it would mean that they (the affixes) are discarded during analysis. However, for the purpose of this project, whose main goal is to integrate two tools and not focusing on a detailed (fine-grained) tagging, the decision was justified. This should make it easier to extract the correct output from the MA as well.

Worth mentioning about the Stanford tagger is that it already comes with an Arabic model, however it is trained on sections from the Penn Arabic TreeBank parts that already is Modern Standard Arabic. Again, for experimental purposes I was really

[1] At the time of the project Python 2.7 was the stable version

interested to see how well the Stanford tagger can perform on MSA when trained on Classical Arabic (CA) and using a morphological analyser.

3.2.2 Retraining the tagger

Retraining the tagger was not as straightforward as anticipated, errors occurred during training and testing due to some minor issues with the training program such as missing settings in the properties file that were not entirely clear in the documentation. The corpus file itself also contained extra trailing characters after the process of reformatting the corpus to the correct format for the training program. These trailing characters were not easily spotted, but credit goes to the great support from the Stanford NLP mailing list on this issue.

3.3 Preparing the morphological analyser

Despite the fact that the morphological analyser is written in Java, the possible integration turned out not to be straightforward. Preparation of the Alkhalil MA consisted of making it compatible with UTF-8. The source code was actually encoded in Windows CP-1256 (Windows Arabic) and decoding in a UTF-8 editor raised problems since the code contained logic that relied on the Windows encoding which led to the source being uncompilable. In particular there was logic related to a character being a solar character (see Appendix B) or not. All the logic related to extracting stems and affixes was also hardcoded with Arabic characters in the same encoding. As UTF-8 is also becoming the de facto encoding standard I thought a conversion of the source code encoding to UTF-8, was something that had to be done.

3.4 The final prototype

After preparing the tools, the next task was trying to integrate and test them as one. This led me to even more problems and a different understanding of the tagger. When inspecting the source code, doing initial tagging experiments and discussing with the authors I discovered that the tagger doesn't have a way of tagging unknown words for a language, i.e. words that were not encountered during training. The tagger develops rules from the training corpus and defines so called Extractors internally that recognise morphological features, these are sufficient for English, but certainly not for a morphologically complex language as Arabic. In the words of one of the authors[2] behind the Stanford PoS tagger regarding this - extracted from an e-mail conversation and also archived[3]:

> If the lexicons are different to the point that most words are unknown, the tagger is going to get most things wrong.

[2] John Bauer

[3] java-nlp-user mailing list

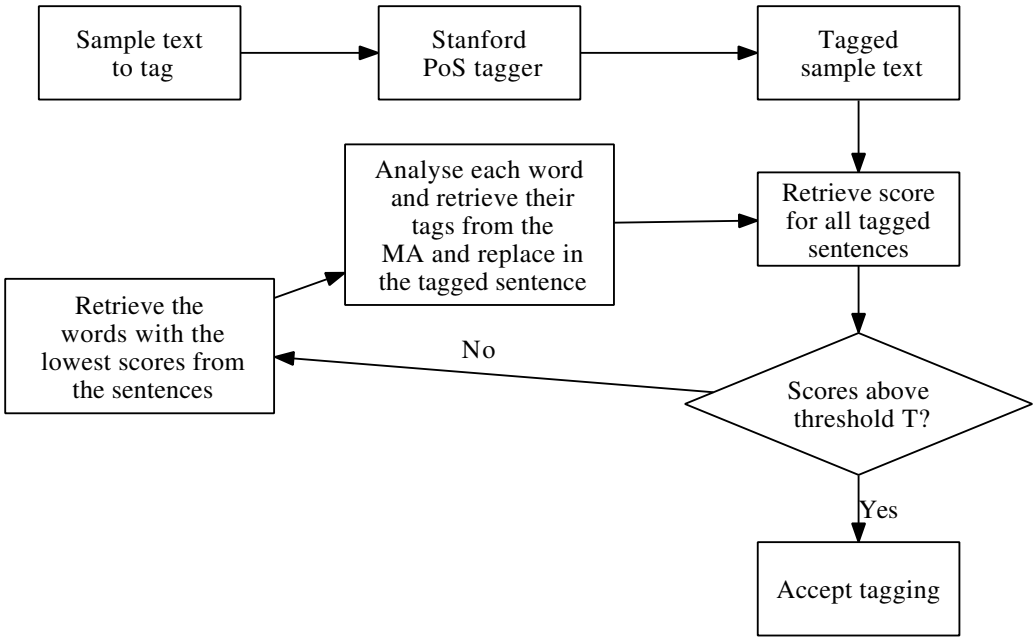

Figure 3.1. Initial thought of the integration between the MA and the PoS tagger

As can be seen in the Tagging Experiments Section 4.2, the initial experiment of tagging a text sample from a different type than the training lexicon resulted in very poor accuracy, see table 4.3 and the entry *Stanford on MSA*.

3.4.1 First attempt to improve

Following the poor initial tagging of the MSA sample text, an integration of a MA could now also be justified and not remain as a way of only experimenting with some tools. The kind of flow I first had in mind for this improvement to be realised is illustrated in Figure 3.1.

Through discussions with the authors it was concluded that my experiment would not be possible to carry out. In the words of one of the authors

> Also unfortunately, there is currently no API for getting a confidence in a particular tag's *score*, marginal probability, or any other useful measure like that.

Therefore I had to discard this approach and rethink my idea of integrating the two tools.

3.4.2 Second attempt to improve

After the failure of the first attempt one member of the Stanford NLP group suggested a different approach - not entirely in line with the idea in section 3.4.1 though. The idea was to try modifying the tagger by adding so called Extractors in the source code. Basically an Extractor can be used for extracting morphological features from the training data and later used in the model. I thought this was a great idea and could possibly be realised with the word lists from the Alkhalil MA and define Extractors with them. This meant the initial idea on trying to make the two tools work together was changed, the closest to an integration was trying to extract relevant information from the MA's word lists and thereby extending the tagger to recognise/learn words from MSA with Extractors. However, when trying to modify it I came to the conclusion that the nature of the Stanford tagger does not allow for these kind of modifications that I was looking for, and therefore not possible to continue with. It was simply not possible to integrate the MA in any way and building a toolkit.

3.5 Change of plan

At this stage there was a choice: should I report the failure and stop, or try something else? After consultation with my supervisors we decided that another tagger should be tested. This didn't mean a complete change of course, but the project goal of initially trying to integrate two tools into an existing toolkit had to be changed in order to fit a realistic time frame. We decided to continue on investigating potential components of a toolkit, and feasibility of integration only. To reiterate, the definition of what my toolkit would look like may seem a bit ambiguous but the focus of searching for tools has been on MA and PoS tagger. In the end, a toolkit is not more than the assembly of several tools. If I could find a MA and a tagger that worked together for the type of experiment I wanted to carry out, it would definitely be an interesting finding and something worth being called a toolkit.

3.5.1 Choosing a new tagger

There weren't any other taggers written in Java to choose from. I consulted the earlier survey table and decided to try the TreeTagger, which I had received great support on during the background research phase. The TreeTagger does not have any source code available for download, despite this fact it allowed for the usage of the MA in a neat way, it was possible to constrain a word's tags during tagging without any modification to the source code. The usage of the MA in this solution was a bit different than illustrated in Figure 3.1, yet sufficient for the purpose of this project, whose aim was to see how a tagger performs when tested on a different text than the one it was trained on, also when the training data is sparse. Constraining a word's tags was possible by appending tags in the text to tag, thereby overriding the lexical information in the tagger parameter file, see Figure 3.1 for an illustration of an input text file to be tagged. This immediately opened up for different possible experiments that could be carried out.

WORD1	TAG1
WORD2	TAG1 TAG2 TAG3
WORD3	TAG1 TAG2
WORD4	TAG1 TAG2 TAG3 TAG4
etc	...

Table 3.1. Sample input text with tag constraints of one tag

aback	RB aback
abacuses	NNS abacus
abandon	VB abandon VBP abandon
abandoned	JJ abandoned VBD abandon VBN abandon
abandoning	VBG abandon
...	...

Table 3.2. The full form lexicon of TreeTagger

In the next chapter many of the experiments revolve around modifying the appended word's tags to see which combination led to the best accuracy.

The TreeTagger requires three files for training. The first file is a full form lexicon where each line of the lexicon contains a word form together with tag-lemma pairs. Example in Figure 3.2

The second file is an open class tags file defining the tags that are open class which the tagger will use when it encounters unknown words, i.e. words not in the lexicon. This is the same as used previously with the Stanford tagger. The third file is the training corpus.

3.5.2 Choosing a new morphological analyser

At this stage I also made the decision to abandon the Alkhalil analyser and instead try the BAMA 1.2 because it outputs the POS tags in English and not as Alkhalil which outputs them only in Arabic. In the background research section (2.2.2) there's a comment by (Altabba et al., 2010) saying the POS tags are not in a good reusable format, unfortunately it was only until this stage that I discovered what it really meant. Having to map all the tags from Arabic to English would've been a very time-consuming task and not suitable for this type of project.

This choice would not have any impact in terms of integration. Instead of running the tagger, do an initial tagging and then improve the tagging with the help of a morphological analyser, as illustrated in 3.1. The idea now was to run the analyser on the input from the sample text file, analyse each word's output and append the best solutions to the sample text file. The newly annotated file was fed into the TreeTagger for tagging.

Another problem to solve was on how to select tags from the MA to append to the

MA output	Quran Corpus equivalent
NOUN	N
N_PROP	PN
VERB.*[4]	V
PREP	P
REL_PRON	REL
ADV	T
INTERROG_PART	INTG
NEG_PART	NEG
EMPHATIC_PARTICLE	EMPH
INTERJ	N
ABBREV	N

Table 3.3. The mapping from BAMA's tag set to the Quran Corpus' tag set

text file. The MA outputs several solutions and I couldn't only select one randomly. Although every solution from the MA is a correct analysis, appending every tag would make it harder for the tagger to predict the most likely tag. The idea is therefore to help the tagger by constraining the set of possible tags. The output order of the solutions from the MA is not an indicator of the order between the probability of the tags, the main purpose of a MA is only to output all possible solutions. However, I let the number of occurrences of a tag be the eliminating factor for choosing the best tag(s). This is related to the problem of mapping the tags to fit my tag set, from the morphological analysis it led to having to collapse many tags, the Table 3.3 contains the exact mapping that is performed. The ABBREV and INTERJ from the MA, does not have any equivalent in the Quran corpus tag set, I mapped them to the common tag N (noun). A minor mapping issue occured with the tag ADV (adverb). From the MA it was ambiguous due to the fact that the Quran Corpus tag set actually distinguishes between T (time adverb) and LOC (location adverb), the output from the MA does not produce such a separation of the adverb. I decided therefore on mapping all ADV to T because that tag was the most common in the training corpus (T=1115 vs LOC=656 times). Collapsing the tags made it easier for counting the occurrences for a tag because all features were removed, e.g. N_3PERSON_PL, N_2PERSON_SG are collapsed to N. They both contribute to the count of the N-tag. Choosing the X first tag(s) with highest count solved the problem of choosing the best tag(s) to be appended to the sample text file.

3.5.3 The new modified prototype

The new prototype that I built is able to tag Modern Standard Arabic text with a tagger trained on a different type of lexicon, namely Classical Arabic (Quran corpus). I also added an adjustable parameter for how many tags one want to append to the sample text. The tagger also accepts a probability for each tag. I used two different probability

distributions that can be appended to a tag: uniform, constant probability for all tags; frequency distribution, calculated as the number of occurrences of a tag divided between the number of occurrences of all tags.

Chapter 4

Evaluation

In this chapter I present all experiments conducted with my toolkit. The aim for the retrained tagger was never maximum performance as the characteristics of this project revolved around evaluating existing tools and resources.

4.1 Qualitative evaluation

What was first set out to do in this project was almost fulfilled in the end with the TreeTagger and BAMA. The integration to an existing toolkit remains still to be accomplished.

After all skepticism regarding BAMA at the start of the project, it turned out to not be so bad. When using it in this project I must say that I disagree about what (Altabba et al., 2010) says about BAMA. The exceptional cases highlighted in their report exist, but I argue that it definitely fills its purpose for many applications, e.g. this project, the toolkit was realised thanks to it. Furthermore an API for developers who wish to integrate the analysis toolkit in their applications was developed. For convenience and other users I made it possible to execute the tagger via command line. Although the code itself was not modified, as I first thought would be necessary when having to assemble different components, the constraining of tags turned out to be easily made by annotating the input data and this can be seen as an integration of a morphological analyser into a PoS tagger, thus resulting in a toolkit. The evaluation of the toolkit has not been made, that would require more time for distributing the package, let users test and awaiting feedback or bug reports.

The performance of the tagger is not that impressive, but it would still be interesting to see how well it performs on other types of MSA texts as well, when there are more tagged MSA corpora to experiment with.

Experiment name	Description
Stanford	Only the usage of Stanford tagger
TreeTagger	Only the usage of the TT
MA tagging	Let the MA choose one tag only.
TreeTagger and MA	Expand to let the MA add several tags.
TreeTagger and MA II	Also add uniform probability of a tag
TreeTagger and MA III	Also add freq. probability of a tag

Table 4.1. Experiments descriptions

4.2 Tagging experiments

The tagger was trained on Quranic Arabic (QA) which is both a smaller set than Modern Standard Arabic (MSA). The former also has a few more complex morphological and syntactic constructs, these are however less of a complication in comparison to the new words available in MSA, which includes *modern* words e.g. TV, mobile phone etc. From this perspective it would be interesting to see how the tagger - trained on QA - would perform on MSA together with the morphological analyser. The accuracy results from the initial tagging experiments are shown in Table 4.3. Below I present the details in the experiments carried out that are listed in Table 4.1. For the MSA sample text I have chosen an extract of an article from the Arabic BBC news site[1] containing 66 words, they were annotated by me and considered the *gold standard* during the evaluation.

4.2.1 The open tag set

The tag set used is a very simple subset extracted from the training corpus (Quran corpus) and described in (Dukes).

The open class words are a class that allow the addition of new words through derivational, inflection or borrowing new words from other languages, usually these are nouns, verbs and adjectives. Examples of closed class words are prepositions, pronouns and conjunctions. The tagger allowed for specifying the open class set and from the Quran Corpus I extracted those presented in Table 4.2. Note this is the open class tag set, it does not mean the only tags which a word can be tagged with, on the contrary many other tags are available for tagging, they belong to the closed tag set but that set is larger and therefore not presented.

4.2.2 Baseline

The baseline is simply tagging each word as N (noun).

[1] `http://www.bbc.co.uk/arabic`

Tag	Description
N	Noun
PN	Proper Noun
ADJ	Adjective
T	Time adverb
LOC	Location adverb
V	Verb
IMPN	Imperative Verbal Noun

Table 4.2. The open tag class

N/اجلاء N/الجيش N/محاولة N/بعد N/التحرير N/ميدان N/في V/المعتصمين T/الى N/الالاف N/انصم
الاسبيع N/ جلال N/ المعتصمون N/ منصبها N/ التي N/ الجيم N/وتفكبك V/المتظاهرين T/من N/الساحة
العسكرية N/الشرطة N/ وعناصو V/ المعتصمين T/ بين N/تدافع V/ وحدث V/الميدان T/ في N/ الماضية N/
T/ من V/ تبقى N/ ما N/ وازالة V/ الميدان T/ الى N/السير N/ اعادة N/ الى V/ تسعى N/ التي N/
فيه N/ بدأت N/ الذي V/ الوقت T/في N/ المحتجين N/ وطوقوا N/ طوابيرا N/ الجنوب V/وشكل V/جيم
V أسبوعين T/ من N/ أكثر N/ منذ N/ مرة V/ لاول N/للميدان V/تعوج N/ السيارات N/ حركة N/

Figure 4.1. Stanford on MSA tagging

4.2.3 The first experiment

The first test was conducted with the Stanford tagger before the project plan was changed. The idea was to see how well the tagger performs on a sample from the Quran itself. This was carried out in two rounds. One with a model that included the test sample and another where the Quran corpus was divided into a training and test data set. In both rounds the tagger successfully tagged 49 words out of 50, a high performance as expected.

4.2.4 Second experiment

Now the idea was to see how well the Stanford tagger performs on another piece of Arabic text, namely MSA text and a sample that is not part of the training data. The initial expected result on this was low, which was confirmed by running the tagger and comparing to a gold standard.

I have chosen the small sample of 66 words to include here to illustrate the output from the Stanford tagger, see Figure 4.1. Many words are tagged as N and V, it is evident that the tagging is not the best.

It was after this experiment when I tried to improve the performance, I discovered that it's not possible to do so, see explanation in section 3.4 for this. This experiment

is called *Stanford MSA* in Table 4.3 and the results are even lower than the baseline!

4.3 The new solution

After abandoning the Stanford tagger and the Alkhalil MA it was time to evaluate a new solution based on the TreeTagger and BAMA.

4.3.1 First experiment

The results from the first experiment can be found in Table 4.4. A `-` mean that tag was never assigned to a word during tagging and is not the same as 0% which means a word was a assigned that tag but always wrongly.

A very naïve heuristic of counting the number of times a tag class occurred was used for choosing the tag to append to the file. The results were better than the baseline and also the Stanford MSA, perhaps due to the small tag set, the accuracy for this test called *MA tagging* resulted in an accuracy of 69%, see Table 4.3. The reason for the name is that even though the TreeTagger is fed with the appended text sample, one tag mean that it is the most probable tag (100%) and therefore no decisions are being made by the tagger itself. The experiment is entirely based on the decisions by the MA.

4.3.2 Second experiment

In this experiment I augmented the solution from previous experiment by involving the tagger by choosing more than one tag from the MA to append to the sample text file. The results are presented in Table 4.5.

4.3.3 Third experiment

Keeping the settings and parameters from the last experiment I was interested to see if the tagger could make better decisions by using the probability flag for each tag. The idea is to append a tag's probability in the text to tag and help the tagger choose the right tag. I choose two different measures, the first one was uniform distribution (all tags are equally likely) and frequency distribution (number of times a tag occurs / number of occurrent tags) where the results are presented in Table 4.6 and Table 4.7 respectively. We see that the difference is not that much higher and the frequency distribution is slightly better.

4.4 Analysis of output

The following section is dedicated to illustrate the different outputs of the program with the different options that were available and also included in the final application.

One example of a file of the format which TreeTagger accepts is shown in Table 4.8

When more than one tag is appended to the sample text file, the tagger will be involved in making decisions between the different tags. When only one tag is chosen

Experiment	Accuracy (%)
Baseline on MSA	44
Baseline on QA	36
Stanford on QA	98
TreeTagger on QA	96
Stanford on MSA	39
TreeTagger only on MSA	35
BAMA only on MSA	69

Table 4.3. Initial experiments accuracy

Tag	Precision (%)	Recall (%)	F-Measure (%)
N	77	82	79
PRON	100	50	67
ADJ	0	0	0
LOC	-	-	-
T	-	-	-
V	73	73	73
P	72	72	72
IMPN	-	-	-

Table 4.4. MA tagging experiment on MSA text

Tag	Precision (%)	Recall (%)	F-Measure (%)	Accuracy (%)
N	76	89	82	
PRON	100	25	40	
ADJ	0	0	0	
LOC	-	-	-	73
T	-	-	-	
V	82	60	69	
P	79	100	88	
IMPN	-	-	-	

Table 4.5. MA tagging and tagger experiment with three appended tags on MSA text, no probabilities.

Tag	Precision (%)	Recall (%)	F-Measure (%)	Accuracy (%)
N	75	86	80	
PRON	100	25	40	
ADJ	0	0	0	
LOC	-	-	-	71
T	-	-	-	
V	90	60	72	
P	79	100	88	
IMPN	-	-	-	

Table 4.6. MA tagging and tagger experiment on MSA text, three appended tags with uniform probability distribution

Tag	Precision (%)	Recall (%)	F-Measure (%)	Accuracy (%)
N	75	86	80	
PRON	100	25	40	
ADJ	0	0	0	
LOC	-	-	-	73
T	-	-	-	
V	91	67	77	
P	85	100	92	
IMPN	-	-	-	

Table 4.7. MA tagging and tagger experiment on MSA text, three appended tags with frequency probability distribution

and input to the tagger, the tag's probability is implicitly 1 hence that experiment is called *MA only*, because it's basically only the output from the MA which is considered, as explained in section 4.3.1.
Another configuration for the experiments was specifying the number of a word's tags to be appended to a file. These tags could also be annotated with a probability derived from the morphological analyser, see Table 4.9 and the example entry بعد. It has the uniform probability 0.25, and not 0.33 (1/3) as one might think. Actually that indicates that there were more solutions for the word's tags - 4 in this case, which could be output for that particular word. But they were not since the limit was set to three tags.

The same sample but with frequency distribution of the tags is visible in Table 4.10.

Word	Tag
انضم	V
الالاف	N
الى	V
المعتصمين	N
في	P
ميدان	N
التحرير	N
بعد	P
محاولة	N
الجيش	N
اخلا	V
الساحة	N
من	P
المتظاهرين	N
وتفكيك	N
الخيم	N
التي	N
نصبها	V
المعتصمون	N
خلال	P
الاسابيع	N
الماضية	ADJ

Table 4.8. Output of the appended tags file

Word	Tag
انضم	V 1.0
الالف	N 1.0
الى	V 0.5 P 0.5
المعتصمين	N 1.0
في	P 0.5 N 0.5
ميدان	N 1.0
التحرير	N 1.0
بعد	P 0.25 N 0.25 T 0.25
محاولة	N 1.0
الجيش	N 1.0
اخلا	V 1.0
الساحة	N 1.0
من	P 0.33 INTG 0.33 REL 0.33
المتظاهرين	N 1.0
وتفكيك	N 1.0
الخيم	N 1.0
التي	N 0.5 REL 0.5
نصبها	V 0.33 P 0.33 N 0.33
المعتصمون	N 1.0
خلال	P 1.0
الاسابيع	N 1.0
الماضية	ADJ 1.0

Table 4.9. Output of the appended tags file with uniform probability.

Word	Tag
انضم	V 1.0
الالاف	N 1.0
الى	V 0.5 P 0.5
المعتصمين	N 1.0
في	P 0.67 N 0.33
ميدان	N 1.0
التحرير	N 1.0
بعد	P 0.33 N 0.33 T 0.17
محاولة	N 1.0
الجيش	N 1.0
اخلا	V 1.0
الساحة	N 1.0
من	P 0.33 INTG 0.33 REL 0.33
المتظاهرين	N 1.0
وتفكيك	N 1.0
الخيم	N 1.0
التي	N 0.75 REL 0.25
نصبها	V 0.62 P 0.08 N 0.31
المعتصمون	N 1.0
خلال	P 1.0
الاسابيع	N 1.0
الماضية	ADJ 1.0

Table 4.10. Output of the appended tags file with freq. probability.

Chapter 5

Conclusions

Through the project I have discovered and confirmed that Arabic NLP is very undeveloped, compared to English and some other languages. The reasons for this cannot be affirmed but some factors that have been mentioned in the introductory chapters is the non-Roman script and also perhaps that Arabic is a morphologically complex language. The challenges that lie ahead in the area are many and this should be a motivating factor for NLP researchers over the world.

5.1 The experiments

From the results presented in the previous chapter we see that the results from using a training corpus with different characteristics than the text to tag yielded expected results (very low accuracy). Obviously the results on QA were also expected (high accuracy), only a small difference between the Stanford tagger and the TreeTagger. It is interesting to see that both the Stanford and TreeTagger were worse on MSA than the baseline results which was basically tagging all words as a noun. We see that changing parameters and settings for the appended tags does actually lead to a slight improvement, see Table 4.7, which was the experiment with the highest accuracy and also the highest values on the tags' error measures. The other experiment scoring the highest was the one from Table 4.5. In that experiment, no probability is associated with the tags but the accuracy is still the same as when choosing the frequency probability, i. e. the results from Table 4.7. We see there is only a slight difference in the error measures between the two. Finally the error measures are not presented for the initial experiments and the Stanford tagger because they only consisted of two tags and it is the accuracy which is of interest for those experiments. In general an accuracy of 70% is probably not good enough for many applications.

As a side note, the IMPN tag (Imperative verbal noun in the Quran Corpus tag set) was never useful because the MA did not have an analysis output resembling that tag. Furthermore the LOC tag was not a part of the tagging because as explained BAMA could not distinguish between time and location adverb and all adverbs were collapsed to only time adverb.

It can be argued that I should have chosen more representative texts for tagging, or at least a text with more words. However as stated earlier, finding tagged texts to be considered as gold standard is a rare resource in Arabic NLP, having to tag something myself with the help of others would have also been a time-consuming task not suitable for this project. With the sample text I could still conduct some experiments and see that the performance does improve with an MA, all sufficient for this project.

A high account is due to the morphological analysis, we can see in the Table 4.3 that the MA itself reaches a 69% accuracy, while the usage of TreeTagger increases it to roughly 73%. By this we can draw the conclusion that the contribution from the tagger is not that much after all.

5.2 Future work

Obviously the first thing to experiment with is having a more fine-grained tag set, that would involve some more sophisticated methods on choosing the best solution from the MA, one way is to assign some sort of score to a solution that aids in the decision. This would open up for example building tools to adjust tagging granularity, depending on end application.

More tagged corpora are needed. My idea is to build on the work of (Sawalha and Atwell, 2010) and try to develop a corpus tagged with that new tag set.

Finally, a lot of resources are presented in (Nizar Habash, 2010) which is quite useful at first sight, but many of those tools are licensed and/or not available publicly, as earlier stated. This is a real impediment for those that wish to get to take further steps in the area. Having only text books that describe the Arabic language and its morphology is not likely going to attract new researchers to the area. Furthermore, the lack of easily accessing tools, is a setback. I believe it is very important to have those at hand, should we wish to see more and better results in the area. In the end it's perhaps only a matter of time and we will see more and better applications being built.

Bibliography

I. Al-Sughaiyer and A. Al-Kharashi. Arabic morphological analysis techniques: A comprehensive survey. *Journal of the American Society for Information Science and Technology*, 55:189-213, 2004.

S. Algrainy, H.M. AlSerhan, and A. Ayesh. Pattern-based algorithm for part-of-speech tagging. In *International Conference on Computer Engineering and Systems, ICCES*, pages 119-124, 2008.

M. Altabba, A Al-Zaraee, and M A Shukairy. An Arabic morphological analyzer and part-of-speech tagger. Master's thesis, Arab International University, Damascus, Syria, 2010.

M.A. Attia. Arabic tokenization system. In *Semitic '07 Proceedings of the 2007 Workshop on Computational Approaches to Semitic Languages: Common Issues and Resources*, 2007.

E. Atwell, L. Al-Sulaiti, S. Al-Osaimi, and B. Abu Shawar. A review of Arabic corpus analysis tools. In *Bel, B and Marlien, I (editors) Proceedings of TALN04: XI Conference sur le Traitement Automatique des Langues Naturelles*, volume 2, pages 229-234, 2004.

K.R. Beesley. Finite-state morphological analysis and generation of Arabic at Xerox research: Status and plans, 2001.

A. Berger, S. Della Pietra, and V.J. Della Pietra. A maximum entropy approach to natural language processing. *Computational Lingustics*, 22(1):39-71, 1996.

Ann Bies. http://www.ircs.upenn.edu/arabic/Jan03release/arabic-POStags-collapse-to-PennPOStags.txt.

S. Bird and E. Loper. NLTK: The natural language toolkit. In *Proceedings of the ACL demonstration session*, pages 214-217. Association for Computational Linguistics, July 2004.

A. Boudlal, A. Lakhouaja, A. Mazroui, A. Meziane, and M. Bebah. Alkhalil morpho sys: A morphosyntactic analysis system for Arabic texts. azze.mazroui@gmail.com.

E. Brill. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Lingustics*, 21(4):543-566, 1995.

T. Buckwalter. *Buckwalter Arabic Morphological Analyzer 1.0*. Linguistic Data Consortium, 2002.

Stanford CoreNLP. http://nlp.stanford.edu/software/corenlp.shtml.

K. Dukes. The Quranic Arabic Corpus. http://corpus.quran.com/.

M. Forsberg and A. Ranta. Functional morphology. In *Proceedings of the ninth ACM SIGPLAN international conference on Functional programming*. ACL, 2004.

GATE. http://gate.ac.uk/.

N. Habash and O. Rambow. Arabic tokenization, part-of-speech tagging and morphological disambiguation in one fell swoop. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*. ACL, 2005.

J. Hajič. Morphological tagging: data vs dictionaries. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, 2000.

D. Jurafsky and J. Martin H. *Speech and Language Processing*. Pearson Education, 2009.

S. Khoja. Apt: Arabic part-of-speech tagger. In *Proceedings of the Student Workshop at the Second Meeting of the North American Chapter of the Association for Computational Linguistics*, 2001.

S. Khoja, R. Garside, and G. Knowles. A tagset for the morphosyntactic tagging of Arabic. *Corpus Linguistics*, 2001.

M. Paul (ed.) Lewis. *Ethnologue: Languages of the World*. SIL International, Dallas, Texas, sixth, edition, 2009. Online version: http://www.ethnologue.com/.

Y. Nizar Habash. *Introduction to Arabic natural language processing*. Morgan and Claypool, 2010.

M. Sawalha and E. Atwell. Fine-grain moprhological analyzer and part-of-speech tagger for Arabic text. In *Language Resources and Evaluation Conference*, 2010.

H. Schmid. Probabilistic part-of-speech tagging using decisions trees. In *International Conference on New Methods in Language Processing*, 1994.

E. Schulz, G. Krahl, and W. Reuschel. *Standard Arabic. An elementary intermediate course*. Cambridge University Press, 2008.

O. Smrž. *Functional Arabic Morphology - Formal System and Implementation*. PhD thesis, Charles University, Prague, 2007.

K. Toutanova and C D. Manning. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-2000)*, pages 63-70, 2000.

K. Toutanova, D. Klein, C. Manning, and Y. Singer. Feature-rich part-of-speech tagging with cyclic dependency network. In *Proceedings of HLT-NAACL*, pages 252-259, 2003.

J.H. Yousif and T. Sembok. Arabic part-of-speech tagger based support vector machines. In *International Symposium on Information Technology*, 2008.

Appendices

Appendix A

The Arabic alphabet

Name of the letter	Phonemic value	Transliteration	Initial Position	Medial Position	Final position	Isolated Position
Alif	a	ā	ا	ـا	ـا	ا
Bā	*b*	b	بـ	ـبـ	ـب	ب
Tā	*t*	t	تـ	ـتـ	ـت	ت
Thā	*θ*	th	ثـ	ـثـ	ـث	ث
Jīm	ʒ	ǧ	جـ	ـجـ	ـج	ج
Ḥā	ħ	ḥ	حـ	ـحـ	ـح	ح
Khā	*x*	kh	خـ	ـخـ	ـخ	خ
Dāl	*d*	d	د	-	ـد	د
Dhāl	ð	*dh*	ذ	-	ـذ	ذ
Rā	*r*	r	ر	-	ـر	ر
Zāy	*z*	z	ز	-	ـز	ز
Sīn	*s*	s	سـ	ـسـ	ـس	س

Continued on next page

Shīn	ʃ	š	شـ	ـشـ	ـش	ش
Ṣād	sˤ	ṣ	صـ	ـصـ	ـص	ص
Ḍad	dˤ	ḍ	ضـ	ـضـ	ـض	ض
Ṭā	tˤ	ṭ	طـ	ـطـ	ـط	ط
Ẓa	zˤ	ẓ	ظـ	ـظـ	ـظ	ظ
ᶜAyn	ʕ	ᶜ	عـ	ـعـ	ـع	ع
Ghayn	ɣ	ġ	غـ	ـغـ	ـغ	غ
Fā	*f*	f	فـ	ـفـ	ـف	ف
Qāf	*q*	q	قـ	ـقـ	ـق	ق
Kāf	*k*	k	كـ	ـكـ	ـك	ك
Lām	*l*	l	لـ	ـلـ	ـل	ل
Mīm	*m*	m	مـ	ـمـ	ـم	م
Nūn	*n*	n	نـ	ـنـ	ـن	ن
Hā	*h*	h	هـ	ـهـ	ـه	ه
Wāw	*w*	w,ū	و	-	و	و
Yā	*j*	y	يـ	ـيـ	ـي ـى	ي ى
Not part	of the	alphabet the				
Hamza	ʔ	glottal stop	ء	ئ	ؤ	إ أ
Ta-marbuta	lit. connected Ta	usually denotes feminine gender	expands to تـ when	followed by connecting suffix.	ة	
Tatweel	used for typography	e.g. بـــ				

Appendix B

Overview of Arabic

In this chapter I give an overview of Arabic, and some of the important concepts in Arabic grammar that are related to this project. This chapter is based on (Schulz et al., 2008; Nizar Habash, 2010). It should be clear that the review in this chapter is not comprehensive. A complete explanation of only Arabic morphology can cover an entire book and is definitely not within the scope of this report.

B.1 The Arabic language

Arabic is a semitic language, closely related to Hebrew and is spoken world wide. It is estimated that around 220 million people are native Arabic speakers(Lewis, 2009). Not everyone speak the same *type* of Arabic though. The contrast between colloquial Arabic and its standard forms is sharp. The latter characterised by a conservative and very strict way of communication, while the former differs geographically, all the way from Morocco in the west to Iraq in the east. It is not unusual that an Arabic speaker from Morocco has difficulties understanding someone from Iraq, or vice versa.

B.1.1 Different types of Arabic

Arabic consists of a wide range of dialects and its two standard forms. The latter is divided into the classical Arabic (CA) and modern standard Arabic (MSA). The classical Arabic is not widely spoken today, it is the same language which the Quran is written in; therefore it is also referred to as Quranic Arabic (QA). Modern standard Arabic is derived from CA/QA and contains many new and *modern* words, e.g. telephone and computer. Another difference is that CA/QA also has some complex grammatical constructs, which do not exist in MSA. However, these constructs are few in relation to the amount of words between the two standard types. MSA is the standard form taught and spoken in schools and universities. It is also the language of media broadcasts and modern literature.

B.2 Alphabet, Words and Vowels

B.2.1 Alphabet

Arabic has 28 letters, many which are phonetically similar to the Latin alphabet. Each letter takes different forms depending on its position in a word. A letter may be in one of the following positions: initial, medial, final or isolated. Many letters have the same initial and isolated form. The alphabet is described in Appendix A, it is recommended to look at it before proceeding to the next sections.

B.2.2 Solar letters

Some letters are called solar letters by tradition, these are ت ث د ذ ر ز س ش ص ض ط ظ ن

Vowels There are three vowels in Arabic: a (Alif), o (Waw), i (Ya). Each vowel is either a short or long. The long vowel is indicated by its corresponding letter in a word a (Alif -), o (Waw -), i (Ya -), while the short vowels are indicated by diacritics, see B.2.4.

B.2.3 Words

Arabic words are traditionally divided into three categories: nouns, verbs and particles. The nouns are words which describe a person, thing or an idea. They may be in derived form, i.e. derived from either verbs, other nouns or particles. Or they are primitive nouns, i.e. not in a derived form. The particles consist of prepositions, adverbs, conjunctions, interrogative word, exceptions(non-,un-,in-) and interjections. There exists three verb tenses: perfect tense, imperfect (not to be confused with swedish imperfe*k*t) and imperative (Khoja, 2001).

To all of the above categories: number, case and gender apply.

B.2.4 Diacritics

A second class of symbols are the diacritics, there are three types of diacritics in Arabic: vowel, nunation and shadda. Often these are omitted during analysis. The diacritics for Arabic are illustrated in the Figure B.1 below and their positions. They indicate the presence of a short vowel, the right column shows the corresponding *nunation* - meaning the short vowel (a,o,i) ends in in "n" when pronounced, this can only appear on the final letter of a word.

For example the word ktb when followed by the vowel/diacritic "a" is pronounced kataba - becomes (they 3.pl.m wrote)

Or with the nunation - becomes kataban (they wrote accusative case, more on cases in B.3). The shadda denotes an emphasis on a letter and looks like .

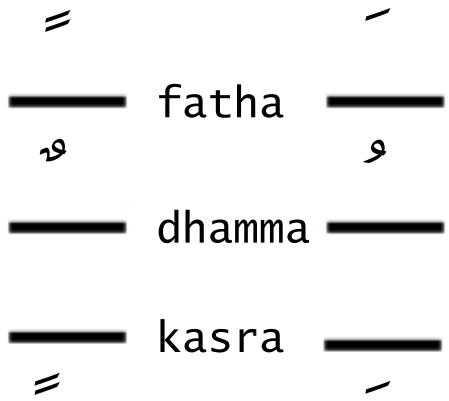

Figure B.1. Arabic short vowel diacritics

B.2.5 Clitics

A clitic is an unstressed part of a word that occurs only in combination with another word. E.g. 'll in you'll. Furthermore they are categorized into proclitics (clitics preceding a word, y in y'all) and enclitics (clitics succeding a word). For morphology it's important to understand the function of the clitics.

B.3 Case, Mood, Gender and Numbers

Gender and Numbers are simple. A noun can either be feminine or masculine. Most words ending in *ta-marbuta* ( ) are feminine. Word can either be singular, dual or plural.

Case in Arabic comes in nominative, accusative and genitive case. They are indicated by diacritics on the final letter of a word. In Figure B.1 the fatha denotes accusative, the dhamma nominative and the kasra genitive.

Mood Additionally, a verb may have the following moods: jussive, subjunctive and indicative

B.4 Morphology

Morphology is the study of the way words are built from morphemes.

Morpheme A meaningful unit in a word, usually the smallest.

Generally a morpheme is either a stem or an affix. The *stem* is often the "main" morpheme of a word (Jurafsky and Martin H, 2009). Affixes are attached to the stem, they are categorized as: prefix, preceds the stem; infix, inserted in a stem; suffix, succeds the stem; circumfix, precede and succeds the stem.

In the word *eats*, eat is the stem - which also is the main morpheme - and *s* is a suffix morpheme.

As mentioned before, understanding clitics is important in Arabic morphology, as well as affixes. Deciding whether a morpheme is an affix or a clitic is important in morphological analyzers. For example in the word "and they will write it" - wasayaktabunhA - illustrated in Table B.1. The word consists of two proclitics, one circumfix and an enclitic as illustrated in the table below (Nizar Habash, 2010).

wasayaktabunhA					
wa+	sa+	y+	aktub+	uwna+	hA
and	will	3person	write	M.plural	it
proclitic	proclitic	circum1	stem	circum2	enclitic

Table B.1. Morphological analysis of wasayaktabunhA

Generally one can say that clitics carry a syntactic function and not part of the word (e.g. negation, definiteness), while affixes a morpho-syntactic function (e.g. mood, tense, gender or number) (Attia, 2007).

Form-based morphology or morpheme-based morphology is the study of how a word is built up from its morphemes. A subcategory is *concatenative morphology* where the stem is a central part, words in this category are characterized by attaching a suffix to the stem.

Functional morphology in contrast is the study of how each morpheme affect the semantic and syntactic meaning of a word.

Appendix C

Buckwalter transliteration

The Buckwalter transliteration table is the most widely used for the Arabic language. It encodes each Arabic letter and diacritics with 7-bit ASCII. This was useful in early systems, and still in systems that cannot handle UTF-8 natively.

'	Hamza
|	Alif with madda
>	Alif with hamza above
&	Waw with hamza above
<	Alif with hamza below
}	Ya with hamza above
A	Alif
b	Ba
p	Ta-marbuta
t	Ta
v	Tha
j	Jim
H	Ḥā
x	Khā
d	Dāl
*	Ṭā

r	Rā
z	Zā
s	S̄in
$	Sh̄in
S	Ṣād
D	Ḍad
T	Ṭā
Z	Ẓā
E	ᶜAyn
g	Ghayn
_	Tatweel
f	Fā
q	Qāf
k	Kāf
l	Lām
m	M̄im
n	N̄ūn
h	Hā
w	Wāw
Y	Alef maksura
y	Yā
F	ً
N	ٌ
K	ٍ
a	َ
u	ُ
i	ِ
~	ّ
o	ْ
`	ٰ
{	ٱ

TRITA-CSC-E 2011:133
ISRN-KTH/CSC/E--11/133-SE
ISSN-1653-5715

www.kth.se