STRUCTURAL ANALYSIS
OF ARABIC TWEETS

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
IN THE FACULTY OF SCIENCE AND ENGINEERING

2017

By
Fahad Albogamy
School of Computer Science
Contents

List of Figures  5
List of Tables  7
Abstract  9
Declaration  10
Copyright  11
Dedication  12
Acknowledgements  13
Publications based on the thesis  14
Arabic transliteration  15
List of abbreviations and acronyms  16

1 Introduction  17
  1.1 Overview of the challenges of Arabic tweets processing  20
  1.2 Research goals  21
  1.3 Contributions  22
  1.4 Thesis structure  23

2 Structural Analysis of Arabic: A Review  25
  2.1 Introduction  25
  2.2 Ambiguity in natural languages  26
    2.2.1 Lexical ambiguity  26
    2.2.2 Structural ambiguity  28
    2.2.3 Scope ambiguity  30
2.3 Ambiguity in Arabic

2.3.1 Missing diacritics

2.3.2 Word order variation

2.3.3 Cliticisation

2.3.4 Pro-dropping

2.3.5 Noun multi-functionality

2.4 Properties of English and Arabic tweets

2.5 Summary

3 State-of-the-art Approaches to Arabic NLP: A Review

3.1 Introduction

3.2 A Typical NLP Pipeline

3.3 POS tagging

3.3.1 POS tagging approaches

3.3.2 POS tagsets

3.3.3 POS taggers

3.4 Stemming

3.5 Syntactic parsing

3.5.1 Parsing frameworks

3.5.2 Parsing algorithms

3.5.3 Parsing strategies
CONTENTS

3.5.4 Parsers for Arabic ........................................... 77
  3.5.4.1 Parasite .................................................. 77
  3.5.4.2 MALTParser .............................................. 79
3.6 Summary ....................................................... 80

4 Analysis of Existing POS Tagging on Arabic Tweets 81
  4.1 Introduction .................................................. 81
  4.2 Experiments ................................................... 82
    4.2.1 Data collection ........................................... 83
    4.2.2 Annotation .............................................. 83
    4.2.3 Tagset unification ....................................... 84
    4.2.4 Inter-annotator agreement ................................ 86
    4.2.5 POS tagging comparison .................................. 87
    4.2.6 Error analysis ......................................... 88
  4.3 Summary ....................................................... 94

5 POS Tagging for Arabic Tweets 95
  5.1 Introduction .................................................. 95
  5.2 Pre- and post-processing .................................... 97
    5.2.1 Pre-processing ......................................... 98
    5.2.2 Post-processing ........................................ 102
    5.2.3 Handling Twitter-specific items, emoticons and emoji 103
    5.2.4 Results for pre- and post-processing .................. 104
  5.3 Bootstrapping ............................................... 106
    5.3.1 Agreement-based bootstrapping results ................ 106
  5.4 Summary ....................................................... 108

6 Stemming Arabic Tweets 110
  6.1 Introduction .................................................. 110
  6.2 Arabic word forms .......................................... 113
  6.3 Experimental setup .......................................... 114
  6.4 Heavy stemmer .............................................. 115
  6.5 Light stemmer ............................................... 118
  6.6 Summary ....................................................... 125

7 Syntactic Parsing for Arabic Tweets 126
  7.1 Introduction .................................................. 126
  7.2 Choice of parsing strategy .................................. 129
  7.3 Choice of syntactic representation ......................... 130
  7.4 Rule-based parsing .......................................... 131
  7.5 Data-driven parsing ......................................... 137
  7.6 Treebanking and parsing experiments ....................... 138
    7.6.1 Implementation ......................................... 138
CONTENTS

7.6.2 Experimental setup ................................................. 139
7.6.3 Using a rule-based parser to build a dependency treebank .... 140
7.6.4 Bootstrapping treebank development .............................. 141
7.7 Summary ................................................................. 146

8 Conclusion and Future Work ............................................. 148
  8.1 Thesis summary .......................................................... 148
  8.2 Main contributions ....................................................... 151
  8.3 Future directions ......................................................... 152

References ................................................................. 155

Appendix A An example of Buckwalter Arabic morphological analyser output ........................................... 166
Appendix B The Penn Arabic Treebank Tagset ......................... 167
Appendix C MADA tagsets .................................................... 168
Appendix D Twitter Stream API (Java implementation) ............... 169
Appendix E Pre- and Post-processing code .............................. 172
Appendix F Stemmer source code .......................................... 182
Appendix G Treebank development source code ......................... 184
Appendix H Examples of Parasite* annotation ........................... 189

Word count: 42,913
List of Figures

3.1 A typical NLP pipeline. ...................................................... 49
3.2 The NLP pipeline for Figure 3.1 adapted for tweets. ................. 51
3.3 The NLP pipeline proposed for Arabic tweets. ......................... 53
3.4 POS tagging approaches. .................................................. 57
3.5 Example of a context-free grammar. .................................... 71
3.6 Phrase structure tree for the English sentence in (3.2). .............. 71
3.7 Dependency tree for the English sentence (3.2). SBJ= Subject, OBJ= Object, ATT= Attribute, PC= Prepositional clausal modifier and PU= Punctuation .......................... 73

5.1 The NLP pipeline adapted for Arabic tweets (tagging). White boxes are discussed in this Chapter. ...................................................... 96
5.2 Agreement-based bootstrapping. ......................................... 107

6.1 The NLP pipeline adapted for Arabic tweets (stemming). White boxes are discussed in this Chapter. ...................................................... 112
6.2 Possible sub-tokens in Arabic nouns. .................................. 113
6.3 Possible sub-tokens in Arabic verbs. .................................. 114
6.4 Arabic grammar-lexis. ....................................................... 114
6.5 Heavy stemmer workflow. .................................................. 116
6.6 Possible stems for the word ‘wbalyd’ and frequencies. .............. 117
6.7 Preliminary regular expressions for nouns [1] ............................ 118
6.8 Regular expression for components of a noun. ....................... 120
6.9 Basic pattern for verbs. ...................................................... 120
6.10 Present tense and person markers with agreement constraints. .... 121
6.11 Light stemmer workflow. .................................................. 122

7.1 The NLP pipeline adapted for Arabic tweets (parsing). White boxes are discussed in this Chapter. ...................................................... 128
7.2 Length of tweets (words). .................................................... 127
7.3 Rule-based parser segments a tweet to three fragments. ............. 133
7.4 Dependency tree for tweet (7.2). Left: (English) Right: (Arabic). .... 135
7.5 Dependency tree for tweet (7.3). Left: (English) Right: (Arabic). .... 135
7.6 Dependency tree for tweet (7.4). Left: (English) Right: (Arabic). .... 136
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.7</td>
<td>Dependency tree for tweet (7.5). Left: (English) Right: (Arabic)</td>
<td>136</td>
</tr>
<tr>
<td>7.8</td>
<td>Dependency tree for tweet (7.6). Left: (English) Right: (Arabic)</td>
<td>137</td>
</tr>
<tr>
<td>7.9</td>
<td>Python pseudo-code for calling Parasite* and MALTParser*</td>
<td>139</td>
</tr>
<tr>
<td>7.10</td>
<td>Using a rule-based parser to build a dependency treebank</td>
<td>141</td>
</tr>
<tr>
<td>7.11</td>
<td>Bootstrapping approach</td>
<td>142</td>
</tr>
<tr>
<td>7.12</td>
<td>Bootstrapping a treebank by using rule-driven parser and data-driven parser</td>
<td>143</td>
</tr>
<tr>
<td>7.13</td>
<td>Using rule-based parser as a filter</td>
<td>144</td>
</tr>
<tr>
<td>7.14</td>
<td>Parsing Accuracy</td>
<td>146</td>
</tr>
</tbody>
</table>
List of Tables

3.1 The active inflection forms for the regular sound verb دراسة / درس “he studied”. .................................. 55
3.2 The derivative words for word درس "lesson". .......................................................... 56
3.3 Comparison of Arabic POS tagsets. .......................................................... 60
3.4 AMIRA2.1 output for the Arabic sentence in (3.1). ........................................... 63
3.5 MADA3.2 output for the Arabic sentence in (3.1). ........................................... 65
3.6 Stanford3.5 output for the Arabic sentence in (3.1). ........................................... 66
3.7 POS tags by three taggers for the Arabic sentence in (3.1). ................................. 66
3.8 Comparison between AMIRA, MADA and the Stanford tagger. ............................ 67
3.9 Examples of parsers. .......................................................... 77
4.1 The tagset used to annotate Twitter phenomena ................................................ 84
4.2 Collapsed POS tagset. .......................................................... 86
4.3 POS tagging performance comparison .......................................................... 88
4.4 POS tagging accuracy on Arabic Tweets for baseline taggers categorised by tokens .......................................................... 89
4.5 Errors percentage of each mistagged class and its accuracy .................................. 90
4.6 Example of Twitter element tokenised and tagged by taggers. ............................ 92
5.1 The impact on accuracy on concatenated words before and after dealing with concatenation as a pre-processing step .......................................................... 99
5.2 Examples of elongated words and their surface forms ........................................ 99
5.3 The impact on accuracy on elongated words before and after dealing with elongated words as a pre-processing step .......................................................... 100
5.4 Character deletion and surface forms .......................................................... 100
5.5 The impact on accuracy on character deletion before and after dealing with character deletion as a pre-processing step .......................................................... 100
5.6 Slang words and their IV equivalents .......................................................... 101
5.7 The impact on accuracy on slang before and after dealing with slang as a pre-processing step .......................................................... 101
5.8 The impact on accuracy on named entities before and after dealing with named entities as a post-processing step .......................................................... 102
### LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.9</td>
<td>The impact on accuracy on English words before and after dealing with English words as a post-processing step.</td>
</tr>
<tr>
<td>5.10</td>
<td>Example of handling Twitter-specific items (Tag hashtag’s words by using Pre- and post-processing).</td>
</tr>
<tr>
<td>5.11</td>
<td>The impact on accuracy on Twitter-specific items before and after dealing with Twitter-specific items.</td>
</tr>
<tr>
<td>5.12</td>
<td>The impact after applying pre- and post-processing on MSA words accuracy.</td>
</tr>
<tr>
<td>5.13</td>
<td>Impact of applying pre- and post-processing on POS tagging accuracy.</td>
</tr>
<tr>
<td>5.14</td>
<td>Accuracy and speed comparison for the taggers.</td>
</tr>
<tr>
<td>5.15</td>
<td>The Stanford tagger performance on Arabic Tweets using different training data.</td>
</tr>
<tr>
<td>6.1</td>
<td>Example of Arabic Affixes.</td>
</tr>
<tr>
<td>6.2</td>
<td>Heavy stemming results.</td>
</tr>
<tr>
<td>6.3</td>
<td>Arabic affixes.</td>
</tr>
<tr>
<td>6.4</td>
<td>Light stemming accuracy if stem length is 3 characters and more.</td>
</tr>
<tr>
<td>6.5</td>
<td>Light stemming accuracy if stem length is 2 characters and more.</td>
</tr>
<tr>
<td>6.6</td>
<td>Overall light stemming accuracy (nouns and verbs).</td>
</tr>
<tr>
<td>6.7</td>
<td>Examples of stemming errors.</td>
</tr>
<tr>
<td>7.1</td>
<td>Constructing treebank ((Treebank^{RB_{162k}})) by using Parasite*.</td>
</tr>
<tr>
<td>7.2</td>
<td>Constructing a treebank ((Treebank^{RB_{957 bootstrapped}})) by bootstrapping.</td>
</tr>
</tbody>
</table>
This thesis explores the task of analysing the linguistic structure of Arabic tweets. Arabic tweets raise many challenges that make Natural Language Processing (NLP) tasks difficult. We are faced with the same linguistic issues that any ordinary language has as well as more genre-specific problems. Tweets are difficult to manipulate because they do not always maintain formal grammar and correct spelling, and abbreviations are often used to overcome length restrictions. Arabic tweets also exhibit linguistic phenomena such as usage of different dialects, Romanised Arabic and borrowing of foreign words. All these characteristics of the microblogging genre make NLP tasks on Twitter very different from their counterparts in more formal texts. Within most NLP systems there are several early stages such as tagging, stemming and parsing that may need to be redesigned to take into account characteristics of tweets in order to be able to extract their important linguistic features. To fulfil this need, three of the most fundamental parts of the linguistic pipeline, namely POS tagging, stemming and parsing have been revisited for Arabic tweets. To the best of our knowledge, this is the first attempt to carry out this task for Arabic tweets.

We investigate the challenges of processing Arabic tweets, studying a number of standard Arabic processing tools and highlighting their limitations when manipulating Arabic tweets. We make three state-of-the-art POS taggers for Modern Standard Arabic (MSA) robust towards noise when applied to the Arabic tweets. We develop the first fast and robust POS tagger for Arabic tweets and create the first POS-tagged corpus of Arabic tweets. Also, we develop two approaches to stemming Arabic tweet words: a heavy stemmer and a light stemmer, and we find that the light stemmer provides the most suitable approach for stemming Arabic tweets words because it does not use dictionaries, is fast, and yields greater accuracy compared with the heavy stemmer and MSA stemmers. We are able to automatically create the first dependency treebank from unlabelled tweets by using two approaches: using a rule-based parser only and using a rule-based parser and a data-driven parser in a bootstrapping technique. Then, we train a data-driven parsing base model on them to parse Arabic tweets.

The findings are encouraging. We are able to improve POS tagging accuracy from 49% to 74.0% on Arabic tweets. Experimental results show that the light stemmer achieves 77.9% accuracy. It outperforms three well-known stemmers for Arabic. Our parser reaches 71.0% accuracy which is better than the performance of French parsing for social media data and it is not far behind English parsing for tweets.
Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.
Copyright

i. The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the “Copyright”) and he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii. Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made only in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.

iii. The ownership of certain Copyright, patents, designs, trade marks and other intellectual property (the “Intellectual Property”) and any reproductions of copyright works in the thesis, for example graphs and tables (“Reproductions”), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv. Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see [http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=487](http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=487)), in any relevant Thesis restriction declarations deposited in the University Library, The University Library’s regulations (see [http://www.manchester.ac.uk/library/aboutus/regulations](http://www.manchester.ac.uk/library/aboutus/regulations)) and in The University’s policy on presentation of Theses.
Dedication

This thesis is dedicated to my mother and father, for their endless love, patience and support without whom none of my success would be possible.
Acknowledgements

After thanking The God for all endless blessings and giving me such strength to achieve my dream and pass all the difficulties encountered during my PhD study, I would also like to express my gratitude to a number of people and institutions who provided me with guidance, support, influence and funding throughout this journey.

First of all, I owe an enormous amount of gratitude to my supervisor, Professor Allan Ramsay, for his efforts, support and dedication during my PhD study. His insights, ideas and guidance helped me to get over many hurdles during my PhD and put me on the right track when I was about to go off the rails. It has been a great pleasure working with him.

I would also like to extend my thanks to those who donated their precious time and expertise to discuss various parts of my thesis with them, especially Prof. Joakim Nivre, Dr. Nizar Habash, Dr. Sardar Jaf and Dr. Ali Almiman. Also, I wish to thank the academic support staff of the school of computer science at the University of Manchester for providing an academic-friendly environment and academic support.

I would like to express my appreciation to my sponsor, King Saud University, for their financial support for my PhD study. I hope to be able to transfer the skills and knowledge I acquired during my study in the UK to Saudi institutions.

Last but not least, there are no words powerful enough in expressing my thanks to my beloved wife who has given me endless love and support. Thank you for understanding my self-imposed isolation and being patient with me. Thank you to my three lovely children, Alfaisal, Manaaf and Adhub for putting up with me when I was busy doing my thesis instead of playing with you.

Despite all the hard work that was required throughout my PhD, I have enjoyed my time as a PhD student and as a volunteer in various organisations. I founded the Saudi Society in the University of Manchester and I am the president of Saudi Students’ Club in Greater Manchester.
Publications based on the thesis

The substantial ideas mentioned in this thesis have been peer-reviewed and published in the following conferences in chronological order:


The transcription of Arabic examples in this thesis follows the Buckwalter (BW) Arabic transliteration scheme.

---

<table>
<thead>
<tr>
<th>Letter</th>
<th>BW</th>
<th>Unicode name</th>
<th>Letter</th>
<th>BW</th>
<th>Unicode name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ء</td>
<td>'</td>
<td>Hamza</td>
<td>Z</td>
<td></td>
<td>Zah</td>
</tr>
<tr>
<td>ـ</td>
<td>Alef-Madda above</td>
<td>ع</td>
<td>E</td>
<td>Ain</td>
<td></td>
</tr>
<tr>
<td>ً</td>
<td>Alef-Hamza above</td>
<td>غ</td>
<td>g</td>
<td>Ghain</td>
<td></td>
</tr>
<tr>
<td>ﯽ</td>
<td>Waw-Hamza above</td>
<td>ـ</td>
<td>Tatweel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ِ</td>
<td>Alef-Hamza below</td>
<td>ﯼ</td>
<td>f</td>
<td>Feh</td>
<td></td>
</tr>
<tr>
<td>َ</td>
<td>Yeh-Hamza above</td>
<td>َٰ</td>
<td>q</td>
<td>Qaf</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>Alef</td>
<td>ﯽ</td>
<td>k</td>
<td>Kaf</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>Beh</td>
<td>l</td>
<td>Lam</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ء</td>
<td>Teh Marbuta</td>
<td>m</td>
<td>Meem</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t</td>
<td>Teh</td>
<td>n</td>
<td>Noon</td>
<td></td>
<td></td>
</tr>
<tr>
<td>v</td>
<td>Theh</td>
<td>h</td>
<td>Heh</td>
<td></td>
<td></td>
</tr>
<tr>
<td>j</td>
<td>Jeem</td>
<td>w</td>
<td>Waw</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ح</td>
<td>Hah</td>
<td>ي</td>
<td>Alef Maksura</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>Khah</td>
<td>y</td>
<td>Yeh</td>
<td></td>
<td></td>
</tr>
<tr>
<td>د</td>
<td>Dal</td>
<td>Arabic diacritics</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ذ</td>
<td>Thal</td>
<td>a</td>
<td>Fatha</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ر</td>
<td>Reh</td>
<td>u</td>
<td>Damma</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ز</td>
<td>Zain</td>
<td>i</td>
<td>Kasra</td>
<td></td>
<td></td>
</tr>
<tr>
<td>س</td>
<td>Seen</td>
<td>F</td>
<td>Fathatan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ش</td>
<td>Sheen</td>
<td>N</td>
<td>Dammatan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ص</td>
<td>Sad</td>
<td>K</td>
<td>Kasratan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ض</td>
<td>Dad</td>
<td>~</td>
<td>Shadda</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ط</td>
<td>Tah</td>
<td>o</td>
<td>Sukun</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# List of abbreviations and acronyms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full form</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>First person</td>
</tr>
<tr>
<td>2nd</td>
<td>Second person</td>
</tr>
<tr>
<td>3rd</td>
<td>Third person</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ATDT</td>
<td>Arabic Tweets Dependency Treebank</td>
</tr>
<tr>
<td>BAMA</td>
<td>Buckwalter Arabic Morphological Analyser</td>
</tr>
<tr>
<td>BPC</td>
<td>Base Phrase Chunker</td>
</tr>
<tr>
<td>DG</td>
<td>Dependency Grammar</td>
</tr>
<tr>
<td>EAGLES</td>
<td>Expert Advisory Group on Language Engineering Standards</td>
</tr>
<tr>
<td>ERTS</td>
<td>Extended Reduced Tagset</td>
</tr>
<tr>
<td>HMM</td>
<td>Hidden Markov Model</td>
</tr>
<tr>
<td>IR</td>
<td>Information Extraction</td>
</tr>
<tr>
<td>HPT</td>
<td>Head Percolation Table</td>
</tr>
<tr>
<td>LDC</td>
<td>Linguistic Data Consortium</td>
</tr>
<tr>
<td>MADA</td>
<td>Morphological Analysis and Disambiguation for Arabic</td>
</tr>
<tr>
<td>MALTParser</td>
<td>Models and Algorithms for Language Technology Parser</td>
</tr>
<tr>
<td>MSA</td>
<td>Modern Standard Arabic</td>
</tr>
<tr>
<td>MSTParser</td>
<td>Maximum-Spanning Tree Parser</td>
</tr>
<tr>
<td>MT</td>
<td>Machine Translation</td>
</tr>
<tr>
<td>NLP</td>
<td>Natural Language Processing</td>
</tr>
<tr>
<td>NP</td>
<td>Noun Phrase</td>
</tr>
<tr>
<td>OVS</td>
<td>Object-Verb-Subject</td>
</tr>
<tr>
<td>P</td>
<td>Precision</td>
</tr>
<tr>
<td>POS</td>
<td>Part-of-speech</td>
</tr>
<tr>
<td>PP</td>
<td>Prepositional Phrase</td>
</tr>
<tr>
<td>PSG</td>
<td>Phrase Structure Grammar</td>
</tr>
<tr>
<td>QA</td>
<td>Question and Answering</td>
</tr>
<tr>
<td>R</td>
<td>Recall</td>
</tr>
<tr>
<td>RTS</td>
<td>Reduced Tagset</td>
</tr>
<tr>
<td>SAMA</td>
<td>Standard Arabic Morphological Analyser</td>
</tr>
<tr>
<td>SRILM</td>
<td>Standard Research Institute Language Modelling</td>
</tr>
<tr>
<td>SVM</td>
<td>Support Vector Machine</td>
</tr>
<tr>
<td>SVO</td>
<td>Subject-Verb-Object</td>
</tr>
<tr>
<td>TOK</td>
<td>Tokeniser</td>
</tr>
<tr>
<td>VP</td>
<td>Verb Phrase</td>
</tr>
<tr>
<td>VSO</td>
<td>Verb-Subject-Object</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The last few years have seen an enormous growth in the use of social networking platforms. Because of the free format of messages and the easy accessibility of these platforms, Internet users have tended to shift from traditional communication tools such as blogs or mailing lists to microblogging platforms such as Twitter[^1]. People post about their life, share opinions on a variety of topics and discuss current issues. There are millions of tweets daily, yielding a corpus which is noisy and informal, but which can be informative. As a result, Twitter has become one of the most important social information platforms. The nature of the text content of microblogs differs from traditional blogs. In Twitter, for example, a tweet is short and contains a maximum of 140 characters. Tweets also are not always written maintaining formal grammar and proper spelling. They are ambiguous and rich in acronyms. Slang and fashionable words are often used to overcome their restricted lengths ([Java et al.](2007)). Its 4A (Anytime, Anywhere, Anyone, Anything) characteristic has brought about a great amount of real-time data and a lot of new media elements such as the hashtag, emoticon, hyperlink, etc. ([Sui et al.](2012)).

All these characteristics of the microblogging genre make Natural Language Processing

[^1]: http://www.twitter.com
(NLP) tasks on Twitter markedly distinct from their counterparts in more formal texts. Therefore, it is an open question as to how well the techniques of NLP used on more well-formed data (e.g. in the newswire genre) will transfer to Twitter in order to increase the understanding of and exploit the potential of tweets. As a result, there has been much research conducted on applying NLP techniques to tweets recently. For example, Sharifi et al. (2010), Barbosa and Feng (2010) and Balasubramanyan et al. (2010) focused on English tweets analysis. However, the syntactic structure of tweets is often neglected in this work and its techniques have usually been built on the basis of lexical information only. They do not use the standard linguistic pipeline tools such as Part-of-Speech (POS) tagging and parsing which might enable a richer linguistic analysis (Gimpel et al., 2011). One possible reason for this is that these tools are typically trained on formal texts and their performance is degraded on tweets (Foster et al., 2011). Within most NLP systems there are several early stages such as tagging, stemming and parsing that may need to be redesigned to take into account characteristics of tweets in order to be able to extract their most important linguistic features. To fulfil this need, one of the most fundamental parts of the linguistic pipeline (POS) has been revisited for English tweets. The ARK\textsuperscript{3} tagger and T-POS\textsuperscript{4} tagger are examples of this. The former reaches 92.8% accuracy (Gimpel et al., 2011) whereas the latter achieves 88.4% accuracy (Ritter et al., 2011). In parsing, Kong et al. (2014) developed a dependency parser for English tweets, but they omitted most of the tweets elements from the material to be parsed which led to losing parts of the content of the tweet.

There has been an enormous increase in the use of microblogging services and social media sites in the Arab world. A study prepared and published by Semiocast\textsuperscript{5} in 2012 has revealed that Arabic was the fastest growing language on Twitter in 2011. Therefore, much interest

\footnote{\textsuperscript{2}Experiments done on English tweets.}
\footnote{\textsuperscript{3}http://www.ark.cs.cmu.edu/TweetNLP/}
\footnote{\textsuperscript{4}http://www.gate.ac.uk/wiki/twitter-postagger.html}
\footnote{\textsuperscript{5}https://semiocast.com/en/publications/2011_11_24_Arabic_highest_growth_on_Twitter}
in Arabic tweets has been generated. Mourad and Darwish (2013), El-Fishawy et al. (2014), Albraheem and Al-Khalifa (2012) targeted Arabic tweets, but all the concentration was on low level lexical relations which were used for shallow parsing and sentiment analysis. Furthermore, the state-of-the-art NLP systems that are used for English in this genre will not work for Arabic. One reason for this is the complex morphological, structural and grammatical nature of Arabic (Habash et al., 2009). So, it is crucial to study the Arabic microblogging genre, analyse texts written by its users’ and develop NLP systems suitable for this genre.

To understand Arabic tweets and exploit them, their structure should be analysed and understood. Therefore, the key task in this research project is to analyse the linguistic structure of Arabic tweets. This task involves three linked subtasks, tagging, stemming and parsing. The new Arabic tweet POS tagger, stemmer and parser should take into consideration the characteristics of Arabic tweets and they should yield acceptable results.

In this Chapter, we provide an overview of our research into analysing the structure of Arabic tweets. Moreover, we explain the main challenges for processing Arabic tweets, and we highlight the objectives of the research. We also shed some light on our contributions made through this work. Finally, we conclude by outlining the structure of this thesis.
CHAPTER 1. INTRODUCTION

1.1 Overview of the challenges of Arabic tweets processing

Arabic tweets raise many challenges that make Natural Language Processing (NLP) tasks difficult. This genre poses a number of problems for programs that carry out lexical and structural analysis. As far as structure is concerned, there are tweets that are written perfectly formally and there are tweets that do not obey the normal rules of Arabic. The former have the same linguistic issues that any ordinary language has, the latter cause more genre-specific problems.

When processing normal Arabic, we are faced with a large amount of ambiguity. The lexical ambiguity arises from two sources: (i) classical Arabic is written with diacritics, but these are optional in the Modern Standard Arabic (MSA) writing system; and (ii) Arabic is a highly inflectional and derivational language, which means that any root can have a number of derived forms. Diacritics are often the only way to differentiate between different words (derived forms) and between inflected forms of the same word. The combination of lack of diacritics and inflectional and derivational leads to a large amount of lexical ambiguity in Arabic.

In addition, Arabic has a high syntactic flexibility (Daimi, 2001). Arabic has a relatively free word order, which is considered one of the sources of its ambiguity (Attia, 2008). Beside the canonical order of Arabic sentences which is Verb-Subject-Object (VSO), there is a range of possible non-canonical orders such as OVS, SVO and VOS (Ramsay and Mansour, 2006). The potential of allowing such non-canonical constructions leads to huge structural ambiguities in Arabic.

Furthermore, subjects can be omitted from Arabic sentences and verbs can recover missing subjects. So, Arabic is a pro-drop language (Attia, 2008). Determining whether or not there is an omitted pronoun in the subject position is a challenge which is increased by the fact that a large number of Arabic verbs can be both transitive and intransitive. It is impossible in Arabic to differentiate between active and passive forms by examining the surface form.

Arabic tweets pose another set of genre-specific problems as well as the above challenges.
Tweets contain acronyms and slang, and they do not always maintain correct spelling. They also have new elements, which do not exist in the MSA, that play different grammatical roles in this context and cannot easily be dealt with using traditional grammar, such as mentions, replies, retweets, hashtags, links, emoji and emoticons. Tweets also have a very open lexicon which leads to the presence of a large number of words not listed in any dictionary. It is not just that there is extra ambiguity. Some of tweets also have the problem that they just do not have a sensible structure at all and contain scrambled material which does not make sense.

It should be noted that it is the combination of the above issues that leads to huge problems. In processing Arabic tweets, each task on its own would be challenging, but the way they interact together compounds the difficulties.

1.2 Research goals

The aim of this study is to investigate a range of approaches to POS tagging, stemming and parsing for Arabic tweets in order to be able to analyse their structure and extract the important linguistic features from their text. The hypothesis of this research is that developing a POS tagger, stemmer and parser that take into account the properties of Arabic tweets could result in producing efficient and accurate tagging, stemming and parsing solutions for Arabic tweets compared with the state-of-the-art Arabic taggers, stemmers and parsers which have been designed for and trained on standard Arabic.

The main objectives of this research are as follows:

**RO1** To investigate the complexity of Arabic syntactic structure in general, identify the sources of ambiguity and study the properties of English and Arabic tweets in particular that may create challenges for taggers, stemmers and parsers.
RO2  To study a number of standard Arabic processing tools and show how the challenges raised by tweets impact on their effectiveness.

RO3  To investigate approaches to POS tagging Arabic tweets which will overcome the limitations of standard taggers when applied to the Arabic tweets.

RO4  To explore different approaches to stemming Arabic tweets words.

RO5  To identify different techniques for automatically creating an Arabic tweets treebank.

RO6  To develop and evaluate a set of tools based on these investigations on Arabic tweets by means of experiments and compare the results with state-of-the-art taggers, stemmers and parsers for MSA.

1.3 Contributions

The main contributions of this research are as follows:

1. We have developed a fast and robust POS tagger for Arabic tweets. We use a combination of normalisation and external knowledge to avoid the noisiness of the genre, and we generate training data to train the Stanford tagger by applying an agreement-based bootstrapping approach on heterogeneous tagger outputs (i.e. AMIRA, MADA and the Stanford tagger) (see Chapter 5 for more details).

2. We have implemented a light stemming approach which does not rely on any root dictionary, which is crucial for stemming Arabic tweets, since they have a very open lexicon (see Chapter 6 for more details).

3. We have presented our approaches for creating a dependency treebank from unlabelled Arabic tweets without any manual intervention. The data-driven parsing model we have
trained with the Arabic tweets treebank achieves an accuracy of 71% (see Chapter 7 for more details).

4. We have created the first POS-tagged corpus and dependency treebank for Arabic tweets (see Chapter 5 and Chapter 7 for more details).

1.4 Thesis structure

This thesis is organised into eight chapters, including the introductory chapter, with additional appendices.

In this Chapter, Introduction, we have introduced the research problem and the challenges of Arabic tweets processing. Then, the main goals of this research and our contributions are explained.

The remainder of the thesis is organised as follows:

Chapter 2, Structural Analysis of Arabic: A Review, describes the sources of ambiguity in natural languages with particular focus on Arabic. The chapter ends with discussion of the properties of English and Arabic tweets and Twitter phenomena.

Chapter 3, State-of-the-art Approaches to Arabic NLP: A Review, presents POS tagging approaches and introduces three state-of-the-art POS taggers for Arabic, namely AMIRA, MADA and the Stanford tagger. It also describes different POS tagsets, different stemming approaches and different frameworks, algorithms and strategies for parsing natural languages.

Chapter 4, Analysis of Existing POS Tagging on Arabic Tweets, starts with presenting our efforts for collecting the data, using AMIRA, MADA and the Stanford tagger to tag Arabic tweets and analysing the experimental results. The material in this chapter is derived from our
papers (Albogamy and Ramsay, 2015a,b).

Chapter 5, \textit{POS Tagging for Arabic Tweets}, begins by presenting our efforts for improving the accuracy of the three taggers described in Chapter 3 by using the combination of normalisation and external knowledge. Next, we use a bootstrapping approach on unlabelled Arabic tweets to create a sufficient amount to train the augmented version of the Stanford tagger on it, resulting in a very fast and robust POS tagger for Arabic tweets. The material in this chapter is derived from our paper (Albogamy and Ramsay, 2016a).

Chapter 6, \textit{Stemming Arabic Tweets}, presents the implementation of two different techniques to stemming Arabic tweets: heavy stemming and light stemming. The material in this chapter is derived in a large part from our paper (Albogamy and Ramsay, 2016b).

Chapter 7, \textit{Syntactic Parsing for Arabic Tweets}, describes the development of the first treebank for Arabic tweets by using two different strategies: a rule-based parser only and using a rule-based parser and a data-driven parser in a bootstrapping technique. The material in this chapter is derived from our paper (Albogamy et al., 2017).

Chapter 8, \textit{Conclusion and Future Work}, concludes with the final remarks of the thesis. The chapter ends with discussion of possible future improvements and research directions.
Chapter 2

Structural Analysis of Arabic: A Review

The purpose of this chapter is to investigate the complexity of Arabic syntactic structure, investigate the sources of ambiguity and study the properties of text in Arabic social media that may create additional challenges for NLP tasks such as POS tagging, stemming and parsing (RO1).

2.1 Introduction

As pointed out in the first chapter, the aim of the work described here is to analyse the linguistic structure of Arabic tweets in order to understand and exploit them, since it is impossible to understand the full contents of a text of any kind unless we understand its structure. This task involves three linked subtasks, tagging, stemming and parsing for Arabic tweets. To undertake these tasks, we should be thoroughly familiar with the sources of ambiguities that we will face. Arabic tweets have an exceptional level of lexical and structural ambiguity. The nature of the text content of tweets differs from that of traditional text, but there are tweets that are written perfectly formally. Those tweets share the same linguistics issues of any ordinary language, while there are tweets that are not formally written that cause more genre-specific problems.
Hence, it is appropriate to begin this chapter by shedding some light on the ambiguities that happen in natural languages with particular emphasis on those related to the tasks in this research (Section 2.2). Then, we will review in more detail the sources of ambiguities in Arabic (Section 2.3). At the end of the chapter, we will explain the characteristics of tweets (Section 2.4).

2.2 Ambiguity in natural languages

In natural languages words, phrases or sentences may have one or more meanings. If they have more than one meaning, then they are said to be ambiguous (Crystal, 2011). Ambiguity is considered one of the major reasons why processing natural languages is a challenging task. It can be subdivided into three main types: lexical, structural and scope ambiguities. In this section, all three types are discussed in greater detail.

2.2.1 Lexical ambiguity

The ambiguity which arises due to an individual word having alternative meanings is referred to as lexical (or word-level) ambiguity. Four types of lexical ambiguities are reviewed below.

- Polysemes vs. Homonyms:

  Polysemes are words which have the same spelling and pronunciation with different but related meanings, whereas words with the same spelling and pronunciation but with unrelated meaning are called homonyms. For example, polysemy can be represented by the word ‘book’. It has two related meanings, i.e. “a bound collection of pages” and “a book-length publication in digital format” as shown in (2.1). On the other hand, homonymy can be represented by the word ‘type’. It has two unrelated meanings, i.e. “to write via keyboard” and “a sort” as shown in (2.2).
(2.1) Polysemes

a. He read a green **book**.

b. I downloaded an interesting **book** from the website.

(2.2) Homonyms

a. You can **type** very fast!

b. There are different **types** of stress.

- **Homophones vs. Homographs:**

Homophones are words which have the same pronunciation but different spelling and meaning, whereas words with the same spelling which differ in meaning are called homographs. For example, the word ‘*bite*’ (it is a verb that means to use the teeth to cut into something) and ‘*byte*’ (a group of binary digits or bits operated on as a unit) are homophones because they have the same pronunciation /b-/ but differ in spelling and meaning as shown in (2.3). On the other hand, the word ‘*bank*’ is a homograph because the same spelling represents two words with different meanings, for example, ‘*bank*’ as a verb means “to deal with a bank”, but as a noun means “an edge of a river” as shown in (2.4).

(2.3) Homophones

a. Did the dog **bite** the boy?

b. There are eight bits in a **byte**.

(2.4) Homographs

a. Follow the footpath along the river **bank**.

b. He is going to **bank** the money.

There are many words that are both homonyms and homographs such as ‘*book*’ in (2.1) and ‘*type*’ in (2.2), and there are also some words that are homographs but are not homonyms since they have the same spelling but different pronunciations and meanings such as ‘*desert*’;
it can be a verb that means abandoning (in this case the stress is on the final syllable when pronounced), or a noun that means a dry place with camels (in this case the stress is on the first syllable when pronounced).

### 2.2.2 Structural ambiguity

The ambiguity which arises due to a sentence having more than one syntactic analysis is referred to as structural ambiguity. This ambiguity does not result from the range of meanings of single words but from the relationship between the words and the sentence structure. Three common types of structural ambiguities, namely attachment, word order variation and pronoun-dropping ambiguities, are described below.

- **Attachment ambiguity**

  In this kind of ambiguity, a particular constituent of a sentence could be attached to more than one part of a sentence. A prepositional phrase (PP) is a common pattern of attachment ambiguity that can modify NP or VP. The way that PPs are attached to a sentence may lead to different interpretations for the same sentence.

  (2.5) I saw the man on the hill with a telescope.

  The sentence (2.5) has two PPs ‘on the hill’ and ‘with a telescope’. Hence, it has five different reasonable interpretations:

  1. \(I\ saw\ [the\ man\{on\ the\ hill\}[with\ a\ telescope]]\) “I saw the man. The man was on the hill. The hill had a telescope.”

  2. \(I\ saw\ [the\ man\{on\ the\ hill\}[with\ a\ telescope]]\) “I saw the man. The man was on the hill. The man had a telescope.”

  3. \(I\ saw\ [the\ man\{on\ the\ hill\}[with\ a\ telescope]]\) “I saw the man. The man was on the hill. I used a telescope to see him.”
CHAPTER 2. STRUCTURAL ANALYSIS OF ARABIC: A REVIEW

4. I saw [the man][on the hill[with a telescope]] “I saw the man. I was on the hill. The hill had a telescope.”

5. I saw [the man][on the hill][with a telescope] “I saw the man. I was on the hill. I used a telescope to see him.”

- **Word order variation**

Most natural languages have a regular structure for a sentence (e.g. SVO in English and VSO in Arabic). However, a sentence can be rewritten in different acceptable structures which can lead to ambiguity. This can be seen in English as in (2.6). Changes in word order do not generally lead to ambiguity in English, but in languages that have a high syntactic flexibility such as Arabic, it is both frequent and problematic, as we will see in more detail in Section 2.3.2.

(2.6) Example of word order variation

a. [A family][was sitting][on the front steps of the house] [S+V+PP]

b. [On the front steps of the house][was sitting][a family] [PP+V+S]

- **Pronoun-dropping (null anaphora)**

Pro-dropping occurs when pronouns are omitted since they are considered unnecessary or redundant in a sentence. This process leads to structural ambiguity since a sentence may have more than one meaning based on whether there is a pro-drop or not in the sentence. Some languages are considered pro-drop languages such as Arabic, Italian and Japanese, whereas English and German are non-pro-drop languages. Nonetheless, in fact, English language does allow both subject and object pronouns to be omitted in imperative sentences as in (2.7).

(2.7) Example of pro-dropping in English[1]

Φ Keep Φ out of the sight and reach of children.

[1]The symbol Φ shows the position of the omitted pronoun.
“(You) keep (it) out of the sight and reach of children.”

In Section 2.3.4 we will discuss the Arabic language as an example of pro-drop languages.

2.2.3 Scope ambiguity

This kind of ambiguity is a common type of ambiguity that occurs when a sentence includes more than one noun phrase (NP) that has a quantifier term such as ‘every’, ‘most’, ‘some’, or others in the determiner position. It arises at a logico-semantic level (Kurtzman and MacDonald 1993). In this situation a sentence has more than one meaning because the words used and the relationships between them underdetermine the meaning. For example, the sentence in (2.8) is syntactically unambiguous but has different meanings.

(2.8) Example of scope ambiguity

Two students met with every teacher.

The sentence in (2.8) has different meanings since it contains a quantifier expression ‘every’:

1. “There were two particular students and each one of them met all of the teachers.”

2. “Each teacher was visited by two students, but possibly different students meeting with each.”
CHAPTER 2. STRUCTURAL ANALYSIS OF ARABIC: A REVIEW

2.3 Ambiguity in Arabic

Arabic is a term that refers to the existence of many varieties of Arabic (Zaidan and Callison-Burch, 2014). Those varieties include many spoken forms (regional dialects), and one written form called Modern Standard Arabic (MSA). MSA is the formal written standard of the education, culture and media across the Arab world. MSA is derived from Classical Arabic (CA) which is the language of the Islamic Holy Book (The Qur’an) (Biadsy et al., 2009). Arabic is written from right to left and has many complexities and subtleties (Fehr, 2013). Its basic alphabet contains 28 letters that is extended to 90 elements by using additional marks, shapes and vowels. The 28 letters represent consonants and three vowels (semiconsonants). There are also three short vowels (i.e. fatha, damma and kasra) and other phonetic information such as shadda (consonant doubling) and sukuun. All short vowels and phonetic information are represented by diacritics, which are optional, and indeed are usually omitted, in the MSA writing system.

As seen before, there are common types of ambiguity that can be found in natural languages. Nonetheless, some languages such as Arabic have different properties, particularly in their written form which lead to various kinds of ambiguity. In this section, the main sources of ambiguity in Arabic (Tayli and Al-Salamah, 1990) will be discussed in detail.

2.3.1 Missing diacritics

In Arabic, there are three sets of diacritics, namely short vowels, double case endings and syllabification marks which can be written as short strokes placed above or below the consonant letter but which are usually omitted (Zitouni et al., 2006).

There are three short vowels in the Arabic writing system as follows:

- Fatha: It is an oblique dash over a consonant as in ب and represents the /a/ sound.
• Damma: It is an apostrophe-like shape written above the consonant as in ح and represents the /u/ sound.

• Kasra: It is an oblique dash written below the consonant as in ح and represents the /i/ sound.

Doubled case ending, called tanween, occurs when short vowels are used at the end of words by doubling the diacritic that indicates adding /n/. It suggests indefiniteness and there are three different diacritics used for tanween (similar to short vowels): fatha tanween as in ح /ban/, damma tanween as in ح /bun/ and kasra tanween as in ح /bin/.

The third set of diacritics which exists in the Arabic writing system is syllabification marks. There are two syllabification marks as follows:

• Sukuun: It is a small circle and sits over the letter which does not have a vowel as in ح. It represents the boundaries between syllables.

• Shadda: If the same consonant occurs twice in a word, with no vowel in between, then shadda (gemination) is used. So, the consonant is written only once and shadda is written above it instead of writing consonant + sukuun + consonant. The shadda is normally combined with a short vowel as in ح.

Diacritics play a significant role in understanding classical Arabic texts since they are often the only way to differentiate between different words (derived forms) and between inflected forms of the same word. Since diacritics are optional in the MSA writing system, many words with different diacritic patterns appear to be identical (Zitouni et al., 2006). This situation leads to considerable ambiguities which make processing the Arabic language a challenge. The following examples show the role of diacritics on the meaning of the words.
(2.9) Distinguish between a noun and verb

\[ 
\begin{array}{ll}
\text{noun} & \text{كتب} \quad \text{“books”} \\
\text{verb}  & \text{كتب} \quad \text{“he wrote”} \\
\end{array}
\]

(2.10) Distinguish between active and passive

\[ 
\begin{array}{ll}
\text{active} & \text{كتب} \quad \text{“he wrote”} \\
\text{passive} & \text{كتبت} \quad \text{“was written”} \\
\end{array}
\]

(2.11) Distinguish between gender and person

\[ 
\begin{array}{ll}
k.tb & \\
n.k.tu & \text{(1st person singular)} \quad \text{“I wrote”} \\
k.ab.t & \text{(2nd masculine singular)} \quad \text{“You wrote”} \\
k.q.ti & \text{(2nd feminine singular)} \quad \text{“You wrote”} \\
k.b.t & \text{(3rd feminine singular)} \quad \text{“She wrote”} \\
\end{array}
\]

As seen before, the diacritics are used to differentiate between different forms in Arabic and they are optional in the MSA writing system. So, if the diacritics are omitted, it becomes impossible to distinguish between the different forms. As we have just seen, this means that Arabic raises much more lexical ambiguity than most other languages.

### 2.3.2 Word order variation

Arabic has a high syntactic flexibility [Daimi 2001]. The canonical order of Arabic sentences is VSO. However, a range of non-canonical orders such as OVS, SVO and VOS are also possible [Ramsay and Mansour 2006]. Hence, the relatively free word order of Arabic is considered one of the sources of its ambiguity [Attia 2008]. In Arabic, a sentence can be constructed in various orders with the same number of words. The meaning of words can be affected by
CHAPTER 2. STRUCTURAL ANALYSIS OF ARABIC: A REVIEW

their locations within the sentence which leads to a huge amount of ambiguity in diacriticless settings since it is not easy to distinguish the subject from the object of a sentence as shown in (2.12).

(2.12) Example of word order variation:

a. ʾI. ʾË@ 	‘« YËñË@ (SVO/OVS)
   Al+klb ED Al+wld
   the+dog bit the+boy

b. “The dog bit the boy”
   Al+klbu ED Al+wlda
   the+dog(nominative subject) bit the+boy(accusative object)

c. “The boy bit the dog”
   Al+klaa ED Al+wldu
   the+dog(accusative object) bit the+boy(nominative subject)

In (2.12a), the sentence is written without diacritics, so there are two possible meanings for it according to the differentiation between nominative subject and accusative object as shown in (2.12b) and (2.12c). In (2.12b), the first noun Al+klb “the dog” is in the nominative case Al+klb and the second noun Al+wld “the boy” is in the accusative case Al+wlda, so the sentence means “The dog bit the boy”. While in (2.12c), the first noun Al+klb “the dog” is in the accusative case Al+klaa and the second noun Al+wld “the boy” is in the nominative case Al+wldu, so the sentence means “The boy bit the dog”. (2.12b) and (2.12c) have been written with diacritics in order to indicate the distinction between different word order variations. However, normal written Arabic is diacriticless and hence it is ambiguous.

Moreover, Arabic allows zero copulas. A copula is a verb that links a sentence predicate with the subject. In Arabic, there is a special kind of nominal sentence called an *equational*
sentence which consists of two parts: ‘the subject’ which is NP, and ‘the predicate’ which can be a NP, an adjective phrase or a PP such as التدريس في المدرسة Al+mdrs fy Al+mdrsap “the teacher at the school”. To make matters even more complicated, the order of the subject and the predicate is reversed if the subject is indefinite as in في المدرسة مدرس fy Al+mdrsap mdrs “at the school, there is a teacher”. Zero copula in conjunction with the existence of construct NPs (see Section 2.3.5 for more details) makes it hard to determine the boundaries between phrases.

2.3.3 Cliticisation

Clitics are defined as morphemes that have the syntactic characteristics of a word; nonetheless, they are bound to other words (Crystal, 2011). Arabic is a clitic language since it contains numerous clitics such as the determiner, pronouns, prepositions and conjunctions which can be attached to Arabic words as prefixes or as suffixes. Clitics are considered another source of ambiguity in Arabic since a sentence can be constructed from what seems to be a single word but, in fact, one or more clitics are attached to the word. The reason for this is because clitics use the same alphabets as that of Arabic words with no distinguishing marks, such as the English apostrophe (Attia, 2008). As a result, one Arabic word could have more than one different morphological segmentation. For instance, the word والي wAIY has five different morphological analyses as shown in (2.13) (Habash et al., 2009). To make matters worse, cliticisation ambiguity usually is combined with lexical ambiguity which gives each lexical segmentation a different meaning, as in (2.13)
Example of clitics:

\[ \text{والي} \quad \text{wAly} \quad \text{‘ruler’} \]
\[ \text{وبالي} \quad \text{w+AlY+y} \quad \text{‘and to me’} \]
\[ \text{وبأالي} \quad \text{w+|ly} \quad \text{‘and automatic’} \]
\[ \text{وبألي} \quad \text{w+ly} \quad \text{‘and follow’} \]
\[ \text{وبألي} \quad \text{w+ly} \quad \text{‘and my clan’} \]

### 2.3.4 Pro-dropping

In the pro-drop theory, “a null category (pro) is allowed in the subject position of a finite clause if the agreement features on the verb are rich enough to enable its contents to be recovered” (Baptista, 1995). In Arabic, verbs can recover missing subjects since they can indicate the gender, person and number of the omitted pronominal subject, so Arabic is a pro-drop language (Attia, 2008). This property, which is called آلمستتر التصريح Al+Dmyr Al+msstr ‘tacit pronoun’, can lead to structural ambiguity since there is a challenge to determine if there is an omitted pronoun or not in the subject position, and to identify the antecedent of a dropped pronoun. This challenge is increased when dealing with Arabic verbs since a large number of verbs can be both transitive and intransitive. Moreover, it is impossible in Arabic to differentiate between active and passive forms by examining the surface form. For instance, the sentence in (2.14) has three different interpretations.

Example of pro-drop in Arabic:

\[ \text{السمكة} \quad \text{Al+smkp} \quad >\text{klt} \quad \text{ate(feminine) the+fish} \]

“The fish ate”, “(She) ate the fish” or “The fish was eaten”

Three facts cause the ambiguity in (2.14): (1) it can be understood from the feminine mark
on the verb أَكَلَت that there is a drop-pronoun which is هي ‘She’ (in this case, the meaning is أَكَلَت (هي) السمكة >akalat (hy) Al+smkp (‘(She) ate the fish’); and (2) the verb أَكَلَ >akala ‘to eat’ can be both transitive and intransitive (in this case, the verb is intransitive so the meaning is أَكَلَت السمكة >akalat Al+smkp “The fish ate”); and (3) the verb أَكَلَت أَكَلَت >ukalat Al+smkp “The fish was eaten”).

2.3.5 Noun multi-functionality

Arabic nouns are known by their multi-functionality. They can function as adjectives, possessive determiners (Idafa), adverbs, quantifiers or even prepositions. Moreover, they can be linked together without any apparent marks and there is no possessive preposition ‘of’ or suffix ‘s’ on possessing noun. Therefore, they are considered another source of ambiguity in Arabic. The examples in (2.15)-(2.19) show how a noun can be used as an adjective, a possessive determiner, an adverb, a quantifier or a preposition respectively and it can still be functioned as an ordinary nominal function.

(2.15) Noun as an adjective

حقيقة ظهر
Zhr Hqybp
“A backpack bag”

(2.16) Construct phrase (Idafa)

وزير الداخلية
Al+dAxlyp wzyr
“The minister of the interior”
(2.17) Noun as an adverb

\[ >sAsA \text{xT} \text{Al+Hl h*A} \]

“This solution is wrong basically”

(2.18) Noun as a quantifier

\[ \text{مجموعة اللاعبين حاضرون} \]

HADrwn Al+lAEbyn jmyE

“All players are present”

(2.19) Noun as a preposition

\[ >byh >mAm \text{wqf} \]

“He stood in front of his father”

All of the phenomena that we have discussed in this chapter occur in many languages such as English but in English, for example, there is usually enough information present to eliminate most of the potential interpretations. However, the combination of these phenomena leads to huge ambiguities in Arabic.
2.4 Properties of English and Arabic tweets

In the last few years Internet users have tended to shift from traditional communication tools such as traditional blogs or mailing lists to microblogging platforms such as Twitter. People post about their life, share opinions on variety of topics and discuss current issues. There were nearly 500 million tweets a day in September 2014. As a result, Twitter has become one of the most important social information platforms. It has dominated recent research in the web data because it is public by default so the privacy of its data is less problematic than other platforms, and it is easy to obtain and flexible to use in different NLP applications. It also provides a single streaming interface for gathering large datasets. In this section, some of the main user intentions on Twitter and many phenomena that are frequent in Twitter will be mentioned.

User Intentions on Twitter

Java et al. (2007) have analysed a large amount of Twitter posts. He read each post and categorised it based on its author’s intention. According to this analysis, a taxonomy of the main user intentions on Twitter was presented as follows:

- **Daily Chatter** In this category, people talk about their daily routine or what they are currently doing. They represent the largest and most common users of Twitter.

- **Conversations** People comment on or reply to their friends’ posts by using the reply feature in Twitter. According to this analysis, almost 21% of users used this form of communication.

- **Sharing information** People in Twitter share information by posting new information or retweet (resend) information to their followers. Due to the small character limit,
about 13% of all posts in this category contain links (shortened URLs)\(^4\) to the source of information.

- **Reporting news** Weather reports and news fall into this collection. Some automated users or agents post updates by using RSS feeds. People also comment on current events or even report the latest news on Twitter.

### Main User Categories on Twitter

According to Java et al. (2007), users on Twitter can be categorised as follows:

- **Information source** Some users have a large number of followers because they are considered a kind of information source due to the valuable nature of their posts.

- **Followers** Most relationships in Twitter fall into this category. A user may have friends, colleagues, family and fans on their follower list.

- **Information seeker** An information seeker is a person who follows other users regularly, but might post rarely.

### Phenomena in Twitter

There are many phenomena that appear in Twitter frequently that are not as frequent, or are entirely absent, in other genres. Most of these phenomena can be observed in tweets regardless of the language (Gimpel et al., 2011). The following is the list of the phenomena in Twitter with examples from English and Arabic\(^5\):

- **Tweet** Any message with 140 characters or less posted to Twitter.

  *English:* Why does Scotland want independence? It’s culture vs. economics
  
  http://nyti.ms/YDUYNw

  *Arabic:* إنك لما هو ما هو، وإنك لما هو ما هو

\(^4\)Twitter has a feature which alters a URL of any length to be 22 characters.

\(^5\)For each phenomenon there are two different examples, one from English and the other from Arabic, which are related to different topics, not meant to be a translation for each other.
• **At-Mention** A Tweet containing another user’s Twitter username, preceded by the ‘@’ symbol. Mentions can occur anywhere in the Tweet except the beginning.

*English:* The @UKLabour Party conference this weekend is set to net Manchester more than 25m

*Arabic:* @7aamed1

• **Reply** Tweet that begins with another user’s username and is in reply to one of their Tweets.

*English:* @JheneAiko I was in Manchester last weekend and got #SouledOut. Last copy of the deluxe version in HMV. Amazing album

*Arabic:* كل ما يجري هو محاربة العرب ووحدتهم وتوحدهم @SaraSoso987163

• **Hashtag** This is used to mark keywords or topics in a Tweet. It can occur anywhere in the Tweet. Users use it to categorise their tweets.

*English:* Manchester #cyclist launches campaign to shame dangerous drivers & fellow riders by posting video footage on Youtube: http://ow.ly/BqcgG

*Arabic:* #خادم_الحرمین_القرينين_يأم_باستعفاته_1440_حاج_العالم_لأداء_فرضية_خ_هذا_العالم

• **Discourse markers** Tweets may include discourse makers such as RT which is used when a user re-tweet (re-send) another user’s tweet

*English:* RT @southernkrazed: Follow @SunstarGUM tonight for great dental care tips and fun facts about your smile!

*Arabic:* RT @ReNgo_Sport: صورة: لـ أحد عشاق جيبارد كاس العالم http://t.co/vY0feFK3F2

• **Emoticons and Emoji** These are used by users to express their feelings or emotions in tweets. An emoticon is constructed by using traditional alphabetics or punctuation, usually a facial expression, while emoji are symbols provided in software as small pictures.

*English:* Happy Birthday!! :) @IsawEmpay
• **Dialects** Users in Twitter use their own dialects and/or accent in their tweets which usually lead to nonstandard spelling.

*English:* *im gonna* centre

*Arabic:* أنا اني بغي ريتويت؟

• **Multiwords Abbreviations** Due to the limited number of characters in a tweet, people tend to use short forms in which each character corresponds to a word in the actual regular phrase.

*English:* Thank you for the birthday tweets *ily* all. (*ily* means *I love* you)

*Arabic:* هذه شركة ذم (ذو مسؤولية محدودة ذم)

• **Expressive Lengthening** This is used by users to express subjectivity and sentiment.

*English:* This look sooo **coolll** @HARDFEST

*Arabic:* هذا التصميم **حلووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووووо
As seen above, all phenomena in Twitter are frequent in Arabic tweets as in other languages (e.g. English) except the capitalisation. However, Arabic tweets show more interesting linguistic phenomena that are worth mentioning. These phenomena are as follows:

- **Arabic dialects** There is an extensive usage of different Arabic dialects in Twitter. Most Arabic speakers use different Arabic dialects in their daily interactions which differ from Modern Standard Arabic (MSA) in terms of lexical and function words. With the spread of online social networking in Arab countries, these dialects are used in written forms on Twitter. There are six main Arabic dialects, namely Gulf, Egyptian, Iraq, Levantine, Moroccan and Yemeni ([Versteegh](#) 1997). As a result, new function words, morphological patterns, lexical choices and different pronunciations of different letters are introduced by different dialects ([Darwish et al.](#) 2012). In the following, one example of each category will be mentioned.

1. **Function words** The Arabic word کذا (means 'like this') is expressed as کذا k*A, هيك hyk, and كده kdh in the Gulf, Levantine and Egyptian dialects respectively. The following tweet is an example of using the Gulf form of this word in Twitter:

   "هَوَّامَ كَذا" (means “It’s always like this”)

2. **Morphological patterns** The Egyptian dialect, for example, uses a new negation construct that does not exist in MSA. The word معرفتش mhft$ (means did not know) is composed of m+Erft+$$. The tweet below is an example of using this word in Twitter:

   "معرفتش انت في البيت" (means “I did not know you were at home”)

3. **Lexical choices** There are different lexical choices to express the same concepts in Arabic dialects. For instance, the Arabic word أريد (means I want) is ex-
pressed as بدي اشوفك اليوم (means “I want to see you today”)

- **Romanised Arabic** Arabic users tend to use Latin letters and Arabic numerals to write Arabic tweets because the actual Arabic alphabet is unavailable for technical reasons, difficult to use or they speak Arabic but they cannot write Arabic script (e.g. Muslims in non-Arab countries). The following tweet is an example of Romanised Arabic tweets:

  Mayta you3 (Arabic script ميت يوع)(means “I am starving”)

In this phenomenon, where users do not use Arabic script, there will be an inconsistency regarding which characters people use, which leads to extra complications because there is no one-to-one mapping between these characters and proper Arabic characters.

- **Borrowing English words** Arabic users borrow some multiword abbreviations from English. They use them as written in Latin letters or they use their Arabic transliteration. For example, LOL in English (laugh out loud) is written in Arabic as لول. The tweet below shows an example of these words:

  نسيت تاريخ ميلادي لول (means “I forgot my birthdate lol”)

### 2.5 Summary

In this chapter, we have explained in detail the challenges of Arabic tweets processing. Arabic tweets have the same linguistic issues that any ordinary Arabic text has as well as genre-specific problems. Although each of these phenomena occurs in other languages, the way
they combine in Arabic leads to challenges when processing Arabic tweets. We discussed the fact that the lack of diacritics makes the combination of pro-dropping, zero-copula and free word order worse for MSA than for other languages. We also pointed out that tweets are even more difficult because of open vocabulary and extreme informality, which means that standard grammar rules do not always apply.

In the next Chapter, we will shed light on POS tagging, stemming and syntactic parsing for Arabic and will discuss some examples of state-of-the-art POS taggers and parsers.
Chapter 3

State-of-the-art Approaches to Arabic NLP: A Review

In this chapter, we explain the NLP pipeline and discuss some of the issues involved in adapting it to suit Arabic tweets. We also review the state-of-the-art Arabic NLP approaches (i.e. POS taggers, stemmers and parsers) in order to be thoroughly familiar with the existing NLP tools for MSA text before undertaking any experiments on Arabic tweets.

3.1 Introduction

The state-of-the-art NLP systems that have been designed for the English language are not suitable for Arabic, mainly due to its complex morphological, structural and grammatical nature (Habash et al., 2009). Hence, many tools have been developed for carrying out different components of an Arabic NLP pipeline (e.g. Tokenisation, POS tagging, morphological analysing, etc.), while other tools such as parsers were trained on MSA corpora in order to be used in an Arabic context. These tools are used in the early stages within most NLP systems such as information extraction (IE), machine translation (MT), question answering (QA), sentiment
3.2 A Typical NLP Pipeline

Language is an activity whose intended end point is that the hearer should do something. What they are expected to do may be some physical action, it may be a linguistic action, it may just be to store what has been said for future reference, but in every case they are expected to do something. It is widely agreed that in order to describe how this happens it is useful to provide several levels of linguistic analysis – morphology, syntax, semantics, pragmatics – and in many computational treatments these are implemented as a pipeline of independent processes. Figure 3.1 shows a typical example of such a pipeline.

The stages in the pipeline work from the input text to the final semantic or pragmatic interpretation, but the reasons why they are there are best explained by working backwards from the final interpretation. In order to decide on the action, the system has to understand and infer the intended meaning of an utterance (planning and inference). In order to do that, it has to have a meaning representation (semantic interpretation). In order to get the meaning representation, it is necessary to find what the arrangement of words is (syntactic analysis). To do that, the system has to know what the words were (morphological analysis). Before even that, the system should take the original input string and break it into tokens (tokenisation). Typically, NLP systems achieve these tasks sequentially as a pipeline starting with tokenisation.
and ending with planning and inference as showed in Figure 3.1. The work reported in this thesis is concerned with the structural parts of this pipeline, as marked by the dashed line in Figure 3.1, but we have included the later stages for completeness, since they are an important part of the overall process of language processing.

1Numerous authors, e.g. (Jurafsky and Martin 2000), (Hahn et al. 2007), have drawn diagrams that they refer to as “The standard NLP pipeline”; Figure 3.1 is an amalgam of several such diagrams.
Figure 3.1: A typical NLP pipeline.
As seen in Section 2.4, the microblogging genre exhibits much more language variation, tends to be less grammatical than normal text, and makes frequent use of abbreviations, dialects and foreign words. Twitter users also code switch between different languages and dialects (El-fardy et al., 2014). Therefore, adaptation of the NLP pipeline to the specifics of the microblog genre is required. In addition to the standard NLP pipeline, a language identification stage is added to the pipeline and external knowledge is used in the pipeline (Figure 3.2).

**Language identification.** It is crucial to detect the language(s) in which a text is written in order to process the text by using suitable NLP tools. This task requires the use of word-level and sequence-level features. Traditional documents are often written in a specific language whereas tweets may contain more than one language. In other words, Twitter users code switch between different languages in tweets, for example, they switch between English and French or Arabic. They might use different dialects from the same language, but it is difficult to detect dialect switch in tweets because tweets are very short and therefore tend not to contain enough information to detect dialects in a tweet.

**External knowledge.** Normalisation is proposed as a solution for overcoming or reducing linguistic noise (Sproat et al., 2001). It involves two steps: first, the identification of orthographic errors in words, and second, the correction of these errors. External sources of knowledge such as regular expression rules and gazetteer lists are also used.
Figure 3.2: The NLP pipeline for Figure 3.1 adapted for tweets.
The difference in the adapted NLP pipeline for tweets (Figure 3.2) for English and Arabic is that in English tagging and morphology are often done in one step whereas in Arabic finding out what the words are is a difficult task. Tagging and stemming are typically carried out as separate stages, because Arabic makes extensive use of clitics where the letters are attached together to form a word with no distinguishing marks. As a result, an Arabic word could have more than one different morphological segmentation (Attia, 2008). Therefore, the morphological analysis stage in Figure 3.2 is split into two tasks (e.g. tagging and stemming) to be suitable for manipulating Arabic tweets as shown in Figure 3.3. Coarse-grained tagging can be carried out on the basis of simple clues using the initial and final characters of space-separated tokens and on the coarse grained tags of the surrounding items. These coarse grained tags are then used to guide stemming, which depends on more detailed knowledge of the structure of words, and thence to a more fine-grained set of tags.

In this research, we adopt the NLP pipeline in Figure 3.3 with the focus on all tasks related to structural analysis. It is worth mentioning that each tagger we use has its own tokeniser which aims to split sentences into words at punctuation or white space. The reliable tweet language identification feature in Twitter API allows us to only retrieve those tweets written in Arabic, Arabic mixed with English and Arabizi. It is a straightforward task to distinguish Arabic from English and Arabizi; to distinguish between English and Arabizi we note that if a tweet contains a mixture of Arabic and Roman characters, the Roman characters are never Arabizi. Although there is not in general enough information in a tweet to detect dialects and the distinction between MSA and the Gulf dialect is hard to observe, we map the most frequently occurring dialect words from our corpus to their in-vocabulary equivalents (Zaidan and Callison-Burch, 2014).

\[^2\text{https://dev.twitter.com/streaming/overview/request-parameters}[^2]
Figure 3.3: The NLP pipeline proposed for Arabic tweets.
3.3 POS tagging

POS tagging is the process of assigning each word of a sentence a part of speech tag (e.g. noun, verb or adverb) (Brill, 1992). It provides fundamental information about word forms which has countless applications in NLP. This is a non-trivial task because some words can belong to more than one part of speech (POS tag) at the same time, due to their ambiguities as in Arabic which is a highly inflectional and derivational language (Attia, 2012). The inflection is a process by which many forms of a word can be produced by adding affixes to its root form. For instance, the regular sound verb ٥دارسة (darasa “he studied” has 34 active inflectional forms, 28 passive forms, 14 jussive forms and 14 future forms. Table 3.1 shows only the active inflectional forms for the verb since there is not enough space to list all its forms in the table.

---

3http://qutrub.arabeyes.org/index?verb=٥درسة
<table>
<thead>
<tr>
<th></th>
<th>1st</th>
<th>masculine</th>
<th>feminine</th>
<th>Past</th>
<th>Present</th>
<th>Imperative</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>singular</td>
<td>دَرَاسَت / درست</td>
<td>“I studied”</td>
<td>أَدْرَس / أدرس</td>
<td>“I study”</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>dual plural</td>
<td>دَرَاسِو / درست</td>
<td>“We studied”</td>
<td>نَدْرَسُ / ندرس</td>
<td>“We study”</td>
<td>-</td>
</tr>
<tr>
<td>2nd</td>
<td>masculine</td>
<td>دَرَاسَت / درست</td>
<td>“You studied”</td>
<td>تُدْرَسُ / تدرس</td>
<td>“You study”</td>
<td>أَدْرُس / أدرس</td>
</tr>
<tr>
<td></td>
<td>dual</td>
<td>دَرَاسِتُ / درست</td>
<td>“You studied”</td>
<td>تُدْرَسُ / تدرس</td>
<td>“You study”</td>
<td>أَدْرُس / أدرس</td>
</tr>
<tr>
<td></td>
<td>plural</td>
<td>دَرَاسُت / درست</td>
<td>“You studied”</td>
<td>تُدْرَسُ / تدرس</td>
<td>“You study”</td>
<td>أَدْرُس / أدرس</td>
</tr>
<tr>
<td></td>
<td>feminine</td>
<td>دَرَاسَتِ / درست</td>
<td>“You studied”</td>
<td>تُدْرَسُ / تدرس</td>
<td>“You study”</td>
<td>أَدْرُس / أدرس</td>
</tr>
<tr>
<td></td>
<td>dual</td>
<td>دَرَاسِتُ / درست</td>
<td>“You studied”</td>
<td>تُدْرَسُ / تدرس</td>
<td>“You study”</td>
<td>أَدْرُس / أدرس</td>
</tr>
<tr>
<td></td>
<td>plural</td>
<td>دَرَاسُتُ / درست</td>
<td>“You studied”</td>
<td>تُدْرَسُ / تدرس</td>
<td>“You study”</td>
<td>أَدْرُس / أدرس</td>
</tr>
<tr>
<td>3rd</td>
<td>masculine</td>
<td>دَرَاس / درس</td>
<td>“He studied”</td>
<td>يُدْرَسُ / يدرس</td>
<td>“He studies”</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>dual</td>
<td>دَرَاسُ / درس</td>
<td>“They studied”</td>
<td>يُدْرَسُ / يدرس</td>
<td>“They study”</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>plural</td>
<td>دَرَاسُ / درس</td>
<td>“They studied”</td>
<td>يُدْرَسُ / يدرس</td>
<td>“They study”</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>feminine</td>
<td>دَرَاسَت / درست</td>
<td>“She studied”</td>
<td>تُدْرَسُ / تدرس</td>
<td>“She studies”</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>dual</td>
<td>دَرَاسُ / درست</td>
<td>“They studied”</td>
<td>تُدْرَسُ / تدرس</td>
<td>“They study”</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>plural</td>
<td>دَرَاسُ / درست</td>
<td>“They studied”</td>
<td>تُدْرَسُ / تدرس</td>
<td>“They study”</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 3.1: The active inflection forms for the regular sound verb دَرَاس / درس “he studied”.

The derivation is also an important characteristic of Arabic, where new Arabic words can be formed by applying morphological patterns to their root forms as shown in Table 3.2. These characteristics ease the process of enriching the Arabic vocabulary by using the word roots and morphological patterns. Moreover, diacritics that specify gemination “shadda” (consonant letter doubling) and short vowels (fatha, damma, kasra, etc.) are optional in the Arabic writing system (Habash et al., 2009). Consequently, any Arabic word written without diacritics can belong to different parts of speech. For instance, the word *Adrs* could be a past verb for singular 1st person, masculine singular 2nd person, feminine singular 2nd person or feminine singular 3rd person as in Table 3.1. A more ambiguous case happens when an Arabic word can be classified as a noun or a verb due to lack of diacritics, for example, word *drs* could be a noun “lesson” as in Table 3.2 or a verb which has 90 different forms as in Table 3.1. Hence, the complexity of the Arabic morphology with the lack of diacritics in Arabic writing system make POS tagging for Arabic a substantial challenge.

<table>
<thead>
<tr>
<th>Word Derivative form</th>
</tr>
</thead>
<tbody>
<tr>
<td>درس / درس &quot;lesson&quot;</td>
</tr>
<tr>
<td>درس مفرد</td>
</tr>
<tr>
<td>درس مثنى</td>
</tr>
<tr>
<td>درس جميع</td>
</tr>
</tbody>
</table>

Table 3.2: The derivative words for word درس *drs* "lesson".

### 3.3.1 POS tagging approaches

Approaches to POS tagging can be classified into three categories: rule-based tagging, statistical tagging and transformation-based learning as shown in Figure 3.4.

#### 3.3.1.1 Rule-based tagging

Rule-based taggers use a set of rules which are manually hand-written to assign a tag to each word. An example of these rules is that a word following a determiner must be a noun. There-
fore, to use a rule-based tagging, a good set of rules has to be available that must be written and checked by human experts. The disadvantage of this approach is that these rules are hard to develop and it does not work when applied to different genres. ENGTWOL tagger is an example of this category (Voutilainen, 1993).

### 3.3.1.2 Statistical tagging

Statistical taggers use annotated training data to extract information about the contexts in which a word should be assigned a particular tag and use this information to tag that word in the unannotated text (Cutting et al., 1992). TNT tagger is an example of this category (Brants, 2000). A statistical tagger could use a Hidden Markov Model (HMM) or a classifier. A HMM can be used to generate the most likely tag sequences. The important feature of HMMs is considering the surrounding context of a word. Classifiers have to have a set of features (e.g. left and right window of words and left and right window of N-letter prefixes and suffixes). They are based on various models such as Neural Nets (Schmid, 1994), Support Vector Machines (Diab, 2009; Habash et al., 2009), Maximum Likelihood (Ramsay and Sabtan, 2009) and Maximum Entropy (Toutanova et al., 2003).
3.3.1.3 Transformation-based learning

The transformation-based approach combines the statistical approach and the rule-based approach. It finds the most likely tag based on a training data and then tries to improve the performance of the classifier by spotting patterns of errors in the output of the original classifier using a certain set of rules. The tagger saves any new rules that has been learnt in the process for future use. One example of this category is the Brill tagger (Brill, 1992). It is fairly common practice to add a transformation-based tagger as a post-processing stage after any of the other kinds of taggers, since transformation-based taggers are particularly effective for correcting systematic errors, and most taggers are prone to making particular kinds of errors.

3.3.2 POS tagsets

POS tags are used to label each word in a corpus. They should show the grammatical class of the word and morphological features such as tense, gender, number, etc. (Sawalha and Atwell, 2013). There are a range of types of POS tagsets, namely fine-grained tagsets and coarse-grained tagsets. The former covers most of a word’s features in detail whereas the latter labels only the main features. Depending upon the potential usage of a corpus, the most suitable set is chosen. Therefore, there is no single optimal POS tagset. This section reviews the most well-known POS tagsets for Modern Standard Arabic (MSA) in the literature.

3.3.2.1 Khoja Arabic tagset

In 2001, Khoja developed a POS tagset for Arabic which is traditional and conventionally based on traditional Arabic grammar categories (Khoja, 2001). This Arabic tagset did not follow the Expert Advisory Group on Language Engineering Standards (EAGLES) guidelines. The reasons for this were that Arabic is a Semitic language while EAGLES recommendations

were designed for European languages; and that some Arabic morphological and syntactic information such as imperative, dual number and inheritance could not be captured if EAGLES guidelines were followed. Inheritance is a key aspect of Arabic, where all subcategories of words inherit properties from the parent categories. Khoja’s tagset consists of 177 tags; 103 nouns, 57 verbs, 9 particles, 7 residual and one punctuation. It covers the morphological features of person, gender, number, case, definiteness and mood [Khoja et al., 2001].

3.3.2.2 Buckwalter tagset

This is an Arabic tagset developed by Tim Buckwalter. It can be used for tokenised and untokenised text. The untokenised tags are produced by the Buckwalter Arabic Morphological Analyser (BAMA) [Buckwalter 2004][5] whereas the tokenised tags are used in the Penn Arabic Treebank tagset[6]. To annotate the Penn Arabic Treebank, the output from Arabic Morphological Analyser was used as a starting point for suggesting a set of candidate analyses for each word, and then the best fitted solution for the context was selected by Arabic linguists [Maamouri et al., 2004]. The tokenised variants were derived from the untokenised tags. It uses a basic 114 subtag symbols such as NSUFF ‘nominal suffix’ and DET ‘determiner’. These subtags are combined to form about 170 morpheme tags such as NSUFF_MASC_SG ‘masculine singular nominal suffix’. A Buckwalter tokenised tagset is about 500 tags and a Buckwalter untokenised tagset is more than 2000 tags [Sawalha and Atwell, 2013].

3.3.2.3 Reduced Buckwalter tagset (Bies tagset)

This tag set has also been referred to as the Reduced Tagset (RTS). It was developed by Ann Bies and Dan Bikel as a collapsed variant of Arabic tags into a smaller set [Maamouri and Bies, 2004] and was introduced by the Linguistic Data Consortium (LCD)[7]. It contains 25 tags inspired by the Penn English Treebank tagset (PTB) and it is a coarse-grained tagset since it

---

[5] Appendix A shows an example of Buckwalter tags.
[6] Appendix B shows the Penn Arabic Treebank Tag Set; basic tags, which can be combined.
[7] https://www.ldc.upenn.edu/
ignores a lot of word inflections in Arabic. It was designed to maximise the performance of Arabic syntactic parsing (Sawalha and Atwell [2013]).

3.3.2.4 Extended reduced tagset

Diab (2007) introduced a new enriched tagset for Arabic. The new tagset consists of 75 tags, of which 25 are RTS tags. Therefore, The Extended Reduced tagset (ERTS) is considered a superset of the RTS tagset. The ERTS tagset enriches the RTS by explicitly adding gender, definiteness and number information to the basic RTS tags.

Table 3.3 shows a comparison of all tagsets mentioned in this section. It summarises the characteristics of each tagset and helps to present the difference between them.

<table>
<thead>
<tr>
<th>Khoja tagset</th>
<th>Motivation for design</th>
<th>Developing a standard tagset for Arabic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main characteristics</td>
<td>It is based on traditional Arabic grammar categories rather than EAGLES. It covers the main classes and subclasses of Arabic</td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td>177 tags; 103 noun, 57 verbs, 9 particles, 7 residual and one punctuation</td>
<td></td>
</tr>
<tr>
<td>Morphological features</td>
<td>Person, Gender, Number, Case, Definiteness and Mood</td>
<td></td>
</tr>
<tr>
<td>Usage</td>
<td>It was used in the design of APT tagger and to annotate its training data</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Buckwalter tagset</th>
<th>Motivation for design</th>
<th>Annotating the Penn Arabic Treebank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main characteristics</td>
<td>Trying to cover detailed grammar features of Arabic</td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td>Tokenised tagset is about 500 tags, untokenised tagset can reach thousands tags</td>
<td></td>
</tr>
<tr>
<td>Morphological features</td>
<td>Person, Voice, Gender, Case, Aspect, Mood, Number, Definiteness, Tense</td>
<td></td>
</tr>
<tr>
<td>Usage</td>
<td>It was used to annotate the Penn Arabic Treebank by using Buckwalter’s morphological analyser</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reduced tagset (RTS)</th>
<th>Motivation for design</th>
<th>Maximising the performance of Arabic syntactic parsing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main characteristics</td>
<td>Derived from the Penn English Treebank tagset</td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td>25 tags</td>
<td></td>
</tr>
<tr>
<td>Morphological features</td>
<td>Person, Gender, Case, Mood and Definiteness.</td>
<td></td>
</tr>
<tr>
<td>Usage</td>
<td>It was used in syntactic annotation of the Penn Arabic Treebank</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Extended Reduced tagset</th>
<th>Motivation for design</th>
<th>Enriching the RTS tagset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main characteristics</td>
<td>Considered a superset of RTS tagset and enriches RTS by explicitly adding gender, definiteness and number information to the basic RTS tags</td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td>75 tags</td>
<td></td>
</tr>
<tr>
<td>Morphological features</td>
<td>Gender, Definiteness and Number on noun phrases</td>
<td></td>
</tr>
<tr>
<td>Usage</td>
<td>To be used for parsing</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.3: Comparison of Arabic POS tagsets.
3.3.3 POS taggers

There has been a considerable amount of work on POS taggers for Arabic in the literature e.g. APT (Khoja et al., 2001), BAMA (Buckwalter, 2004), AMIRA (Diab, 2009), MADA (Habash et al., 2009) and Stanford (Toutanova et al., 2003). The last three are the best and most quoted POS taggers in the literature, which achieve state-of-the-art accuracies in Arabic tagging (Al-abbas and Ramsay, 2012). They are also freely available for downloading and evaluation. In this section, we will discuss AMIRA, MADA and Stanford taggers in detail and we will use the Arabic sentence in (3.1) to show the actual results of each tagger.

(3.1) Arabic sentence

. شارك عشرة طلاب в symposium in students ten participated Ten students participated in the symposium.

3.3.3.1 AMIRA

The AMIRA toolkit has been widely used for preprocessing stages in different NLP applications such as Machine Translation, Information Extraction, Parsing, etc. due to its speed and high performance (Diab, 2009). It includes three main components, namely Tokeniser (TOK), Part of Speech tagger (POS) and Base Phrase Chunker (BPC). We will review all three components of AMIRA toolkit below.

- **AMIRA-TOK**

  This is a clitic tokeniser which conforms to the Linguistic Data Consortium (LDC) standard of clitic tokenisation. It segments off separating conjunctions, affixival prepositions and pronouns, future marker clitics, and definite articles. The user can select among different choices of tokenisation schemes. For example, the output of *wsyktbwnhA* could
be $w \ s \ ykbtwn \ hA$ or $w+s+yktbwn#hA$. The tokeniser has a high F-score of 0.992 on the Penn Arabic treebank.

- **AMIRA-POS**
  AMIRA-POS tagging system uses a Support Vector Machine (SVM) based classification approach using character n-grams as features in the sequence models. It produces the RTS tagset as well as ERTS tagset. The user has the flexibility to use raw text or tokenised text as input to the tagger. The tagger is reported to perform with over 96% accuracy on the Penn Arabic Treebank, 96.15% for RTS and 96.13% for ERTS.

- **AMIRA-BPC**
  This is a base phrase chunker which groups together a sequence of adjacent words to form syntactic phrases such as NPs and VPs. It is considered to be the first step towards shallow syntactic parsing. An English example of base phrases would be $[I]_{NP} \ [would \ eat]_{VP} \ [green \ apples]_{NP} \ [on \ Saturdays]_{PP}$. The chunker uses the ERTS tagset internally, but if a user requests the RTS tagset, the system will generate that by using an internal mapping process from the ERTS tagset to the RTS tagset. The BPC is very fast and it is reported to yield an F1 measure (harmonic mean) of 0.963.

In the current version of AMIRA2.1, the user can choose among different options to run the systems. These options are: tok-only (run the AMIRA-TOK only), tok+pos (run the AMIRA-TOK and then AMIRA-POS) and all (run all AMIRA toolkit tools as follows: the AMIRA-TOK, the AMIRA-POS, and then the AMIRA-BPC). Table 3.4 shows the output of AMIRA 2.1 for the Arabic sentence in (3.1) using ERTS_PER tagset.
3.3.3.2 MADA

MADA is a highly customizable and freely available toolkit for Arabic NLP applications. It is reported to perform around 97.6% accuracy on the Penn Arabic Treebank. This toolkit consists of two components, namely MADA and TOKAN (Habash et al., 2009). The two components will be explained below.

- **MADA**

MADA (Morphological Analysis and Disambiguation for Arabic) is a tool that adds as much lexical and morphological information as possible to a given raw Arabic text by disambiguating in one operation POS tags, lexemes, diacritizations and full morphological analyses. MADA manipulates the raw Arabic text in stages. First, it solves the problem of morphological analysis by using a list of potential analyses for each word in the text provided by the Buckwalter Arabic Morphological Analyser (BAMA) (Buckwalter, 2004) or the Standard Arabic Morphological Analyser (SAMA) (Maamouri et al., 2010). Then, MADA makes use of 19 orthogonal features to select a proper analysis from the list. These features include 14 Morphological features that MADA predicts using 14 SVMs trained on the Penn Arabic Treebank. The remaining five features are used to capture spelling variation and n-gram statistics. MADA considers a word’s diacritized form, its morphological features, its lexeme and an English glossary entry in each analysis.

---

This is the mostly used resource for Arabic computational linguistics and for instance has been used for training the well-known Arabic taggers e.g. AMIRA and MADA
The possible analyses are ranked by appending a numerical score to each one. MADA sorts these analyses in descending order and the highest scoring analysis for each word in the context is flagged with an asterisk (*). All decisions related to morphological ambiguity, lexical morphological, diacritization and POS tagging in any possible POS tagset are made in one step because MADA takes a full analysis from BAMA or SAMA. Internally, MADA uses three different components to analyse the raw text. These components are: BAMA/SAMA, Standard Research Institute Language Modeling (SRILM) toolkit and SVMTools packages to operate its SVMs.

- **TOKAN**

  This is a tool that works as a tokeniser for Arabic. It can generate a tokenisation (segmentation) formatted to user specifications by manipulating the information MADA produces. It provides an easy-to-use resource for tokenising MADA’s outputs into a broad set of potential tokenisation schemes. For example, MADA decides whether an Arabic word has a preposition clitic or a conjunction whereas TOKAN decides if and how these clitics are separated before using them in an application. Table 3.5 shows the output of MADA3.2 for the Arabic sentence in (3.1).
Table 3.5: MADA3.2 output for the Arabic sentence in (3.1).
3.3.3.3 The Stanford tagger

This is a POS tagger implemented using the RTS tagset. It uses a maximum entropy conditional sequence model in a bidirectional dependency network approach and is therefore able to capture both directions of influence \cite{Toutanova et al. 2003}. It explicitly makes use of both preceding and following tag context and lexical features. So, it picks the best tag given an observation word and its context and the previous tags. It has many trained models for Arabic, Chinese, English, French and German. The tagger is retrainable so given POS-annotated training data for a language, it can be retrained on that language. The tagger is reported to perform around 96.5% accuracy on the Penn Arabic Treebank. Table 3.6 shows the output of the Stanford3.5 tagger for the Arabic sentence in (3.1).

<table>
<thead>
<tr>
<th>Input sentence:</th>
<th>شارك عثرة طلاب في الندوة.</th>
</tr>
</thead>
<tbody>
<tr>
<td>POS</td>
<td>/VBD /CD /NN /IN /NN ./PUNC</td>
</tr>
</tbody>
</table>

Table 3.6: Stanford3.5 output for the Arabic sentence in (3.1).

Table 3.7 shows the three taggers output for the Arabic sentence in (3.1).

<table>
<thead>
<tr>
<th>ARABIC</th>
<th>Gloss</th>
<th>AMIRA</th>
<th>MADA</th>
<th>The Stanford tagger</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Ark</td>
<td>participated</td>
<td>VBD_MS3</td>
<td>verb</td>
<td>VBD</td>
</tr>
<tr>
<td>عثرة</td>
<td>ten</td>
<td>NNCD_FS</td>
<td>noun_num</td>
<td>CD</td>
</tr>
<tr>
<td>طلاب</td>
<td>students</td>
<td>NN</td>
<td>noun</td>
<td>NN</td>
</tr>
<tr>
<td>في</td>
<td>in</td>
<td>IN</td>
<td>prep</td>
<td>IN</td>
</tr>
<tr>
<td>الندوة</td>
<td>the symposium</td>
<td>DET_NN_FS</td>
<td>noun</td>
<td>NN</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>PUNC</td>
<td>punc</td>
<td>PUNC</td>
</tr>
</tbody>
</table>

Table 3.7: POS tags by three taggers for the Arabic sentence in (3.1).

3.3.3.4 Comparison between AMIRA, MADA and the Stanford tagger

There is no optimal POS tagger for Arabic; each tagger tends to target a specific application or a tagset \cite{Habash et al. 2009}. Even the state-of-the-art taggers such as AMIRA, MADA and the Stanford tagger do not cover all aspects for Arabic, but they are general enough to be used
in different NLP applications. Table 3.8 shows a comparison between them. It summarises the characteristics of each tagger and helps to present the differences between them. AMIRA takes a multi-step approach to tokenisation, POS tagging and lemmatisation whereas MADA and the Stanford tagger treat all of these tasks in one fell swoop.

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>AMIRA</th>
<th>MADA</th>
<th>The Stanford tagger</th>
</tr>
</thead>
<tbody>
<tr>
<td>Components</td>
<td>TOK, POS, BPC</td>
<td>MADA, TOKAN</td>
<td>POS</td>
</tr>
<tr>
<td>Technology</td>
<td>SVM, Relies on surface data</td>
<td>SVM, Provides deep analysis</td>
<td>Maximum entropy conditional model</td>
</tr>
<tr>
<td>Steps</td>
<td>Multi-step</td>
<td>One step</td>
<td>One step</td>
</tr>
<tr>
<td>Trainable</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 3.8: Comparison between AMIRA, MADA and the Stanford tagger.

### 3.4 Stemming

Stemming is an essential processing step in a wide range of high level text processing applications such as information extraction, machine translation and sentiment analysis (Habash et al., 2009). It is a non-trivial problem especially with languages that have rich and complex morphology such as Arabic. The function of a stemmer is to reduce words to their stems by stripping off all affixes (Frakes, 1992). Affixes are syntactic units that do not have free forms, but are instead attached to other words. They use the same alphabet as that of words and are concatenated one after the other with no demarcation such as the English apostrophe. Therefore, they are not easily recognisable. Although there have been several studies on developing morphological and stemming tools for Modern Standard Arabic (MSA), a stemmer that can work on Arabic tweets or similar text styles is yet to be developed.

There have been several studies on developing morphological and stemming tools for MSA. These tools can be classified into three categories: manually constructed dictionaries (heavy stemming), light stemming and statistical stemming.
3.4.1 Heavy stemming

Heavy stemming approaches try to find the stems and roots by using dictionaries. There are many examples of recent research work under this category such as Buckwalter (2004), Khoja and Garside (1999), and Algarni et al. (2014). The Buckwalter Arabic Morphological Analyzer (BAMA) is the best known example of such an approach. It is an open-source software package morphological analyser developed by Tim Buckwalter. He developed a set of lexicons of Arabic stems, prefixes, suffixes and a table containing valid morphological combinations in order to produce all possible stems for each word. The Khoja stemmer is another example of this class. It is based on one dictionary for Arabic roots. It removes the longest suffix and the longest prefix. Then, it matches the remaining word with roots in the dictionary and a list of patterns. These approaches produce well-formed stems and roots and are potentially correct, but they generate multiple analyses, therefore they have to have some downstream mechanisms for choosing between them and the dictionaries are extremely difficult to maintain.

3.4.2 Light stemming

The light stemming approach is a process of stripping off a set of affixes from a word without trying to recognise patterns and find roots. It is fast, does not need roots or stems dictionaries and plays safe in order to avoid over-stemming errors, but it may produce a stem that may not even be a real Arabic word (Paice, 1994). Al-Kabi et al. (2015) uses a light stemmer approach but does not cover all affixes in Arabic.

3.4.3 Statistical stemming

The statistical approach is a process of inducing a list of affixes automatically and using clustering techniques to group word variants. Most statistical approaches require annotated training data such as Darwish and Oard (2007) and De Roeck and Al-Fares (2000). This data is
expensive and is not always available. Although there have been some positive results with unannotated training data for many different languages such as [Goldsmith (2001)] and [Das-gupta and Ng (2007)], it seems likely that the complexity of Arabic language would make that kind of approach infeasible.

### 3.5 Syntactic parsing

Parsing natural language, or syntax analysis, is a process in which sentences are examined and analysed automatically to determine their syntactic structure based on the grammatical rules of a given language [Aho and Ullman (1972)]. Parsing is considered to be an important stage in different Natural Language Processing (NLP) applications [Thant et al. (2011)]. Therefore, the performance of such NLP applications will be affected by the performance of parsing solutions they use. For instance, parsing has been used in the following applications:

- **Information Retrieval (IR):** Parsing has been used to improve information retrieval performance in IRs [Metzler and Haas (1989)].

- **Question and Answering (QA):** Sophisticated parsing techniques have been used in most of the high performing QA systems [Lee et al. (2008)].

- **Machine Translation (MT):** To translate phrases efficiently, parsing solutions are often used in MT systems [Zollmann and Venugopal (2006)].

There are three dimensions along which parsing systems can be classified: the type of parse trees they produce, the type of algorithms they use and the information source that the algorithms make use of.
3.5.1 Parsing frameworks

There are two main frameworks for parsing, namely the phrase structure framework and the dependency framework. The output of a parsing framework is a representation, normally a tree, displaying the relations between the constituents of a sentence. In this section, phrase structure parsing and dependency parsing will be discussed. We will use the English sentence in (3.2) to illustrate the difference between the phrase structure and dependency structure.

(3.2) Bad news has huge effect on financial markets.

3.5.1.1 Phrase structure parsing

Phrase structure parsing is a kind of natural language syntactic parsing which is based on Phrase Structure Grammar (PSG). The PSG has very long traditions, with major advances introduced by Noam Chomsky in the late 1950s. It is a technique of describing a natural language by breaking a sentence down into its constituent parts and classifying them by structural categories such as NP, VP, PP, etc. It depends on the phrase structure theory. Functional relations such as subject and object can be identified in phrase structure grammar in terms of structural configurations (e.g. ‘NP’ under ‘S’ and ‘NP’ under ‘VP’) (Kübler et al., 2009). Generally speaking, a phrase structure representation may be found more appropriate for languages with clear constituency structures and fixed word order patterns such as English (Karlsson et al., 1995).
Consider, for instance, the grammar in Figure 3.5

\[
\begin{align*}
S & \rightarrow NP \; VP \; PU \\
NP & \rightarrow JJ \; NN \mid JJ \; NNS \\
VP & \rightarrow VBD \; NP \; PP \\
PP & \rightarrow IN \; NP
\end{align*}
\]

The structure in Figure 3.6 will be assigned to the sentence (3.2) based on the grammar in Figure 3.5.

Figure 3.5: Example of a context-free grammar.

Figure 3.6: Phrase structure tree for the English sentence in (3.2).
3.5.1.2 Dependency parsing

Dependency parsing is a kind of natural language syntactic parsing which is based on Dependency Grammar (DG). DG was introduced by Tesnière (1959). It is a technique of describing a natural language based on the idea that the sentence syntactic structure consists of binary asymmetrical relations between the words of the sentence (Tesnière, 1959). The idea that underlies dependency grammar is that each word in a sentence depends on other words, except one word which is considered as the root. For instance, the sentence ‘a boy plays’ can be simply analysed based on its dependency grammar as shown in (3.3).

(3.3)  
a depends on boy
   
   boy depends on plays
   
   plays depends on nothing (i.e. plays is the root of the sentence)

or, alternatively

a modifies boy
   
   boy is the subject of plays
   
   plays is the matrix verb of the sentence

In general, a dependency representation may be found more suitable for languages which allow greater freedom of word order and in which linearisation is controlled more by pragmatic than by syntactic factors, such as Arabic, Czech and Polish (Karlsson et al., 1995). Dependency parsing models a sentence as a graph (usually a tree). In the tree, vertices represent words, while directed edges represent grammatical functions. Therefore, every vertex has a single parent apart for the one which is the root of the tree. The dependency structure illustrates head-dependent relations between vertices that are classified by dependency types (e.g. SBJ ‘subject’, OBJ ‘object’, ATT ‘attribute’, etc.). Figure 3.7 shows the dependency tree for the sentence in (3.2), where the labels on arrows show the dependency types, while the arrows themselves between pairs of vertices indicate the dependency relations pointing from the head
to dependent.

![Dependency tree for the English sentence (3.2).](image)

Figure 3.7: Dependency tree for the English sentence (3.2). SBJ= Subject, OBJ= Object, ATT= Attribute, PC= Prepositional clausal modifier and PU= Punctuation

### 3.5.1.3 Phrase structure and dependency structure conversion

In both parsing representations mentioned above, the notion of head is important. In a phrase structure, a head has several levels of projection and it determines the main properties of the phrase. For example, Figure 3.6 shows the phrase structure tree for the sentence in (3.2) where the verb VBD projects a verbal phrase VP and makes the phrase a VP while the noun NN projects a noun phrase NP and makes the phrase an NP. This shows the importance of heads in determining the type of phrases.

In a dependency structure, the head is linked to its direct daughter(s). For example, Figure 3.7 shows the dependency tree for the sentence in (3.2) where the arrows between pairs pointing from the head to its dependent(s). Note that the main word ‘has’ in the sentence has been assigned a dummy word ‘root’ because no words in the sentence can be the head of ‘has’. So, the head is a common notion in dependency structures and phrase structures. In practice, the head information is explicitly marked in the former, but not always so in the latter (Xia and Palmer, 2001).

The conversion from phrase structures to dependency structures is a fairly easy task if the
heads in phrase structures are found. Using a Head Percolation Table (HPT) is a common way to find the head in a phrase structure (Collins [1997]; Magerman [1995]). Once the head is identified, a transformation algorithm such as the one discussed by Xia and Palmer (2001) is used to produce dependency structures. Each entry in HPT contains the labels for all subtrees which are headed by a given label in a specific order. Therefore, different organisations of labels in HPT produce different dependency structures.

3.5.2 Parsing algorithms

There are two widely used classes of parsing algorithms, namely grammar-based algorithms and action-based algorithms.

3.5.2.1 Grammar-based parsing

In this kind of algorithm, the information used by the parser is about what combinations of words occur in the language L. This information is used to compute a given sentence analysis. One of the main shortcomings of a grammar-based approach has been to achieve robustness, where robustness is defined as the ability of grammar-based systems to analyse any input sentences because they are not in the language L(G) defined by the grammar G.

Theoretically speaking, there are two problematic cases related to robustness. In the first case, a sentence is a well-formed sentence of the language L but it is not part of L(G). This case can be called the coverage problem, since increasing the grammar coverage should eliminate this problem. In the second case, a sentence is not part of L, and should therefore not be part of L(G), however, it has a sensible syntactic structure. This case is referred to as the robustness problem. For example, if the sentence from Figure 3.6 had contained the word impact or the word effect, instead of the word effect, then the sentence would not have been included in the language defined by the grammar in Figure 3.5. In the first case, it would be a coverage
problem, since impact is an English word, which can play the same structural rule as effect. In the second case, this would be a robustness problem, since effect is not a word of English.

To solve the robustness problem for grammar-based systems, there are two main methods that have been proposed in the literature (Samuelsson and Wiren, 2000). The first method is called partial parsing, where grammar-based systems are used to recover as much structure as possible from well-formed fragments of the sentence by maintaining the constraints of G. In the second method, a complete analysis can be assigned to a sentence outside L(G) by relaxing the grammatical constraints of G.

Within this class, there are many variations concerned with the way the search space is explored—top-down, bottom-up, left-corner, chart, etc.—and what kind of grammar—hand-coded, derived from corpus, context-free, feature-based, etc.

3.5.2.2 Action-based parsing

In action-based parsing, also called transition-based parsing, the information used by the parser is about what to do in a given situation. It uses greedy algorithms to move from one state to the next by simply choosing the most likely next state (Kübler et al., 2009). Many variations on this class are deterministic and efficient, but they have two possible disadvantages which are a lack of backtracking and error propagation because they build a parse by a sequence of actions, scoring each action separately (Duan et al., 2007). To avoid these disadvantages, a variety of search strategies are used such as SyntaxNet which uses a beam search to score each partial solution and find the highest scoring parse tree from all possible analyses (Andor et al., 2016).

3.5.3 Parsing strategies

There are two broad types of strategy for parsing: the rule-driven approach and the data-driven approach (Carroll, 2000). However, many existing methods actually combine elements of both
3.5.3.1 Rule-driven parsing

In this strategy, the parser uses a set of rules which are manually hand-written to produce the analyses. To use rule-driven parsing, a good set of rules has to be available; these are hard to develop, and wide-coverage rules sets often lead to the generation of large numbers of analyses.

3.5.3.2 Data-driven parsing

The information used by the parser in this strategy was extracted from a corpus. The data-driven approach uses an inductive mechanism to define the mapping from input sentences to output analyses by applying it to a training data from the language L to be analysed. Therefore, a formal grammar is no longer an essential element of the parsing system. In order for data-driven systems to be usable in practice, they have to have both a learning method and a parsing method:

**Learning Method:** In this phase, the parameters of the parsing model are estimated by applying the learning method to the training data.

**Parsing Method:** In this phase, the analyses are constructed and scored for individual sentences based on the given parsing model.

The robustness problem is eliminated in most existing data-driven systems since they are defined in such a way that each input sentence is assigned the best analysis. Grammars that are inferred from data produce large numbers of analyses in the same manner as wide-coverage rule-driven grammars. However, the algorithms that exploit such grammars tend to use some kind of best-first strategy and therefore only produce one analysis in practice (Nivre, 2006).

(Nivre, 2006).
Table 3.9: Examples of parsers.

<table>
<thead>
<tr>
<th>Framework</th>
<th>Algorithm</th>
<th>Strategy</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dependency</td>
<td>Grammar-Based</td>
<td>Rule-Driven</td>
<td>Parasite (Ramsay, 1999)</td>
</tr>
<tr>
<td>Parsing</td>
<td></td>
<td></td>
<td>(Tapanainen and Järvinen, 1997)</td>
</tr>
<tr>
<td></td>
<td>Data-Driven</td>
<td></td>
<td>(Foth et al., 2005)</td>
</tr>
<tr>
<td>Action-Based</td>
<td>Grammar-Based</td>
<td>Rule-Driven</td>
<td>(Marcus, 1978)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(Milne, 1980)</td>
</tr>
<tr>
<td></td>
<td>Data-Driven</td>
<td></td>
<td>MALTParser (Nivre et al., 2006)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>McParseface (Andor et al., 2016)</td>
</tr>
<tr>
<td>Phrase Structure</td>
<td>Grammar-Based</td>
<td>Rule-Driven</td>
<td>(Bangalore and Joshi, 1999)</td>
</tr>
<tr>
<td>Parsing</td>
<td></td>
<td></td>
<td>(Joshi and Schabes, 1997)</td>
</tr>
<tr>
<td></td>
<td>Data-Driven</td>
<td></td>
<td>(Collins, 1997)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Stanford (Klein and Manning, 2003)</td>
</tr>
<tr>
<td>Action-Based</td>
<td>Grammar-Based</td>
<td>Rule-Driven</td>
<td>(Ades and Steedman, 1982)</td>
</tr>
<tr>
<td></td>
<td>Data-Driven</td>
<td></td>
<td>(Bos et al., 2004)</td>
</tr>
</tbody>
</table>

The range of frameworks, parsing algorithms and parsing strategies outlined above means that there are numerous ways in which these elements can be combined, as shown in Table 3.9. Note that of the eight possible combinations, virtually all have been instantiated. Of the examples in Table 3.9, we are aware that versions of the Parasite, MALTParser and Stanford parsers have been used for processing Arabic.

3.5.4 Parsers for Arabic

In this section, we will discuss two parsers used in the literature to parse Arabic text, namely Parasite and MALTParser. We will be using versions of (Parasite$^+$ and MALTParser$^+$) for analysing Arabic tweets in Chapter 7.

3.5.4.1 Parasite

Parasite is a Prolog program which is used to perform a range of linguistics tasks including a parser for analysing the linguistic structure for a number of languages (Ramsay, 1999). It uses a general syntactic framework which lies somewhere between Head-driven Phrase Structure
Grammar (HPSG) and pure Categorial Grammar (CG). HPSG is a lexical grammar which uses the notion of SIGNS. A sign is an object which describes all information you know about some word or phrase. CG is equivalent to a context-free grammar and it allows only for one way for items to combine. Parasite’s signs have five components, so that a basic sign has the following shape:

```
sign(structure(...),
    morphology(...),
    syntax(...),
    meaning(...),
    remarks(...))
```

- **structure** contains details about the words and phrases that this sign is made up from.
- **morphology** contains information required for combining sublexical items to make words.
- **syntax** contains values for features called **args** and **target**. The **args** is a list of items that the item subcategorises for, with the first item on this list annotated as **dir=left** or **dir=right** to indicate whether it should be found to the left or the right. The process of combining an item and its arguments is referred to as saturating it. The target describes those items, if any, which the current one is capable of modifying. The two main processes underlying parsing within this framework are combining a modifier and its target and combining an item and its arguments. The combination processes are considered as being essentially like the rules of categorial grammar.
- **meaning** contains the semantic analysis.
- **remarks** contains information that might be useful for some application tasks.
The parser is a chart parser. It allows simple representations of alternative analyses of phrases which give them the same external syntactic characteristics but which are internally different. It stops if it finds a complete analysis of a sentence or it returns the largest non-overlapping potential fragments if does not find a complete analysis after a predetermined number of edges have been created. It uses a just-in-time constraint which allows the parser to delay making decisions until the required information is obtained to avoid performing large amounts of backtracking.

3.5.4.2 MALTParser

Models and Algorithms for Language Technology Parser (MALTParser)\footnote{Freely available at:http://www.maltparser.org/} is a language-independent system which is used for processing natural languages. It is a state-of-the-art dependency parser that has been successfully applied to a range of different languages such as English, Turkish, Bulgarian, etc. \cite{Bengoetxea2010}. It is a data-driven dependency parser and does not use any grammatical rules when analysing the language syntactic structure. It uses an annotated dependency treebank of the language in question to induce a parsing model. This induced parsing model is used to parse new data for the same language \cite{Nivre2007}.

MALTParser is a transition-based parser \cite{Kubler2009}. It transitions through abstract machine states to produce dependency trees. It uses its current state, a history of parsing decisions and the input sentence to learn models. The parser starts with an initial state and based on the prediction of the history and feature models, it moves to subsequent states until a termination state is reached. There are two different models to run MALTParser:

**Learning Mode:** In this mode, the parser takes a training set of sentences with dependency graph annotations and induces a parsing model that can be used to parse new sentences.

**Parsing Mode:** In parsing mode, the parser takes a set of sentences and a previously induced
parsing model to derive the optimal dependency graph for each sentence based on the given parsing model.

MALTParser performs best on dependencies that are further from the root of a tree (e.g., nouns and pronouns dependencies) and those with shorter dependencies (Nivre et al., 2007). MALTParser can also be turned into a phrase structure parser to parse phrases with both phrase labels and grammatical functions (Hall and Nivre, 2008).

### 3.6 Summary

In this chapter, we have described POS tagging for Arabic, including different POS tagsets and taggers. We have also described different techniques for stemming Arabic. Then, we discussed different syntactic parsing frameworks, algorithms and strategies. While much progress has been made on different approaches to Arabic NLP tasks such as POS tagging, stemming and syntactic parsing, many obstacles in processing informal Arabic texts remain before these problems can be said to be solved as will we see in the coming chapters.

In Chapter 4 and Chapter 5, we explore ways of exploiting the taggers discussed in this chapter to deal with Arabic tweets. In Chapter 6, we look at effectiveness of heavy and light stemming for this materials. In Chapter 7, we explore the use of Parasite and MALTParser for constructing and exploiting a silver treebank for such texts.
Chapter 4

Analysis of Existing POS Tagging on Arabic Tweets

In this chapter, we explain the development of our research corpus, covering its collection, annotation and inter-annotator agreement estimation. Then, we use a dataset from this corpus to evaluate three state-of-the-art POS taggers for Arabic, namely AMIRA, MADA and the Stanford tagger and highlight their limitations when manipulating Arabic tweets (RO2). The work in this chapter of the thesis has appeared in Albogamy and Ramsay (2015a,b).

4.1 Introduction

The last few years have seen an enormous growth in the use of social networking platforms such as Twitter in the Arab world. A study prepared and published by Semiocast\(^1\) in 2012 has revealed that Arabic was the fastest growing language on Twitter in 2011. Millions of tweets daily are posted about peoples lives, their opinions on a variety of subjects, on current affairs. The output this yields is noisy and informal although it does contain some informative

\(^{1}\text{http://semiocast.com/en/publications/2011_11_24_Arabic_highest_growth_on_Twitter}
material. It has led to Twitter’s current status as one of the most important social information platforms. Tweets are short, with a maximum of 140 characters, are not always grammatically correct and do not always stick to standard spelling, can be ambiguous in meaning and rich in slang, acronyms and abbreviations, the latter being often used to overcome length restrictions (see Section 2.4 for more details).

POS tagging is an essential processing step in a wide range of high level text processing applications such as information extraction, machine translation and sentiment analysis (Barbosa and Feng, 2010). However, people working on Arabic tweets have tended to concentrate on low level lexical relations used for shallow parsing and sentiment analysis such as Mourad and Darwish (2013) and El-Fishawy et al. (2014). They do not use the standard linguistic pipeline tools such as POS tagging which might enable a richer linguistic analysis (Gimpel et al., 2011). The properties listed in Section 2.4 of the microblogging genre make POS tagging on Twitter very different from their counterparts in more formal texts. It is an open question as to how well the features and techniques of NLP used on more well-formed data (e.g. in the newswire) will transfer to Twitter in order to build systems that can understand and exploit tweets. In this chapter, we experimentally evaluate the performance of state-of-the-art POS taggers for MSA on Arabic tweets. We also analyse their limitations, identify problem areas in tagging Arabic tweets and explore the causes of the majority of errors.

4.2 Experiments

We evaluate three state-of-the-art publicly available POS taggers for Arabic, namely AMIRA (Section 3.3.3.1), MADA (Section 3.3.3.2) and the Stanford tagger (Section 3.3.3.3).
CHAPTER 4. ANALYSIS OF EXISTING POS TAGGING ON ARABIC TWEETS

4.2.1 Data collection

There is a growing interest within the NLP community to build Arabic social media corpora by harvesting the web (Abdul-Mageed et al., 2012; Refae and Rieser, 2014). However, none of these resources are publicly available yet. Neither do they contain all the tweets phenomena as they appear in their original forms in Twitter, and they have been constructed primarily to be used in sentiment analysis. Hence, we built our own corpus which preserves all phenomena of Arabic tweets. We used the Twitter Stream API\(^2\) to retrieve tweets from the Arabian Peninsula by using latitude and longitude coordinates, since Arabic dialects in these regions share similar characteristics and are the closest to MSA. We did not restrict tweet language to ‘Arabic’ in the query since users may use other character sets such as English to write their Arabic tweets (Romanisation) or they may mix Arabic script with another language in the same tweets. Next, we excluded all tweets which were written completely in English. Our corpus consists of a million Arabic tweets (10 million tokens).

4.2.2 Annotation

A set of correctly annotated Arabic tweets (gold standard) is required in order to be able to appraise the outputs of POS taggers. Once we have this, we can compare the outputs of the POS taggers with this gold standard. Since there is no publicly available annotated corpus for Arabic tweets, we have created a POS-tagged corpus (hereafter MainCorpus) with specific POS tags for Twitter phenomena (i.e. REP, MEN, HASH, LINK, USERN, RET, EMOT and EMOJ for replies, mentions, hashtags, links, usernames, retweets, emoticons and emojis respectively) (see Table 4.1). We sampled 390 tweets (5454 words) from MainCorpus to be used in our experiments (hereafter Corpus 1) and we manually annotated our dataset (similar studies for English tweets also use a few hundred tweets, e.g. (Gimpel et al., 2011)). The manual analysis

---

\(^2\)We call Twitter Stream API by using Java as shown in Appendix D.
indicated that only a small sub-set of these are non-Arabic (i.e. 1.2% is mixed Arabic and English, 0.5% is mixed Arabic and Romanisation, 8% Twitter-specific, 1.1% Emoji and 0.2% Emoticons) and the large majority of the tweets is in Arabic (89%) (see Section 4.2.6 for more details). To speed up manual annotation, we tagged tweets by using the taggers. However, these taggers use different tagsets so to construct a gold standard we mapped their tagsets to a unified tagset, and then we corrected the mistagged tokens (see Section 4.2.3). This dataset is used in Chapter 5 and Chapter 6 for developing and evaluating our tagging and stemming tools.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>REP</td>
<td>Reply is a tweet that begins with another user’s username and is in reply to one of their Tweets</td>
<td>... كي ما @SaraSoso987163</td>
</tr>
</tbody>
</table>
| MEN | At-Mention containing another user’s Twitter username and it can occur anywhere in the Tweet except the beginning | @7aamed1 ...
| RET | Retweet (re-sends another user’s tweet) | RT @ReNgo_Sport:...
| HASH | Hashtag is used to mark keywords or topics in a Tweet | ... خديمة البرامج الفري...
| LINK | Link or email address which refers to web pages or email address | صورة http://t.co/vY0feFK3F2 |
| EMOT | It is used by users to express their feelings or emotions in tweets. It is constructed by using alphabetic characters or punctuation | ☺: ... |
| EMOJ | It is used by users to express their feelings or emotions in tweets. It is constructed by using symbols provided in software as small pictures in line with the text | 💖 |

Table 4.1: The tagset used to annotate Twitter phenomena.

### 4.2.3 Tagset unification

The taggers use different tagsets. AMIRA uses the RTS tagset which consists of 25 tags and Stanford uses the same tagset with extra tags to represent the determiner ‘Al’. So, we reduce the Stanford tagger tagset to the AMIRA tagset by omitting determiner tags. On the other hand, MADA\(^3\) has 34 different tags which make different types of distinctions with the RTS tagset. For example, the RTS tagset uses one tag (RP) to cover a range of particles which are subdivided into nine subclasses by MADA (part_det, part_focus, part_fut, ...

\(^3\)We use MADA3.2 tagset, see Appendix C for the older version.
part_interrog, part_neg, part_restrict, part_verb, part_voc, part); and it uses several tags to distinguish between verbs tenses (VBD, VBG, VBN, VBP) where MADA just uses one tag (verb). Therefore, mapping between these tagsets is a prerequisite for using an agreement-based method on the tagger outputs. We construct a unified tagset consisting of the main POS tags as shown in Table 4.2 to do the mapping.
<table>
<thead>
<tr>
<th>AMIRA/Stanford</th>
<th>MADA</th>
<th>Collapsed Tagset</th>
<th>Gloss</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>conj</td>
<td>CC</td>
<td>coordinating conjunction</td>
</tr>
<tr>
<td>CD</td>
<td>digit</td>
<td>CD</td>
<td>cardinal number</td>
</tr>
<tr>
<td>DT</td>
<td>pron_det</td>
<td>DT</td>
<td>demonstrative pronoun</td>
</tr>
<tr>
<td>IN</td>
<td>prep</td>
<td>IN</td>
<td>subordinating conjunction or preposition</td>
</tr>
<tr>
<td>JJ</td>
<td>adj</td>
<td>JJ</td>
<td>adjective</td>
</tr>
<tr>
<td></td>
<td>adj_comp</td>
<td>JJ</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adj_num</td>
<td>JJ</td>
<td></td>
</tr>
<tr>
<td>NN/ NNS</td>
<td>noun</td>
<td>NN</td>
<td>common noun</td>
</tr>
<tr>
<td></td>
<td>noun_num</td>
<td>NN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>noun_quant</td>
<td>NN</td>
<td></td>
</tr>
<tr>
<td>NNP/NNPS</td>
<td>noun_prop</td>
<td>NNP</td>
<td>proper noun</td>
</tr>
<tr>
<td>PRP</td>
<td>pron</td>
<td>PRP</td>
<td>personal pronoun</td>
</tr>
<tr>
<td></td>
<td>pron_exclam</td>
<td>PRP</td>
<td></td>
</tr>
<tr>
<td>PUNC</td>
<td>punc</td>
<td>PUNC</td>
<td>punctuation</td>
</tr>
<tr>
<td>RB</td>
<td>adv</td>
<td>RB</td>
<td>adverb</td>
</tr>
<tr>
<td></td>
<td>part</td>
<td>RP</td>
<td>particle</td>
</tr>
<tr>
<td></td>
<td>part_det</td>
<td>RP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>part_focus</td>
<td>RP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>part_fut</td>
<td>RP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>part_interrog</td>
<td>RP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>part_neg</td>
<td>RP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>part_restrict</td>
<td>RP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>part_verb</td>
<td>RP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>part_voc</td>
<td>RP</td>
<td></td>
</tr>
<tr>
<td>UH</td>
<td>interj</td>
<td>UH</td>
<td>interjection</td>
</tr>
<tr>
<td>VBD</td>
<td>verb</td>
<td>VB</td>
<td>verb</td>
</tr>
<tr>
<td>VBG</td>
<td>verb</td>
<td>VB</td>
<td>verb</td>
</tr>
<tr>
<td>VBN</td>
<td>verb</td>
<td>VB</td>
<td>verb</td>
</tr>
<tr>
<td>VBP</td>
<td>verb</td>
<td>VB</td>
<td>verb</td>
</tr>
<tr>
<td>WP</td>
<td>pron_rel</td>
<td>WP</td>
<td>relative pronoun</td>
</tr>
<tr>
<td>WRB</td>
<td>adv_rel</td>
<td>WRB</td>
<td>wh-adverb</td>
</tr>
</tbody>
</table>

Table 4.2: Collapsed POS tagset.

### 4.2.4 Inter-annotator agreement

Inter-annotator agreement is carried out during annotation tasks to assess levels of bias, consistency and reliability of the annotated data. To estimate the inter-annotator agreement for POS
tagging, we chose 40 Arabic tweets, which an independent annotator\(^4\) tagged from scratch. We achieved an agreement rate (i.e. the percentage of tokens on which the annotators agreed) of 90.5%. This inter-annotator agreement result is comparable to those on the English, Dutch and Irish task for tweets (where the inter-annotator agreement were 92.2%, 92.06% and 90% respectively compared to 90.5% for Arabic tweets) (Avontuur et al., 2012; Gimpel et al., 2011; Lynn et al., 2015). This suggests that the maximum achievable accuracy for POS tagging is around 90%—it is not possible for such a tool to agree with both our annotators with greater accuracy than that.

### 4.2.5 POS tagging comparison

To evaluate the performance of the POS taggers, we use four well-known evaluation measures (Manning and Schütze, 1999):

1. **Precision (P)**
   
   This is used to measure the ability of a system to show only correctly tagged items as in equation 4.1.
   
   \[
   P = \frac{\text{number of correctly tagged items retrieved}}{\text{total number of items retrieved}} \quad (4.1)
   \]

2. **Recall (R)**

   This is used to measure the ability of a system to show all correctly tagged items as in equation 4.2.

   \[
   R = \frac{\text{number of correctly tagged items retrieved}}{\text{total number of items in the collection}} \quad (4.2)
   \]

3. **F-score**

   F-score (or F\(_1\)) is the harmonic mean of precision and recall as in equation 4.3. It reaches its best value at 1 and worst value at 0.

---

\(^4\)He is a postgraduate native speaker student studying engineering at the University of Manchester.
\[ F - score = 2 \times \frac{P \times R}{P + R} \] (4.3)

4. Accuracy

If every word in a collection is assigned a POS tag, then the value of precision is equal to the value of recall because the total number of items retrieved equals to the total number of items in the collection, i.e. formulae 4.1 and 4.2 will produce the same result. In this case, this value is called accuracy in the common practice.

We compare three taggers on 390 tweets (5454 words) from our corpus. The performance of these taggers are computed by comparing the output of each tagger against the manually corrected gold standard using the unified tagset. The results for the AMIRA, MADA and the Stanford tagger which were trained on newswire text present poor success rates\(^5\) for example, the accuracy for AMIRA, MADA and the Stanford tagger on Arabic tweets are 60.2%, 65.8% and 49.0% respectively (see Table 4.3). These figures are far below the performance of the same taggers on well-formed genres such as the Penn Arabic Treebank, where accuracy is around 96% for AMIRA and Stanford whereas MADA achieves over 97% accuracy. This huge drop in the accuracy of these taggers when applied to Arabic tweets warrants some analysis of the problem and of mistagged cases.

<table>
<thead>
<tr>
<th>Tagger</th>
<th>Newswire</th>
<th>Arabic Tweets</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMIRA</td>
<td>96.0%</td>
<td>60.2%</td>
</tr>
<tr>
<td>MADA</td>
<td>97.0%</td>
<td>65.8%</td>
</tr>
<tr>
<td>Stanford</td>
<td>96.5%</td>
<td>49.0%</td>
</tr>
</tbody>
</table>

Table 4.3: POS tagging performance comparison.

4.2.6 Error analysis

We noticed that most of the mistagged tokens are tagged as nouns. The taggers typically tag unknown words as nouns because that is the commonest POS tag and we suspect that a lot

\(^5\)As noted in Section 4.2.6, AMIRA and MADA omit emojis and therefore the scores should be given as precision and recall but for simplicity we mark omitted emojis as being incorrectly tagged.
of these mistakes are words that are not in the training corpus. In this case, the taggers rely on contextual clues such as the morphology of the word and its sentential context to assign them the most appropriate POS tags (Foster et al., 2011). We identified the unknown words that were mistagged and classified them into three groups: Arabic words, Twitter-specifics and non-Arabic tokens. Table 4.4 shows the baseline performance of the taggers in each category (see Table 4.5 for more details. This table shows the percentage of the overall errors that fall into each class and the accuracy on that class).

**Arabic words** These are words which are written in Arabic, but which were assigned incorrect POS tags by the taggers. This category represents 73.5%, 68.1% and 73.2% of the total of mistagged items by AMIRA, MADA and the Stanford tagger respectively. We observed that words in this category have different characteristics and most of them are twitter phenomena. So, we classify them into subcategories as follows:

- **MSA words** These are proper words which are used in well-formed text and part of MSA vocabulary, but which were assigned incorrect POS tags by the taggers. We observed that the accuracy of MSA words which are not noisy dropped from 96% for AMIRA, 97% for MADA and 96.5% for Stanford on newswire genre to 71.8%, 79.3% and 55% respectively on Arabic tweets. There are three possible reasons for this: 1) the context of MSA words are noisy, 2) changes in text structure, for example many function words are omitted in tweets and 3) the domain change between the Arabic treebank corpus on which they were trained and tested and the Arabic tweets. For example, the word ‘النصية’

<table>
<thead>
<tr>
<th>Tokens</th>
<th>AMIRA</th>
<th>MADA</th>
<th>The Stanford tagger</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic words</td>
<td>67.1%</td>
<td>73.8%</td>
<td>54.0%</td>
</tr>
<tr>
<td>Twitter-specific</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
</tr>
<tr>
<td>Non-Arabic</td>
<td>19.9%</td>
<td>9.1%</td>
<td>14.0%</td>
</tr>
<tr>
<td>Overall</td>
<td>60.2%</td>
<td>65.8%</td>
<td>49.0%</td>
</tr>
</tbody>
</table>

Table 4.4: POS tagging accuracy on Arabic Tweets for baseline taggers categorised by tokens.
CHAPTER 4. ANALYSIS OF EXISTING POS TAGGING ON ARABIC TWEETS

<table>
<thead>
<tr>
<th>Tagger</th>
<th>AMIRA</th>
<th></th>
<th>MADA</th>
<th></th>
<th>The Stanford tagger</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Misstagged items</td>
<td>Errors</td>
<td>Accuracy</td>
<td>Errors</td>
<td>Accuracy</td>
<td>Errors</td>
<td>Accuracy</td>
</tr>
<tr>
<td>MSA words</td>
<td>53.3%</td>
<td>71.8%</td>
<td>45.5%</td>
<td>79.3%</td>
<td>56.8%</td>
<td>55%</td>
</tr>
<tr>
<td>Concatenation</td>
<td>1.8%</td>
<td>0%</td>
<td>2.1%</td>
<td>0%</td>
<td>1.6%</td>
<td>0%</td>
</tr>
<tr>
<td>Repeated letters</td>
<td>0.8%</td>
<td>40%</td>
<td>0.8%</td>
<td>50%</td>
<td>1.6%</td>
<td>15.2%</td>
</tr>
<tr>
<td>Named Entities</td>
<td>8.7%</td>
<td>49.2%</td>
<td>8.5%</td>
<td>57.0%</td>
<td>3.7%</td>
<td>66.8%</td>
</tr>
<tr>
<td>Spelling mistakes</td>
<td>0.6%</td>
<td>35%</td>
<td>0.6%</td>
<td>40%</td>
<td>0.7%</td>
<td>27.3%</td>
</tr>
<tr>
<td>Slang</td>
<td>6.2%</td>
<td>30.4%</td>
<td>7.1%</td>
<td>32.0%</td>
<td>7.4%</td>
<td>15.9%</td>
</tr>
<tr>
<td>Characters deletion</td>
<td>0.9%</td>
<td>16.7%</td>
<td>1.0%</td>
<td>20.8%</td>
<td>0.5%</td>
<td>35.8%</td>
</tr>
<tr>
<td>Transliteration</td>
<td>1.2%</td>
<td>61.8%</td>
<td>2.4%</td>
<td>35.3%</td>
<td>0.9%</td>
<td>48.8%</td>
</tr>
<tr>
<td>Romanisation</td>
<td>1.0%</td>
<td>21.4%</td>
<td>1.4%</td>
<td>7.1%</td>
<td>0.9%</td>
<td>25.0%</td>
</tr>
<tr>
<td>Emoticons</td>
<td>0.5%</td>
<td>0%</td>
<td>0.5%</td>
<td>0.0%</td>
<td>0.3%</td>
<td>0.0%</td>
</tr>
<tr>
<td>Emojis</td>
<td>2.8%</td>
<td>0%</td>
<td>3.3%</td>
<td>0.0%</td>
<td>2.7%</td>
<td>0.0%</td>
</tr>
<tr>
<td>Foreign words</td>
<td>2.6%</td>
<td>35.6%</td>
<td>3.9%</td>
<td>17.2%</td>
<td>2.8%</td>
<td>11.7%</td>
</tr>
<tr>
<td>Twitter specific</td>
<td>19.6%</td>
<td>0%</td>
<td>22.8%</td>
<td>0%</td>
<td>20.2%</td>
<td>0%</td>
</tr>
</tbody>
</table>

Table 4.5: Errors percentage of each misstagged class and its accuracy.

(disobey) was tagged *NN* by AMIRA, *noun* by MADA and *NNP* by the Stanford tagger but, in fact, it is a verb.

- **Concatenation** In this classification, two or more words were connected to each other to form one token. So, the taggers struggled to label them. Users may connect words deliberately to overcome tweets’ restricted length, or accidentally. In this experiment, the taggers mistagged all connected words in the subset. For example, the word ‘لا’ was tagged *NN* by AMIRA, tagged *noun* by MADA and tagged *NNP* by the Stanford tagger. But, in fact, it is two words ‘لا’ and ‘أ’ connected together which are a verb and a complementiser respectively.

- **Repeated letters** Words in this classification have one or more letters repeated. Users repeat letters deliberately to express subjectivity and sentiment. For example, the word ‘standing’ was labelled *NNS* by AMIRA and Stanford and *noun* by MADA but, in fact, it is an adjective.
CHAPTER 4. ANALYSIS OF EXISTING POS TAGGING ON ARABIC TWEETS

- **Named entities** All these words should be labelled proper noun by the taggers because they refer to a person, place or organization, but they were mistagged because these words were not part of the tagger’s training data. For example, the proper noun ‘السـيد’ was tagged *NN* by AMIRA and the Stanford tagger and labelled *noun* by MADA. In some languages, there is information in the text that helps systems to recognise named entities, e.g., capital letters in English. However, in Arabic, the taggers rely on dictionaries to recognise named entities. So, this task becomes more difficult within an open vocabulary genre.

- **Spelling mistakes** It is not easy to know the intent of the user, but some words seem likely to have been accidentally misspelled. Most words belonging to this category were mistagged by the taggers. For example, the word ‘العـديد’ was misspelled and it should be written as ‘العـدید’ (abounded). AMIRA and the Stanford tagger tagged it *NN* and MADA labelled it *noun* but, in fact, it is a verb.

- **Slang** The use of slang is a widespread Twitter phenomenon. The words in this category are regarded as informal and are typically restricted to a particular context or group of people. They are often mistagged by the taggers. For example, the slang word ‘العـديد’ is the counterpart of the MSA word ‘العـديد’ which means *Look!*

- **Character deletion** Arabic users delete letters from words deliberately to overcome tweets’ restricted length or because they do not have enough time to write complete words. For example, the word ‘العـديد’ (at) was shortened to only one letter ‘العـديد’. This word was tagged *PUNC* by AMIRA, *conj* by MADA and *CC* by the Stanford tagger but, in fact, it is a preposition.

- **Transliteration** Arabic users borrow some words and multiword abbreviations from English. They use their Arabic transliteration in Arabic tweets. For example, LOL in En-
glish (Laugh Out Loud) is written in Arabic as ‘أخر’ and ‘mix’ in English is written in Arabic as ‘مختلط’. AMIRA and the Stanford tagger tagged the translated form of mix as NN whereas MADA labelled them all as noun but, in fact, it is a verb.

**Twitter-specific elements** These are elements unique to Twitter, such as reply, mention, retweet, hashtag and url. They represent 19.6%, 22.8% and 20.2% of the total of mistagged items by AMIRA, MADA and the Stanford tagger respectively. In fact, taggers mistagged all Twitter-specific elements in the experiment and they tokenised them in different ways. AMIRA uses punctuation as an indicator for a new token so replies, mentions, retweets and hashtags in tweets are broken into the indicator part (@ for replies, mentions and retweets and # for hashtags) and the remainder of them. Moreover, if the remainder part contains punctuation marks, AMIRA will split it further into parts. AMIRA also breaks URLs into parts since they contain punctuation marks. In contrast, MADA and the Stanford tagger do not break all Twitter-specific elements into parts since they use the space as an indicator for a new token. MADA has one exception to this rule. If a hashtag started with an Arabic letter, then MADA breaks it into parts when punctuation is found. We notice that MADA always labels unsplit Twitter-specific elements as nouns noun (see Table 4.6).

<table>
<thead>
<tr>
<th>AMIRA</th>
<th>MADA/The Stanford tagger</th>
</tr>
</thead>
<tbody>
<tr>
<td>Twitter element</td>
<td>Token</td>
</tr>
<tr>
<td>@Moh_Ali</td>
<td>@</td>
</tr>
<tr>
<td>Moh</td>
<td>NN</td>
</tr>
<tr>
<td>-</td>
<td>FUNC</td>
</tr>
<tr>
<td>Ali</td>
<td>NN</td>
</tr>
</tbody>
</table>

Table 4.6: Example of Twitter element tokenised and tagged by taggers.

**Non-Arabic tokens** This group contains the remaining twitter phenomena which appear in Arabic tweets, but which are not written by using the Arabic alphabet. They represent 6.9%, 9.1% and 9% of the total of mistagged items by AMIRA, MADA and the Stanford tagger respectively. We classify them into subcategories based on their shared characteristics as follows:
• **Romanisation** Arabic users tend to use Latin letters and Arabic numerals to write Arabic tweets because the actual Arabic alphabet is unavailable for technical reasons, difficult to use or they speak Arabic but they cannot write Arabic script. For example, the word 3ala which is the Romanised form of the Arabic word ‘نا’ was tagged *NN* by AMIRA, labelled *noun* by MADA and *CD* by the Stanford tagger but, in fact, it is a preposition.

• **Emoticons** They are constructed by using traditional alphabetic characters or punctuation, usually a facial expression. They are used to express feelings or emotions in tweets. AMIRA and MADA break emoticons into parts during the tokenisation processes and they deal with each part as punctuation so all emoticons lose their meaning. For example, the emoticon (= was broken into two parts: ‘(’ (labelled *PUNC*) and ‘=’ (labelled *PUNC*). In contrast, the Stanford tagger does not break them into parts but it has mistagged all of them.

• **Untagged emoji** Emoji means symbols provided in software as small pictures in line with the text which are used by users to express their feelings or emotions in tweets. AMIRA and MADA omitted these symbols in the tokenisation stage and they did not tag them. For example, the heart symbol ♥ was omitted when tweets were tokenised by these taggers. By contrast, the Stanford tagger does not omit them but it mistagged all of them.

• **Foreign words** Some Arabic tweets contain foreign words especially from English. These words may refer to events, locations, English hashtags or retweet of English tweets with comments written in Arabic. The tweet “I’m at Arab Bank في البنك العربي” is an example of this category. AMIRA and the Stanford tagger tagged foreign words in this tweet as ‘I’m’ is a *VBD*, ‘at’ is a *PUNC*, ‘Arab’ is a *NN* and ‘Bank’ as *NN* whereas MADA labelled them all as *noun*. 
CHAPTER 4. ANALYSIS OF EXISTING POS TAGGING ON ARABIC TWEETS

4.3 Summary

In this chapter, we have examined the consequences of applying MSA-trained POS tagging to Arabic tweets by means of experiments. Off-the-shelf taggers present poor success rates on Arabic tweets due to the genre noisiness. Furthermore, existing approaches suffer from insufficient labelled training data. They also fail to deal with Twitter phenomena as shown in the detailed error analysis. Therefore, the outcomes of standard POS taggers such as AMIRA, MADA and the Stanford tagger when applied to the Arabic tweets genre are not useful as inputs to any parser.

In the next chapter, we will discuss our approaches to making these taggers robust towards noise and to developing a fast POS tagger for Arabic tweets by taking into consideration Arabic tweets characteristics and tackling most of the errors mentioned in this chapter.
Chapter 5

POS Tagging for Arabic Tweets

In this chapter, we describe in detail our approaches for improving the accuracy of tagging Arabic tweets (RO3). We outline the techniques of using pre- and post-processing to the tagging. Next, we present a bootstrapping approach on unlabelled Arabic tweets to create a sufficient amount to train the augmented version of the Stanford tagger on it. The trained tagger produces a corpus with a coarse-grained tagset. The whitened boxes in Figure 5.1 show where the work in this chapter fits into the overall NLP pipeline. The material in this chapter has been published in Albogamy and Ramsay (2016a).

5.1 Introduction

As seen in Chapter 4, our experiments show that the standard taggers present poor success rates since they were trained on newswire text and designed to deal with MSA text. They fail to deal with Twitter phenomena. As a result, their outcomes are not useful for linguistics downstream processing applications such as information extraction and machine translation in the microblogging genre and there is a direct correlation between the accuracy of parsers and the accuracy of POS taggers (Croce and Basili, 2012; Foster et al., 2011). A substantial amount of syntactic parsing errors is due to the propagation of POS tagging errors. Therefore,
Figure 5.1: The NLP pipeline adapted for Arabic tweets (tagging). White boxes are discussed in this Chapter.
there is a need for a POS tagger which should take into consideration the characteristics of Arabic tweets and yield acceptable results.

Our goal is not to build a new POS tagger for Arabic tweets. The goal is to make existing POS taggers for MSA robust towards noise. There are two ways to do so: one is to retrain POS taggers on Arabic tweets and alter their implementation if needed; the other is to overcome noise through applying pre- and post-processing to the tagging. Our approach is based on both approaches. We combine normalisation and external knowledge to boost the taggers’ performance. Then, we retrain the Stanford tagger on Arabic tweets since its speed is ideal for the tweets genre and it is the only one of the three which is retrainable. However, we do not have suitable labelled training data to undertake this. Therefore, we use bootstrapping on unlabelled data to create a sufficient amount of labelled training tweets. By using this approach, we are able to develop a fast and robust tagger for Arabic tweets.

5.2 Pre- and post-processing

As seen in error analysis (Section 4.2.6), unknown words (out-of-vocabulary tokens or OOV) represent a large proportion of mistagged tokens. We argue that normalisation and external knowledge will reduce this proportion and hence improve the performance of the proposed tagger. Normalisation is the process of providing in-vocabulary (IV) versions of OOV words (Han and Baldwin 2011). We create a mapping from OOV tokens to their IV equivalents by using suitable dictionaries and the original token is replaced with its corresponding IV token. External sources of knowledge such as regular expression rules, gazetteer lists and the output of an English tagger are also used. The combination of normalisation and external knowledge is applied to text as pre- and post-processing step.\footnote{The code is included as Appendix E}
5.2.1 Pre-processing

- **Handling Concatenation** Users may connect words deliberately, to overcome length restrictions of tweets, or accidentally. This forms tokens which all taggers struggle to tag correctly. One approach towards dealing with these cases is to use an MSA dictionary. We constructed an MSA dictionary from 250k Arabic words which were extracted from news websites.

We handle concatenation for a word $W$ in the corpus as follows:

1. If the length of $W$ is $\leq 5$, then it is left as it is, since the average length of Arabic words is five letters (Mustafa, 2012).
2. Else, if $W$ exists in the MSA dictionary, then it is left as it is, since it is a valid MSA word.
3. Else, if a part $P$ of $W$ exists in the MSA dictionary, then $W$ is split into two parts (all parts must be in the MSA dictionary), $P$ and the remainder and the same steps are applied to the remainder.

We apply the above algorithm on ‘@Y»AK’. The length of this token is six characters, which is larger than the average length of Arabic words, so we check if it exists in the MSA dictionary, but it does not exist in the dictionary. Then we check if any part of it exists in the dictionary, we find ‘Y»AK’ in the dictionary so we split the token into two parts ‘Y»AK’ and the remaining characters and then we apply the algorithm to the second part. Because the length of the second part ‘@’ is two characters, it is left as it is and the algorithm stops.

By using this approach, we improve concatenation words tagging accuracy as shown in Table 5.1. This approach could potentially introduce errors because it could lead to early cliticisation. However, this does not happen because every part must be found in the

\[^2\text{http://sourceforge.net/projects/ar-text-mining/files/Arabic-Corpora/}\]
dictionary.

<table>
<thead>
<tr>
<th>Concatenation</th>
<th>AMIRA Before</th>
<th>AMIRA After</th>
<th>MADA Before</th>
<th>MADA After</th>
<th>The Stanford tagger Before</th>
<th>The Stanford tagger After</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.0%</td>
<td>25%</td>
<td>0.0%</td>
<td>25%</td>
<td>0.0%</td>
<td>25%</td>
</tr>
</tbody>
</table>

Table 5.1: The impact on accuracy on concatenated words before and after dealing with concatenation as a pre-processing step.

- **Handling Elongated Words** We handle these cases by using the same MSA dictionary mentioned above. Given a word \( W \) in the corpus, we undertake the following steps:

  1. If a word \( W \) exists in the MSA dictionary, then it is left as it is, even if it contains repeated letters.
  2. Else, a compressed form of it is constructed by removing any repetition in letters.

As shown in Table 5.2, the first two tokens do not exist in the dictionary. So, they are replaced by their compressed forms. The third token has repeated letters, but it exists in the dictionary so it is left as it is. By using this approach, we are able to improve the accuracy of elongated words tagging as shown in Table 5.3. This step can introduce errors because the dictionary that we used cannot contain all MSA words that have repeated characters. For example, the MSA word ‘\( \text{ًءَفِين} \)’ (wfyy) which means *loyal* does not exist in our MSA dictionary so the repeated letter ‘\( \text{ًء} \)’ will be removed. As a result, the word will be tagged as a preposition but, in fact, it is an adjective.

<table>
<thead>
<tr>
<th>Token</th>
<th>MSA</th>
<th>Surface form</th>
<th>Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>لَا فَيْن</td>
<td>No</td>
<td>واقفين</td>
<td>Standing</td>
</tr>
<tr>
<td>التَّقَام</td>
<td>No</td>
<td>القام</td>
<td>Algasim</td>
</tr>
<tr>
<td>الَّهِ</td>
<td>Yes</td>
<td>الله</td>
<td>Allah</td>
</tr>
</tbody>
</table>

Table 5.2: Examples of elongated words and their surface forms.
CHAPTER 5. POS TAGGING FOR ARABIC TWEETS

<table>
<thead>
<tr>
<th>Elongated Words</th>
<th>AMIRA</th>
<th>MADA</th>
<th>The Stanford tagger</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Before</td>
<td>After</td>
<td>Before</td>
</tr>
<tr>
<td></td>
<td>40%</td>
<td>46.7%</td>
<td>50%</td>
</tr>
</tbody>
</table>

Table 5.3: The impact on accuracy on elongated words before and after dealing with elongated words as a pre-processing step.

- **Handling Character Deletion** We have noticed that users tend to shorten closed-class lexical items more than other speech classes to overcome tweet length restrictions since it is easy for recipients of tweets to recognise them. Table 5.4 shows some examples of these classes. We handle these cases by detecting and replacing them by their IV equivalents. By using this approach, we are able to improve the accuracy of tagging.

<table>
<thead>
<tr>
<th>Short form(OOV)</th>
<th>Surface form(IV)</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>ع</td>
<td>على</td>
<td>Preposition</td>
</tr>
<tr>
<td>ف</td>
<td>في</td>
<td>Preposition</td>
</tr>
<tr>
<td>م</td>
<td>ما</td>
<td>Neg. particle</td>
</tr>
<tr>
<td>ي</td>
<td>يا</td>
<td>Voc. particle</td>
</tr>
</tbody>
</table>

Table 5.4: Character deletion and surface forms.

words with deleted characters as shown in Table 5.5. This step does not introduce errors because people do not use single characters in normal Arabic and when they do write one of these single characters it is always one of the words in Table 5.4.

<table>
<thead>
<tr>
<th>Character Deletion</th>
<th>AMIRA</th>
<th>MADA</th>
<th>The Stanford tagger</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Before</td>
<td>After</td>
<td>Before</td>
</tr>
<tr>
<td></td>
<td>16.7%</td>
<td>69.6%</td>
<td>20.8%</td>
</tr>
</tbody>
</table>

Table 5.5: The impact on accuracy on character deletion before and after dealing with character deletion as a pre-processing step.

- **Handling Slang** We handle these cases by mapping slang words to their IV equivalents, but slang is an open class and it is difficult to detect all slang items within the tweets.
CHAPTER 5. POS TAGGING FOR ARABIC TWEETS

genre. Therefore, we order 747k types from our corpus (10 million tokens) in descending order based on their frequencies. We select the twenty most frequently occurring slang words from them because we notice that the frequencies of the slang words that have lower rank drop significantly which indicate that these words are not common. Then, we map the twenty slang words to their IV equivalents as shown in Table 5.6.

<table>
<thead>
<tr>
<th>Slang (OOV)</th>
<th>IV equivalent</th>
<th>Translation</th>
<th>Frequency in the corpus</th>
</tr>
</thead>
<tbody>
<tr>
<td>إلي أو اللي</td>
<td>الذي</td>
<td>that</td>
<td>22,946</td>
</tr>
<tr>
<td>بس</td>
<td>لكن</td>
<td>but</td>
<td>20,731</td>
</tr>
<tr>
<td>مش مو</td>
<td>ليس</td>
<td>not</td>
<td>7,630</td>
</tr>
<tr>
<td>لذا وش</td>
<td>لماذا</td>
<td>why</td>
<td>7,135</td>
</tr>
<tr>
<td>عشان عنان</td>
<td>لأن</td>
<td>for</td>
<td>5,357</td>
</tr>
<tr>
<td>لا أدرى</td>
<td>لا أدرى</td>
<td>I don’t know</td>
<td>2,072</td>
</tr>
<tr>
<td>زي</td>
<td>مثل</td>
<td>like</td>
<td>2,042</td>
</tr>
<tr>
<td>أيه</td>
<td>نعم</td>
<td>yes</td>
<td>1,918</td>
</tr>
<tr>
<td>أيش</td>
<td>ماذا</td>
<td>what</td>
<td>1,794</td>
</tr>
<tr>
<td>لين</td>
<td>حتى</td>
<td>until</td>
<td>1,713</td>
</tr>
<tr>
<td>ده</td>
<td>هذا</td>
<td>this</td>
<td>1,083</td>
</tr>
<tr>
<td>كيف</td>
<td>كيف</td>
<td>how</td>
<td>953</td>
</tr>
</tbody>
</table>

Table 5.6: Slang words and their IV equivalents.

By using this approach, we are able to improve the accuracy of slang words tagging as shown in Table 5.7. This step cannot introduce errors since we have used a fixed list of slang words.

<table>
<thead>
<tr>
<th>Slang</th>
<th>AMIRA Before</th>
<th>AMIRA After</th>
<th>MADA Before</th>
<th>MADA After</th>
<th>The Stanford tagger Before</th>
<th>The Stanford tagger After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slang</td>
<td>30.4%</td>
<td>41.2%</td>
<td>32%</td>
<td>45.4%</td>
<td>15.9%</td>
<td>49%</td>
</tr>
</tbody>
</table>

Table 5.7: The impact on accuracy on slang before and after dealing with slang as a pre-processing step.
5.2.2 Post-processing

- **Handling Named Entities** These can be recognised by using gazetteer lists. We use ANERGaze[3] which is a collection of three Gazetteers— (i) Locations: contains names of continents, countries, cities, etc.; (ii) People: has names of people collected manually from different Arabic websites; and finally (iii) Organisations: contains names of organisations like companies, football teams, etc. By using this approach, we are able to improve the accuracy of named entities tagging as shown in Table 5.8. This step might introduce false positives because there are named entities and common nouns that have been written the same way in Arabic and there is no information in the text that helps systems to recognise named entities, like capital letters in English.

<table>
<thead>
<tr>
<th>Named Entities</th>
<th>AMIRA Before</th>
<th>AMIRA After</th>
<th>MADA Before</th>
<th>MADA After</th>
<th>The Stanford tagger Before</th>
<th>The Stanford tagger After</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>49.2%</td>
<td>72.7%</td>
<td>57%</td>
<td>76.5%</td>
<td>66.8%</td>
<td>75.1%</td>
</tr>
</tbody>
</table>

Table 5.8: The impact on accuracy on named entities before and after dealing with named entities as a post-processing step.

- **Handling English Words** Our focus is on Arabic tweets, but some of them contain English words. These words may refer to events, locations, English hashtags or retweet of English tweets with comments written in Arabic and they are part of the syntactic structure of Arabic tweets. So, they need to be tagged correctly. In this case, we use the Stanford English tagger (Toutanova et al. 2003) to tag English words as a post-processing step. By using this approach, we are able to improve the accuracy of English words tagging as shown in Table 5.9. This approach can produce errors with Arabizi because if there were Arabizi in tweets then the tagger would mark them all as nouns.

---

5.2.3 Handling Twitter-specific items, emoticons and emoji

We use regular expression rules to detect and tag Twitter-specific elements such as mentions, hashtags, urls, etc. by doing some pre-processing and then tagging and finally doing post-processing. As an example, we present the way we deal with hashtags: all the remaining Twitter elements are tagged in similar ways. First, we detected hashtags by using regular expression rules. Then, we removed the hashtag signs and underscores from raw tweets. Next, we tagged them by using AMIRA, MADA and the Stanford tagger. Finally, we inserted hashtag signs in their original place in tweets to indicate the beginning and the end of hashtags content as shown in Table 5.10. By using this approach, we are able to tag all Twitter-specific items, emojis and emoticons correctly as shown in Table 5.11. We have a fixed list of emoticons and if new emoticons are created, we will have then a potential problem. However, new emoticons are rarely created. Emojis are open class but they can be fairly easily detected by inspecting their unicode.

<table>
<thead>
<tr>
<th>English Words</th>
<th>AMIRA Before</th>
<th>AMIRA After</th>
<th>MADA Before</th>
<th>MADA After</th>
<th>The Stanford tagger Before</th>
<th>The Stanford tagger After</th>
</tr>
</thead>
<tbody>
<tr>
<td>35.6%</td>
<td>57.5%</td>
<td>17.2%</td>
<td>29.8%</td>
<td>11.7%</td>
<td>16.1%</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.9: The impact on accuracy on English words before and after dealing with English words as a post-processing step.

<table>
<thead>
<tr>
<th>Raw Tweet</th>
<th>حياني فلبيت بقضيها جنب الشاحن!! #جالاكسي لا_تتكلمني</th>
</tr>
</thead>
<tbody>
<tr>
<td>MADA</td>
<td>... !,punc !,punc #,punc jAlAksy,noun _noun IA,verb</td>
</tr>
<tr>
<td>Pre-processing</td>
<td>حياني فلبيت بقضيها جنب الشاحن!! جالاكسي لا_تتكلمني</td>
</tr>
<tr>
<td>MADA</td>
<td>... punc !,punc jAlAksy,noun IA,part_neg tklmny,verb</td>
</tr>
<tr>
<td>Post-processing</td>
<td>... punc !,punc &lt;hash&gt; jAlAksy,noun IA,part_neg tklmny,verb &lt;/hash&gt;</td>
</tr>
</tbody>
</table>

Table 5.10: Example of handling Twitter-specific items (Tag hashtag’s words by using Pre- and post-processing).
In fact, the taggers did not just mistag Twitter elements, but they also mistagged some MSA words in the same tweets because the text is noisy and the taggers rely on contextual clues. By using the above approach, we are not just able to tag Twitter elements correctly but we also make the context less noisy so the taggers are more likely to tag MSA words correctly, as with ‘IA’ in Table 5.10. As a result, MSA words tagging accuracy is improved as shown in Table 5.12.

<table>
<thead>
<tr>
<th>MSA words</th>
<th>AMIRA</th>
<th>MADA</th>
<th>The Stanford tagger</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Before</td>
<td>After</td>
<td>Before</td>
</tr>
<tr>
<td></td>
<td>71.8%</td>
<td>72.5%</td>
<td>79.3%</td>
</tr>
</tbody>
</table>

Table 5.12: The impact after applying pre- and post-processing on MSA words accuracy.

### 5.2.4 Results for pre- and post-processing

In our experiments, the taggers were adapted to handle Twitter phenomena. The experiments were run using three off-the-shelf taggers trained on the Penn Arabic Treebank and our augmented approach to address Arabic tweets noisiness as described in Section 5.2. Table 5.13 shows the performance difference on each of three categories of mistagged tokens and the overall performance compared with their baseline performance in Table 4.4. By combining normalisation and external knowledge, we are able to reduce unknown tokens in each category, which boosts the taggers’ performance. The overall performance of the three taggers increases by twelve percent for AMIRA and by thirteen percent for MADA and Stanford. This improvement in accuracy will reduce the propagation of POS tagging errors to downstream applications on Arabic tweets such as information extraction. We can infer from the above
results that the augmented version of MADA is the most appropriate tagger for tagging Arabic tweets since it outperforms its counterparts of AMIRA and Stanford in accuracy.

However, the accuracy is not only the factor that should be considered when tagging Arabic tweets. The tagging speed is another crucial factor to take into account since there are millions of tweets that need to be tagged. So, the most suitable tagger may be the one that performs well in terms of both speed and accuracy. So, we measure the taggers’ speed below.

<table>
<thead>
<tr>
<th>Tokens</th>
<th>AMIRA Before</th>
<th>AMIRA After</th>
<th>MADA Before</th>
<th>MADA After</th>
<th>The Stanford tagger Before</th>
<th>The Stanford tagger After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic words</td>
<td>67.1%</td>
<td>70.4%</td>
<td>73.8%</td>
<td>77.9%</td>
<td>54.0%</td>
<td>62.1%</td>
</tr>
<tr>
<td>Twitter-specific</td>
<td>0.0%</td>
<td>100%</td>
<td>0.0%</td>
<td>100%</td>
<td>0.0%</td>
<td>100%</td>
</tr>
<tr>
<td>Non-Arabic</td>
<td>19.9%</td>
<td>68.3%</td>
<td>9.1%</td>
<td>66.1%</td>
<td>14.0%</td>
<td>66.1%</td>
</tr>
<tr>
<td>Overall</td>
<td>60.2%</td>
<td>72.6%</td>
<td>65.8%</td>
<td>79.0%</td>
<td>49.0%</td>
<td>65.2%</td>
</tr>
</tbody>
</table>

Table 5.13: Impact of applying pre- and post-processing on POS tagging accuracy.

**Tagging speeds** We measure speed performance of AMIRA2.1, MADA3.2 and the Stanford3.5 POS taggers on 250k tweet tokens. It is measured in words processed per second. The speed evaluation is conducted for Stand-alone (raw input) modes on a Dell XPS laptop computer with Intet(R) Core(TM)2 Duo CPU T8100 @2.1GHz and 4GB memory.

<table>
<thead>
<tr>
<th>Tagger</th>
<th>Accuracy</th>
<th>Speed(words/sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMIRA</td>
<td>72.6%</td>
<td>289</td>
</tr>
<tr>
<td>MADA</td>
<td>79.0%</td>
<td>48</td>
</tr>
<tr>
<td>The Stanford tagger</td>
<td>65.2%</td>
<td>8966</td>
</tr>
</tbody>
</table>

Table 5.14: Accuracy and speed comparison for the taggers.

As shown in Table 5.14 the MADA tagger outperforms AMIRA and the Stanford tagger in tagging accuracy, but it is the slowest tagger, whereas the Stanford tagger is the fastest tagger (31-186x faster than the others), but it is the lowest tagger in tagging accuracy. So, neither of them is ideal for tagging Arabic tweets at this stage and they need further improvements. Due to the fact that we have no access to their code, the reasons for the variations in speed are
not visible. Based on that, we decided to further improve the accuracy of the Stanford tagger while preserving its speed by using Arabic tweets training data. However, there is no labelled training data available to do so. Therefore, we use bootstrapping on unlabelled data to create a sufficient amount of labelled training tweets.

5.3 Bootstrapping

Bootstrapping is generally used to create a labelled training data from large amounts of unlabelled data (Cucerzan and Yarowsky 2002). There are different ways to select the labelled data from the tagger outputs. We follow Clark et al. (2003) in using an agreement-based training method. We use the augmented versions of AMIRA, MADA and Stanford taggers to tag a large amount of Arabic tweets and add the tokens which they agreed upon to the pool of training data. Then, we retrain the Stanford tagger on the selected labelled data.

5.3.1 Agreement-based bootstrapping results

We used the augmented versions of AMIRA, MADA and the Stanford taggers to tag 25K Arabic tweets which contain 203,682 tokens. In these experiments, we focused on Arabic words so we omitted all non-Arabic tokens from the tagged data. This reduced the number of tokens used in the agreement-based bootstrapping experiment to 166,573 Arabic words. Then, we mapped the tagger outputs to our unified tagset. Next, we admitted the maximal sequence of words in each tweet that all taggers were agreed upon to the pool of training data. The taggers reached agreement on 60.4% of Arabic words, which produced 100,691 agreed bootstrapped tokens as a training data. Finally, we trained the Stanford tagger on the bootstrapped training data (see Figure 5.2). Training on bootstrapped data improves the performance of the Stanford tagger from 55% to 66.5% on Arabic words and from 49.0% to 69.1% on overall tokens (Table 5.15). We tried adding 50K further Arabic words to the training pool by repeating the same
experiment on different Arabic tweets, but they did not give a performance increase which suggests no potential benefit from more bootstrapping data.

We have noticed that 75% of Arabic tweets from our dataset in Section 4.2 comprises MSA words, so we thought it would be worth investigating whether adding some amount of labelled newswire tokens to the pool of bootstrapped data above and retraining the Stanford tagger on it would boost its accuracy. Table 5.15 shows that the performance of the Stanford tagger improves from 66.5% to 72.1% on Arabic words and from 69.1% to 74% on overall

Figure 5.2: Agreement-based bootstrapping.
tokens when the Penn Arabic Treebank was added to the training pool. By using the above approaches, we are able to increase the accuracy of Stanford tagger while preserving its speed. It now outperforms AMIRA in terms of speed and accuracy. It also outperforms MADA in terms of speed and it is not a long way behind MADA’s accuracy. So, it is the most suitable tagger for tagging Arabic tweets.

<table>
<thead>
<tr>
<th>Training data</th>
<th>Overall</th>
<th>Arabic words</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Penn Arabic Treebank</td>
<td>65.2%</td>
<td>62.1%</td>
<td>145K</td>
</tr>
<tr>
<td>Bootstrapped</td>
<td>69.1%</td>
<td>66.5%</td>
<td>100K</td>
</tr>
<tr>
<td>The Penn Arabic Treebank+Bootstrapped</td>
<td>74.0%</td>
<td>72.1%</td>
<td>245K</td>
</tr>
</tbody>
</table>

Table 5.15: The Stanford tagger performance on Arabic Tweets using different training data.

5.4 Summary

In this Chapter, we have described the steps involved in the development of an Arabic tweets tagger. We introduced approaches for avoiding the noisiness of this genre of text by combining normalisation and external knowledge to boost the tagger’s performance. Then, we generated training data by applying agreement-based bootstrapping on heterogeneous tagger outputs to retrain the Stanford tagger on it. These combined to improve POS tagging for Arabic tweets. Our techniques yield a very fast and robust POS for Arabic tweets. By using a pool of bootstrapped data combined with the Penn Arabic Treebank to train the augmented version of the Stanford tagger, we are able to improve its accuracy from 49% to 74% on Arabic tweets while preserving its speed. These results are not far behind the state-of-the-art for tagging other languages in Twitter (e.g. English, French, German).

This means that subsequent stages of the NLP pipeline will be significantly less reliable for tweets than they are for ordinary languages. However, we will see in Chapter we can do reasonably accurate parsing on the basis of the output of the augmented Stanford tagger.

---

4JAR file for the tagger and bootstrapped data are available upon request. (fralbogamy at gmail dot com)
In the next chapter, we will explore different approaches towards stemming Arabic tweets. We will use the corpus which is tagged by our tagger in this chapter to obtain a more fine-grained tagged corpus in order to parse it.
Chapter 6

Stemming Arabic Tweets

The purpose of the chapter is to explore different approaches to stemming words in Arabic tweets (RO4). As noted in Section 3.2, Arabic tagging and stemming are typically carried out as separate stages. We initially tagged Corpus 1 with a coarse-grained tagset and then refined that by stemming to produce a tagged corpus with a fine-grained tagset that can be used as an input to the next stage (parsing) in the adapted NLP pipeline for Arabic tweets. The white-out boxes in Figure 6.1 show where the work in this chapter fits into the overall NLP pipeline. This work has been published in Albogamy and Ramsay (2016b).

6.1 Introduction

Before we can begin to parse text, we have to be able to identify the words that make it up (i.e. stemming). This is challenging for languages like Arabic which make extensive use of clitics, since identifying the units that make up a space-separated ‘word’ is not straightforward (Habash et al. 2009). It is particularly challenging for the kind of text that we find in social media, where the vocabulary is open-ended and where the normal rules of the language are frequently violated, either deliberately (to save space or for rhetorical effect) or by mistake.
As mentioned in Section 4.2.3, we are using a tagset obtained by unifying the tagsets from AMIRA, MADA and Stanford. The resulting tagset consists of the main POS tags both coarse- and fine-grained, in addition to Twitter-specifics tags. At this point all affixes have not been identified, for example all nouns are tagged as NN and all verbs are tagged VB. To achieve our ultimate goal in this research, which is parsing, we have decided to use a finer-grained set for all affixes (e.g. pronouns and prepositions) because we believe we can split them accurately so a parser can be provided with more useful information which helps the parser to produce the correct analysis. Clitic pronouns can be arguments of verbs in Arabic so splitting them off will help the parser to sort out the syntactic structure, particularly given that Arabic is a pro-dropping language.

In this study, we investigate two different techniques for stemming Arabic tweets: heavy stemming and light stemming. Heavy stemming approaches are dictionary-based whereas light stemming techniques are not and they are also faster. We find that a light stemming approach is the most suitable approach for stemming Arabic tweets words because tweets have a very open lexicon and there are millions of tweets that need to be stemmed. The results of stemming experiments are compared with Arabic stemmers which use similar approaches described in the literature.

The rest of this Chapter is organised as follows. In Section 6.2, we give an overview of the Arabic word forms. The corpus that we used in our experiments and our dataset is explained in Section 6.3. In Section 6.4, we develop and evaluate a heavy stemmer. In Section 6.5, we implement and test a light stemmer. In Section 6.6, we reflect on the work described in the chapter.
Figure 6.1: The NLP pipeline adapted for Arabic tweets (stemming). White boxes are discussed in this Chapter.
6.2 Arabic word forms

Arabic is a morphologically rich language where the letters are attached together to form a word. A word often conveys complex meanings that can be decomposed into several morphemes (i.e. prefix, stem, suffix). Consequently, it presents significant challenges to many NLP applications such as tagging. For example, the Arabic word ‘وسيفالون’، which means “and they will do”, consists of five elements as shown in Table 6.1.

<table>
<thead>
<tr>
<th>Arabic</th>
<th>Translit.</th>
</tr>
</thead>
<tbody>
<tr>
<td>و</td>
<td>w</td>
</tr>
<tr>
<td>س</td>
<td>s</td>
</tr>
<tr>
<td>ي</td>
<td>y</td>
</tr>
<tr>
<td>فعل</td>
<td>fEl</td>
</tr>
<tr>
<td>ون</td>
<td>wn</td>
</tr>
</tbody>
</table>

Table 6.1: Example of Arabic Affixes.

The order of the elements that make up an Arabic noun or verb is fixed. These affixes differ based on the word type, but in general we can represent the word as follows:

Word ≡ Prefixes + Stem + Suffixes

![Figure 6.2: Possible sub-tokens in Arabic nouns.](image)

Arabic morphotactics allow words to have affixes. Affixes themselves can be concatenated one after the other. A noun can comprise up to six sub-tokens as illustrated by Figure 6.2. It

---

1The word is delimited by a white space.
may have a conjunction, a preposition, a definite article attached in front of it, and an agreement marker and a pronoun attached on the end.

![Figure 6.3: Possible sub-tokens in Arabic verbs.](image)

Similarly a verb can comprise up to five sub-tokens as illustrated by Figure 6.3. It may have a conjunction, a tense marker (which may include a future marker) attached in front of it and a person marker and a pronoun attached on the end. The order of these items is fixed (even in tweets) – for a noun, the conjunction (if present) must precede the preposition (if present), the preposition (if present) must precede the definite article (if present) and so on. The combination of words with affixes is governed by various rules. These rules are called grammar-lexis specifications (Dichy and Farghaly, 2003). The main grammar-lexis for Arabic are shown in Figure 6.4:

- A noun stem can be any sequence of three or more characters.
- The prefix ‘s’, which denotes the future of verbs, is only combined with imperfective verb stems.
- The form of a tense marker before the verb has to match the form of the person marker that follows.

![Figure 6.4: Arabic grammar-lexis.](image)

### 6.3 Experimental setup

To create our test set, we select the distinct nouns and verbs from Corpus 1 (390 tweets) which is the same dataset we used in Chapter 4 and tagged by our tagger in Chapter 5 (see Section 4.2.1 for more details). A set of correctly annotated nouns and verbs for this sample (gold standard) is required in order to be able to appraise the outputs of the stemmer. Once we have
this, we can compare the outputs of the stemmer with this gold standard. Since there is no publicly available annotated corpus for Arabic tweets, we manually annotated our dataset to be used as a gold standard (hereafter Corpus 2). The dataset contains 1250 nouns and 692 verbs. To estimate the inter-annotator agreement for our dataset, we use one independent judgment from a native speaker\(^2\). We sampled 20% of each type and then we asked her to annotate them. We agreed on 96.3% and 92.4% on nouns and verbs respectively. As before, this sets an upper bound on the achievable accuracy of the stemmer since it is impossible for it to agree with both annotators in more than 96% / 92% of cases.

As stated, in our experiments, we used two different approaches to stemming Arabic tweets: a heavy stemming approach and a light stemming approach.

6.4 Heavy stemmer

Although Arabic tweets and MSA are closely related and share many characteristics, there are substantial differences between them in lexicon and syntax (Albogamy and Ramsay, 2015a). The lexicon is always evolving and many words in Arabic tweets are not present in MSA. Therefore, the existing dictionary-based stemming systems will not work for Arabic tweets. The statistical approaches that are used to induce a list of affixes automatically are also not applicable because there is no large annotated training data available. In the Arabic tweets genre, there are millions of words that need to be stemmed, so stemming approaches that infer a set of rules to identify words patterns are not suitable because they consume too many computational resources (Al-Serhan and Ayesh, 2006).

Based on that, we initially decided to use a heavy stemming approach similar to the one that has been used by Khoja and Garside (1999), but with a dictionary that is appropriate for the Arabic tweets genre and using highest stem frequency strategy. However, there is no such

\(^2\)She is a linguistics lecturer at Qatar university
dictionary publicly available yet. Hence, we extracted our own dictionary from the corpus by selecting words which occur more than 20 times in the 747k tokens types in the corpus (the corpus contains 10 million tokens). This dictionary contains 51K entries, from which we create nouns and verbs frequency tables. We define all possible affixes and write a set of rules of valid prefixes, suffixes and word forms and select the highest stem frequency as the correct stem. We are interested in two major word types; noun and verb. We defined them as a combination of allowable prefixes and suffixes, and a word stem which is all the letters between the prefix and the suffix.

![Figure 6.5: Heavy stemmer workflow.](image)

Our approach has two phases and deals with one word at a time (See Figure 6.5). The first phase is dedicated to taking the input word and trying it against the definition of its word type (noun or verb) in the grammar. Then the stemmer will produce a list of all possible stems. The
CHAPTER 6. STEMMING ARABIC TWEETS

Analysis 1:['w', 'b', 'al', 'yd']
- prefix: ['w', 'b', 'al']
- stem: yd
- stem frequency: 306
Analysis 2:['w', 'b', 'alyd']
- prefix: ['w', 'b']
- stem: alyd
- stem frequency: 157
Analysis 3:['w', 'balyd']
- prefix: w
- stem: balyd
- stem frequency: 0
Analysis 4:['wbalyd']
- stem: wbalyd
- stem frequency: 73

Figure 6.6: Possible stems for the word باليد ‘wbalyd’ and frequencies.

The second phase is to select the highest stem frequency from frequency tables as the correct stem. We use the word باليد ‘wbalyd’ which means “and by the hand” to demonstrate our approach (see Figure 6.6). The stemmer tried this word against the noun type grammar. It produced three possible stems. Then, the highest stem frequency (analysis 1) was selected ‘yd’ as the stem for this word. In this particular example, the stem chosen stem is the correct stem. We notice that only 44% of generated verbs stems are found in the verbs frequency table and 74% of them are correct. We also notice that only 58% of generated nouns stems are found in the nouns frequency table and 76% of them are correct (see Table 6.2). We expect that this is because the extracted dictionary is too sparse. We extracted the dictionary from 10 million tokens and even so one in ten words is unknown. Therefore, it seems likely that the dictionary will always be too sparse no matter how big it is. The stemming speed by using this approach is 2451 words/second.

3In this example, the highest stem frequency is also the shortest stem.
6.5 Light stemmer

Using the heavy stemming approach, we notice that when generated stems were found in the frequency tables, 95% of them were actually the shortest stems. As a result, we use the previous approach but instead of selecting the highest stem frequency, we select the shortest stem of a word as the correct stem so we do not need a dictionary anymore. This approach is referred to as a light stemming approach and it is similar to the one that has been used in (Al-Kabi et al., 2015), but we cover all Arabic affixes (Table 6.3). Therefore, we write the basic grammar of valid affixes combination that describe the morphological rules of Arabic noun and verb (Figure 6.4) in regular expression interpreter. Patterns of this kind can be expressed as regular expressions. The easiest way to do this is by building them up step-by-step, using rules of the kind given in Figure 6.7.

```
"NSTEM": "\.{3, }?",
"ART": "Al|",
"CONJ": "w|b",
"PREP": "b|k|l",
"PRON": "k|kmA|km|kn|h(A+)|hm(A*)|hn|hm|nA|y",
"AGREE": "wn|yn|p|An|At",
"NOUN": "(CONJ)?(PREP)?(ART)?(?P<stem>NSTEM)(AGREE)(PRON)?",
```

Figure 6.7: Preliminary regular expressions for nouns.

These rules express the basic observations above about the structure of Arabic nouns. We will refine them below. For now, the key point is that rules of this kind can easily be converted into standard regular expressions (regexes) simply by replacing defined terms by their defini-

---

4The full listing of this code is included as Appendix F.
### Table 6.3: Arabic affixes.

<table>
<thead>
<tr>
<th>Prefixes</th>
<th>Transliteration</th>
<th>Suffixes</th>
<th>Transliteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>آل</td>
<td>Al</td>
<td>ي</td>
<td>y</td>
</tr>
<tr>
<td>و</td>
<td>w</td>
<td>ه</td>
<td>h</td>
</tr>
<tr>
<td>ب</td>
<td>b</td>
<td>ها</td>
<td>ha</td>
</tr>
<tr>
<td>ل</td>
<td>l</td>
<td>هم</td>
<td>hm</td>
</tr>
<tr>
<td>س</td>
<td>s</td>
<td>هن</td>
<td>hn</td>
</tr>
<tr>
<td>و</td>
<td>O</td>
<td>ك</td>
<td>k</td>
</tr>
<tr>
<td>ن</td>
<td>n</td>
<td>كن</td>
<td>kn</td>
</tr>
<tr>
<td>أُ</td>
<td>A</td>
<td>كم</td>
<td>km</td>
</tr>
<tr>
<td>يَ</td>
<td>y</td>
<td>كمَا</td>
<td>kmA</td>
</tr>
<tr>
<td>تُ</td>
<td>t</td>
<td>نا</td>
<td>nA</td>
</tr>
<tr>
<td>نيَ</td>
<td>ny</td>
<td>اَ</td>
<td>A</td>
</tr>
<tr>
<td>انَ</td>
<td>An</td>
<td>واَ</td>
<td>wA</td>
</tr>
<tr>
<td>وَن</td>
<td>wn</td>
<td>نَ</td>
<td>n</td>
</tr>
<tr>
<td>نَمَا</td>
<td>tmA</td>
<td>تَم</td>
<td>tm</td>
</tr>
<tr>
<td>تَمَ</td>
<td>tm</td>
<td>تَن</td>
<td>tn</td>
</tr>
<tr>
<td>تَ</td>
<td>t</td>
<td>يَن</td>
<td>yn</td>
</tr>
<tr>
<td>ةَ</td>
<td>p</td>
<td>اَت</td>
<td>At</td>
</tr>
</tbody>
</table>
tions. The resulting regexes are fairly complex, as shown in Figure 6.8. They are, however, just regexes, and hence can be compiled to run extremely efficiently.

\`
\`\(^\((?P<CONJ>w|b)?(?P<PREP>b|k|l)?(?P<ART>A|Al)\)\)?\((?P<NSTEM>.{3, })((?P<AGREE>wn|yn|p|An|At)\)((?P<PRON>k|kmA|km|kn|hn|hm|nA|y)?)\)$\`

Figure 6.8: Regular expression for components of a noun.

We can do much the same with verbs. A word with a verb at its heart can be made up of a conjunction, a future marker, a tense marker, the verb itself, a person marker and a pronoun, where the only obligatory element is the verb, as shown in Figure 6.9. As with the basic rule for nouns it looks as though substituting definitions of the terms that appear in this pattern will produce a standard regex.

"VERB": "(CONJ)?(TENSE)(?P<stem>VSTEM)\n(PERSON)?(VPRON)?action"

Figure 6.9: Basic pattern for verbs.

However, there is a connection between the tense prefix and the person marker. Past tense forms (which have an empty tense prefix) go with past tense person markers, present tense forms go with specific person markers. We need to be able to write constrained matches – to say that the second person form of the present tense prefix goes with second person suffixes and so on. This requires that constraints are carried forward during the matching process – that when we are looking at the person marker we can remember which version of the tense prefix was used.

This is not possible with standard regexes, but the ‘regular expression’ handling in a number of languages (including Python and at least some versions of Perl) includes apparently minor extensions which increase their expressive power. In particular, the Python implementation allows us to assign names to captured groups (this is an innocent extension) and to write patterns which will match only if some named group has a non-null value. A pattern like
STEMMING ARABIC TWEETS

Figure 6.10: Present tense and person markers with agreement constraints.

There are four present tense prefixes, ُ(O), ﺕ (n), ِ(t) and ﺕ (y). We could capture this by saying that the present tense prefixes have to fit the pattern ُ|ٍ|١|١|١|١. However, these prefixes each combine with different sets of person suffixes. It would therefore be useful to know not just that one of them had been matched, but which of them had been matched.

We can do this by naming them: (ُ|ٍ|١|١|١|١|١|١|١|١|١|١) names three groups – if the prefix matched ُ then the group called ُ will be bound; if it matched ١ then the group called ١ will be bound; if it matched ١ then the group called ١ will have been bound. We can think of group names as variables which get assigned a value when the pattern that they name is matched, or as features which get unified with the string that their pattern matched.

Given the ability to see whether some group name has been bound, we can link the set of person markers to the tense prefixes. There are, in total, 12 distinct person affixes (including the empty affix): ( ), أ (A), أن (An), ن (n), ت (nA), تم (tm), تما (tmA), تما (tn), و (wA), ون (wn) and ين (yn). If we simply demanded that every verb had a tense prefix, by using the pattern أ| أن| ن| ت | تم | تما | تما | تما |Î | و | ون | ين, and a person marker, we would decompose a
string like ُ يدرسَم into a tense prefix ُ (y), a stem ُ (drs) and a person marker ُ (tm), despite the fact that ُ (y) is a third person present tense marker and ُ (tm) specifies that the word is second person plural past. Given, however, that we have a record of which tense prefix was used, we can introduce a set of person markers, each to be used only if the specified group was matched: "AGR_Y": "(An|A|wA|wn|n)" from Table 6.10 for instance, lists the person markers that go with the tense prefix y. The pattern for person markers, "PERSON": "(?ON)AGR_ON | (?Y)AGR_Y | (?t)AGR_T)) | (?past)AGR_PAST)" then says that if the group called ON has been matched then we can try to match an empty person marker, if the group Y has been matched then we can have a person marker that matches (An|A|wA|wn|n) (note that this includes, as the last option, the empty affix).

![Figure 6.11: Light stemmer workflow.](image)

The light stemmer accepts a text file that includes the Arabic nouns and verbs (See Figure 6.11). Then, it produces the stems of those words as shown in Table 6.1. In our experiments, we used two different settings of stem length: 3-character stem and 2-character stem. In the first experiment, we restricted the length of stems to at least three characters for nouns and
verbs since most of words in MSA have a 3-letter stem. We achieved 78.16% and 67.19% stemming accuracy for nouns and verbs respectively as shown in Table 6.4. However, we noticed that for some nouns and verbs their stems were written in two letters. Therefore, they were ignored by the stemmer. In the second experiment, we restricted the length of stems to at least two characters or more for nouns and verbs to cover all missed cases in the first experiment. The stemming accuracy for verbs was improved by about ten percent whereas the stemming accuracy for noun was decreased by around two percent as shown in Table 6.5.

Based on the results of the experiments, we decided to use two different constraints; one for nouns and the other for verbs. The length of stems has to be at least three characters or more for nouns and to be at least two characters or more for verbs. As we can see in Table 6.6, the best overall stemming performance is achieved when the minimum stem length is three characters for nouns whereas it is two characters for verbs. By using this approach, we are able to reach 77.91% overall accuracy (see Table 6.6) and the speed is improved because the grammar was written by using regular expression and there is also no need for consulting the dictionary in this approach (182K words/second). We have compared our stemming accuracy with Al-Kabi et al. (2015), Khoja and Garside (1999) and Ghwanmeh et al. (2009). Those three stemmers yield accuracies of 75.03%, 74.03% and 67.40% respectively (Al-Kabi et al., 2015). Our results show improvements over the performance of those three well-known stemmers for
TABLE 6.6: Overall light stemming accuracy (nouns and verbs).

<table>
<thead>
<tr>
<th>Stem length</th>
<th>Overall accuracy (nouns and verbs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;=2 characters</td>
<td>77.03%</td>
</tr>
<tr>
<td>&gt;=3 characters</td>
<td>74.25%</td>
</tr>
<tr>
<td>Verb stem &gt;=2 and noun stem &gt;=3</td>
<td>77.91%</td>
</tr>
</tbody>
</table>

Arabic. Moreover, we used a light stemming approach whereas they used heavy stemming approaches. We also experimented on the Arabic tweets genre which is noisier than MSA. In addition, our light stemmer is fast and we are able to extract two and three character stems length which most stemmers for Arabic cannot achieve.

We examined the words that were incorrectly segmented by our system. The errors can be broadly divided into three categories: under-stemming, over-stemming and orthography errors. Under-stemming errors happen when the stemmer does not remove all affixes in the words. For example, the word رأى للمؤمنين ‘For the believers’: the algorithm removes the first letter of the prefix لل but not the second one, since it is considered as part of the stem in this case (see Table 6.7 the first row). Over-stemming errors happen when the stemmer considers part of the word is an affix and removes it. Consider the Arabic word واحد ‘One’, the stemmer removes the original letter و because it looks like a conjunction (see Table 6.7 the second row). Orthography errors occur when the stemmer removes all affixes and finds the correct stems, but the last letter of the stems have a wrong shape. For example, the word كمتبهم ‘Their word’ the stemmer removes the suffix هم and it considers كمتب as the stem which is correct except that the last letter should be replaced by ك instead of م (see Table 6.7 the third row).
CHAPTER 6. STEMMING ARABIC TWEETS

<table>
<thead>
<tr>
<th>Arabic word</th>
<th>Our stemmer</th>
<th>Gold Standard</th>
<th>Error Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>لَوْمَمَّينَ</td>
<td>لَوْمَمَّينَ</td>
<td>لَوْمَمَّينَ</td>
<td>under-stemming</td>
</tr>
<tr>
<td>‘For the believers’</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>وَاحِدَ</td>
<td>وَاحِدَ</td>
<td>وَاحِدَ</td>
<td>over-stemming</td>
</tr>
<tr>
<td>‘One’</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>كَمْشِهِمْ</td>
<td>كَمْشِهِمْ</td>
<td>كَمْشِهِمْ</td>
<td>orthographical</td>
</tr>
<tr>
<td>‘Their word’</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.7: Examples of stemming errors.

6.6 Summary

In this Chapter, we have proposed, implemented and evaluated a heavy and light stemmer for Arabic tweets. The former uses a dictionary which we extracted from an Arabic tweets corpus whereas the latter does not rely on any root dictionary. The first experiment shows that using the heavy stemmer does not yield comparative results because the extracted dictionary is too sparse due to the fact that tweets have a very open lexicon. In the second experiment, a light stemming approach was evaluated. It uses the shortest stem strategy to extract the stems of the words. It has two phases: phase 1 is dedicated to producing a list of all possible stems by using the grammar, and phase 2 is for selecting the shortest stem as the correct stem instead of selecting the highest stem frequency as in the heavy stemming approach. We compared the light stemmer with three Arabic stemmers where one of them uses a similar approach to ours. Results showed that the light stemmer is better in terms of accuracy in comparison with other three Arabic stemmers. In addition, the light stemmer is fast and it is able to extract two and three character stems which most stemmers for Arabic cannot do.

In the next chapter, we will explore different approaches towards developing an Arabic tweets dependency treebank which can be used to train a parser. The parser is used to produce dependency trees for Arabic tweets. We will use a corpus which is tagged by our tagger (Chapter 5) and stemmed by the light stemmer described in this chapter.
Chapter 7

Syntactic Parsing for Arabic Tweets

The purpose of this chapter is to explore different approaches to build a silver standard treebank\(^1\) (RO5), and hence to use a parser to evaluate its usefulness. The whitened out boxes in Figure 7.1 show where the work in this chapter fits into the overall NLP pipeline. The material in this chapter has been published in Albogamy et al. (2017).

7.1 Introduction

Syntactic parsers (data-driven and rule-based) have reached high performance levels on well-formed texts, but they have performed less well on user-generated text such as the type of material that we find in tweets (Kong et al., 2014). The nature of the text content of Arabic tweets is mixed. It includes relatively long sentences (36% of sentences contain 25 or more words, see Figure 7.2) some of which are reasonably well-written (these can be considerably longer than sentences in English tweets because of the fact that Arabic is very economical with characters); others comprise informal material with large numbers of errors and neologisms, and some consist of material which is scrambled. Therefore, annotated data for well-formed genres such as the Penn treebank and the Penn Arabic Treebank is not suited to tweets (Kong et al., 2014). As a result, there is an increasing interest in the development of treebanks of user-

\(^1\)The silver standard refers to an automatically generated corpus without human intervention (An et al., 2003).
generated text because they are invaluable resources for the development of NLP tools. These are usually small, manually annotated corpora: English Web Treebank (16K sentences) \cite{Bies2012}, Tweebank Treebank (929 sentences) \cite{Kong2014}, Spanish Web Treebank (2846 sentences) \cite{Taulé2015} and French Social Media Bank (FSMB) (1700 sentences) \cite{Seddah2012}. However, these are small treebanks because people find that annotating tweets is difficult and the agreement between annotators is not comparable to that for normal languages \cite{Jamatia2014}. So, generating automatically a large “silver” treebank could be beneficial in this case.

![Figure 7.2: Length of tweets (words).](image)

In this Chapter, we investigate the use of a silver treebank that is automatically generated by using the output of different parsers without human intervention. We explore two scenarios: one in which the treebank is built by using just a rule-based parser, and one in which the treebank is generated by using a rule-based parser and a data-driven parser in a bootstrapping approach.

We report on the development of the first Arabic tweets dependency treebank and describe an approach to parsing tweets that have reasonable structures whilst marking the others as being unparsable. We explore the idea that producing a small treebank using a rule-based parser will suffice to train an initial MALT-style parser, which we can then use, in conjunction with the rule-based parser, to produce a much larger treebank which can be used to improve the
Figure 7.1: The NLP pipeline adapted for Arabic tweets (parsing). White boxes are discussed in this Chapter.
performance of the base parser. We used a rule-based parser to produce a dependency treebank partly to save effort instead of annotating it manually but also to remove the nonsense from the training data. If a tweet does not have a sensible structure, then even if we ascribed a structure to it, we would not want to use this to train our data-driven parser, since this structure would not be replicated in unseen tweets. Given that the rule-based parser will fail to provide an analysis of such material, it acts as an automatic filter to remove it. The rule-based parser is, however, quite slow and to construct a large treebank using it is time-consuming. Therefore, we only use it to produce a small treebank as seed training data for a version of MALTParser and use a bootstrapping technique to make a larger treebank out of it.

Our work is, to the best of our knowledge, the first step towards developing a dependency treebank for Arabic tweets which can benefit a wide range of downstream NLP applications such as information extraction, machine translation and sentiment analysis.

### 7.2 Choice of parsing strategy

As mentioned in Section 3.5.3, there are two types of strategy for parsing: rule-based and data-driven. Rule-based parsers have been developed and used for decades in the NLP community. In such parsers, linguistic rules are written to represent knowledge about the syntactic structure of a language. The parser produces the resulting parse trees by applying these rules to input sentences. It uses a dictionary or lexicon to store information about each word in input text before applying the linguistic rules. Although this kind of parser is widely-used in a variety of NLP systems to provide deep linguistic analyses, they do have disadvantages. They are slow and fragile, and it is time-consuming, expensive and tedious to construct dictionaries and to write the rules by expert linguists. Additionally, they are hard to maintain.

In recent years, data-driven parsers have been widely used due to the availability of annotated data such as the Penn Treebank [Marcus et al., 1993] and the Penn Arabic Treebank
These parsers are robust and produce state-of-the-art results compared to rule-based ones. However, the reliance on annotated data is one of the significant disadvantages of using data-driven parsers because a large amount of rich annotated data is not always available for many languages and domains due to various factors (Ramasamy and Žabokrtský, 2011).

Although data-driven parsers have achieved state-of-the-art results on well-formed texts, they have not performed well on user-generated texts because the nature of the texts found in user-contributed online forums rarely complies with the standard rules of the underlying language, which makes them challenging for traditional NLP tools, including data-driven approaches, even if domain adaptation techniques have been used (Dredze et al., 2007; Seddah et al., 2012).

We have constructed dependency treebanks of Arabic tweets by using just a rule-based parser (Section 7.6.3) and by using a bootstrapping approach (rule-based parser and MALT-Parser) (Section 7.6.4). We were able to create a dependency treebank from unlabelled tweets without any manual intervention. The results of experiments show that using a data-driven parser and a rule-based parser together to construct a large training set (treebank) is better in terms of speed of training and of accuracy of parsing than constructing it by using the rule-based parser only.

### 7.3 Choice of syntactic representation

There are two main syntactic representations for parsing, namely phrase structure trees and dependency trees (see Section 3.5.1 for more details). Examples of phrase structure constituency treebanks are the Penn Treebank for English (Marcus et al., 1993) and the Penn Arabic Treebank (Maamouri et al., 2004). In recent years, dependency treebanks have been developed for many languages such as Arabic (Hajic et al., 2004), Czech (Böhmová et al., 2003) and Turkish...
Phrase structure parsing is based on phrase structure grammar (PSG). It describes a natural language by breaking a sentence down into its constituent parts as classified by structural categories such as NP, VP, PP, etc. It depends on phrase structure theory. A phrase structure representation is usually used to represent languages with clear constituency structures and fixed word order patterns such as English (Karlsson et al., 1995). By contrast, dependency parsing is based on dependency grammar (DG). It describes a natural language based on the idea that each word in a sentence depends on other words, except one word which is considered as the root (Tesnière, 1959). A dependency representation is often used to represent languages which allow greater freedom of word order such as Arabic, Czech and Polish (Karlsson et al., 1995).

In this research, we are interested in analysing the structure of Arabic tweets which means we want to extract the grammatical relations between individual words in an Arabic tweet. The language which is used to write an Arabic tweet is actually derived from MSA, so Arabic tweets share some of the characteristics of MSA, such as free word order (Attia, 2008), but it has many new phenomena (Section 2.4). From the above comparison between phrase structure parsing and dependency parsing, we can infer that the most appropriate parsing representation we can use to achieve our aim is dependency parsing, since it will represent the relations between constituents of Arabic tweets and it is often used in free word order contexts such as Arabic tweets. Moreover, it is aligned with the current line of the NLP research community so syntactic dependencies can be used directly in future Arabic tweets NLP research.

## 7.4 Rule-based parsing

We use a rule-based chart parser similar to the one described by Ramsay (1999) (Section 3.5.4.1) (hereafter Parasite*). This parser stops if it finds a complete analysis of a tweet. If it does not find a complete analysis after a predetermined number of edges have been created, it
stops and returns the largest non-overlapping potential fragments. We have no lexicon because of the rapidly changing nature of tweets and the presence of misspellings, both accidental and deliberate – tweets make use of a very open lexicon: even after you have looked at over a million tweets you still find that one in ten words is unknown, which suggests that even if you look at very large corpora you will continue to find new words. Instead we use a tagger with a coarse-grained tagset, simply labelling all nouns as NN, all verbs as VB and so on (see Chapter 5 for more details). It is striking that even without fine-grained subcategorisation labels (e.g. between intransitive, transitive and sentential complement verbs), the rule-based parser produces good quality analyses when it produces anything at all.

Because the rule-based parser looks for maximal fragments, it can also analyse tweets which actually consist of more than one sentence with no punctuation between them. The following tweet for instance consists of three separate sentences:

@alabbas75@DrA_Farouk235

هذا حق محمد الله و نحب الله ورسوله محمد وكل نبي ورسول ونحن خوارج العصر

The rule-based parser returns three sub trees (largest fragments) that represent these sentences as we can see in Figure 7.3.

To use a parser to extract the syntactic structure of Arabic tweets, the parser should take into consideration the grammatical structure of Arabic tweets. As seen in Section 2.4, tweets have many phenomena such as mentions, replies, retweets, hashtags, links, etc. These elements become parts of the tweet’s text and they will play grammatical roles in this context. For example, both the hashtags and the mention in (7.1) are parts of the tweet syntactic structure.

(7.1) عند د سؤال عن القبول في #الجامعة ؟ أسأل عن طريق برنامج

@UniAdmission

بيحادثة - الجامعة لتحصل على اجابة من

These new elements cannot be assigned the traditional POS tags. This means that they cannot easily be dealt with using traditional grammar, and hence it is important to discover their
Figure 7.3: Rule-based parser segments a tweet to three fragments.
grammatical roles. As seen above, these elements are part of tweets so they must have grammatical relations with the rest of tweets, but because they are new parts and do not exist in the MSA grammar, we need to know what kinds of relations they have. There are some efforts in the literature towards aiming at parsing English tweets. In Kong et al. (2014) a dependency parser for English tweets was developed, but they omitted most of the tweets elements from the material to be parsed which leads to losing parts of the content of the tweet. By contrast, in this research we argue that all the new tweet elements play grammatical roles and we will try to discover the structures of Arabic tweets by taking into account all tweet constructions. One way to discover their grammatical roles is to rewrite tweets in ordinary more formal Arabic and try to preserve the meaning as far as possible, which will help us to understand the relationships between these elements and to analyse tweets structures.

We will next show a few examples of the grammatical functions which can be played by these elements and present their dependency trees. It should be noted that the rules that cover the grammatical structure of Arabic tweets have been implemented in the rule-based parser which we used in this research.

1. The link in tweet (7.2) is a Twitter-specific element and functions as the subject of a zero-copula sentence:

   (7.2) Original tweet: صورة لاحق عشاق جيبارد http://t.co/vY0feFK3F2
   Transliteration: Gerrard’s fans one-of picture http://t.co/vY0feFK3F2
   Paraphrase to MSA: هذه صورة لاحق عشاق جيبارد2
   Paraphrase to English: http://t.co/vY0feFK3F2 (is) a picture of one of Gerrard’s fans
2. The link in tweet (7.3) is a Twitter-specific element and works as a modifier:

(7.3) Original tweet:  http://t.co/mT2feDS5n5

Transliteration:  http://t.co/mT2feDS5n5 : raining it’s

Paraphrase to MSA:  http://t.co/mT2feDS5n5

Paraphrase to English:  It is raining : see the picture http://t.co/mT2feDS5n5

3. The reply mark is a Twitter-specific element and it is equivalent to “reply to someone” phrase. It works as a verb in tweet (7.4):
(7.4) Original tweet: انا سويت هذا @AhamedMoh
Transliteration: this did I @AhamedMoh
Paraphrase to MSA: رد على @AhamedMoh: انا سويت هذا
Paraphrase to English: Reply to @AhamedMoh: I did this

Figure 7.6: Dependency tree for tweet (7.4). Left: (English) Right: (Arabic).

4. The mention mark is a Twitter-specific element and works as an object in tweet (7.5):
(7.5) Original tweet: @FahadTiger اليوم قابلت
Transliteration: @FahadTiger today met I
Paraphrase to MSA: @FahadTiger اليوم قابلت
Paraphrase to English: I met @FahadTiger today

Figure 7.7: Dependency tree for tweet (7.5). Left: (English) Right: (Arabic).

5. The re-tweet is a discourse marker and it is equivalent to “someone says” phrase. It works as a verb in tweet (7.6) :
(7.6) Original tweet: @MomiAH: الدراسة صعبة مرررة

Transliteration: very difficult study @MomiAH:

Paraphrase to MSA: يقول أن الدراسة صعبة جداً @MomiAH

Paraphrase to English: @MomiAH says that the study is very difficult

Figure 7.8: Dependency tree for tweet (7.6). Left: (English) Right: (Arabic).

7.5 Data-driven parsing

The data-driven parser that we used is a version of MALTParser (Section 3.5.4.2) (hereafter MALTParser⁺) with three basic data structures – a queue of unexamined words, a stack of words that have been considered but which have not been assigned heads, and a collection of <head, relation, daughter> triples and with three basic actions, namely shift (move an item from the queue to the stack), leftArc (make the top item on the stack a daughter of the head of the queue and remove it from the stack), rightArc (make the head of the queue a daughter of the top item of the stack, remove the head of the queue and move the top item of the stack back to the queue) (Nivre et al., 2006). It is a deterministic parser which uses decision tree ID3 as a machine learning algorithm. It makes a single decision at each step (Quinlan, 1986).
CHAPTER 7. SYNTACTIC PARSING FOR ARABIC TWEETS

7.6 Treebanking and parsing experiments

7.6.1 Implementation

In our experiments, we use Parasite$^+$ and MALTParser$^+$ as black-boxes which require an input in order to give us an output. What does matter here is to find an efficient and effective approach to create a silver treebank by using those parsers. In order to do that, we will investigate two strategies: using just the rule-based parser and the rule-based parser and the data-driven parser in a bootstrapping approach. Parasite$^+$ is implemented by using Prolog whereas MALTParser$^+$ is a Python program. We use a Python script to call them in different scenarios$^2$ (Figure 7.9). In the first experiment (Section 7.6.3), we call only the function $\text{def train(corpus)}$ to generate a large silver treebank by using Parasite$^+$ and train MALTParser$^+$ on it. In the second experiment (Section 7.6.4), we call first the function $\text{def train(corpus)}$ to create a small treebank and train MALTParser$^+$ on it and then we call the function $\text{def retrain(corpus, parser)}$ which uses the parse model (MALTParser$^+$) which has been created by $\text{def train(corpus)}$ to parse a large number of tweets and then it calls Parasite$^+$ to filter these so that only legal analyses remain. Next, it trains a new parse model on the bootstrapped treebank.

$^2$The full listing of this code is included as Appendix G
def train(corpus):
    
    Call Parasite to create the initial treebank: this involves running the Prolog program parser.sav as an external process

    subprocess.Popen("/usr/local/bin/sicstus -r parser.sav -goal consult('%sssegments'), parse2conll('%sbasetreebank'), halt.'%(corpus, corpus)).split(' '), stdout=sys.stdout, stderr=subprocess.STDOUT).wait()

    ## Train MaltParser on the the initial treebank
    return wp.wholething(f="%sbasetreebank.cnll"%(corpus))

def retrain(corpus, parser):
    
    Parse N2 Arabic tweets using MaltParser which has been trained in function def train(corpus) above

    makeCompleteTreeBank(corpus, parser, outsink="%sconstrained.pl"%(corpus), N2=N2)

    ## Call Parasite to return only legal analyses
    subprocess.Popen("/usr/local/bin/sicstus -r parser.sav --goal consult('%sconstrained'), parseConstrained('%sreparsed.cnll'), halt.'%(corpus, corpus)).split(' '), stdout=sys.stdout, stderr=subprocess.STDOUT).wait()

    ## Train MaltParser on the bootstrapped treebank
    return wp.wholething(f="reparsed.cnll")

Figure 7.9: Python pseudo-code for calling Parasite+ and MALTParser+.

7.6.2 Experimental setup

The corpus from which we extract our dataset is an existing POS-tagged corpus taken from Twitter described in Chapter 5. The corpus was tagged by our Arabic tweets tagger described in Chapter 5 and stemmed by our stemmer described in Chapter 6. We sampled 20K tagged tweets from the corpus to experiment on.

We use two different strategies to create a dependency treebank with a reasonable size. In the first experiment we used just a rule-based parser as described in Section 7.6.3 whereas in the second experiment we used Parasite+ and MALTParser+ in a bootstrapping technique as described in Section 7.6.4. We do the evaluation on tweets that the rule-based parser gives analyses for. The accuracy of other tweets which do not have sensible analyses cannot be tested because it is impossible to say what the right answer would be. Given that, as we mentioned earlier, one of the reasons for using rule-based parser is to eliminate nonsense tweets, it is
reasonable to test only on filtered tweets because the vast majority of other tweets are nonsense and do not have sensible parse trees.

To evaluate the performance of parsers, the parsed output of test data is compared to a silver standard of the test data generated by Parasite+. The accuracy of a parser can be calculated as follows:

\[
\text{accuracy} = \frac{\text{number of correct dependencies}}{\text{total number of dependencies}}
\]  

(7.1)

We tested on a reserved testset of 1K tweets. The evaluation is done with respect to the output of the rule-based chart parser (Parasite+).

### 7.6.3 Using a rule-based parser to build a dependency treebank

In the first experiment (See Figure [7.10]), we use Parasite+ to create a silver treebank (hereafter Treebank\text{RB}_{162K}) from 20K tagged tweets (342K words). Then, we trained MALTParser+ on Treebank\text{RB}_{162K}. It took 20K seconds to construct Treebank\text{RB}_{162K} and the accuracy of MALTParser+ after training on Treebank\text{RB}_{162K} is 68% (see Table 7.1). The speed is a crucial factor to take into account when parsing Arabic tweets since there are millions of tweets that need to be parsed. Therefore, rule-based parsers are not suitable to manipulate Arabic tweets because they are slow (as mentioned in the literature and proved by the first experiment below).

<table>
<thead>
<tr>
<th>Size (words)</th>
<th>Strategy</th>
<th>Accuracy</th>
<th>Training time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>162K</td>
<td>Rule-based parser</td>
<td>68%</td>
<td>20K</td>
</tr>
</tbody>
</table>

Table 7.1: Constructing treebank (Treebank\text{RB}_{162K}) by using Parasite+.
7.6.4 Bootstrapping treebank development

Bootstrapping is used to create labelled training data from large amounts of unlabelled data (Cucerzan and Yarowsky [2002]).

The bootstrapping method (Figure 7.12) begins by using Parasite* to parse an existing POS-tagged corpus to create a small treebank of one thousand Arabic tweets (hereafter Treebank^{RB7957}). Then, MALTParser* is trained on Treebank^{RB7957} which was created by Parasite*, to produce an initial model MALTParser_{0+}, which is then used to parse a much larger set of POS-tagged Arabic tweets. During parsing by MALTParser_{0+}, Parasite* is used as a filter. To use it as a filter, we run Parasite* but only allow hypotheses that correspond to links that were suggested
by MALTParser$_0^+$ so it produces a tree if and only if that tree was produced by MALTParser$_0^+$. As a result, all dependency analyses which do not conform to the defined language rules are omitted. All the resulting legal dependency trees are moved to the training pool to create a larger silver treebank (hereafter $Treebank_{RB7957}^{bootstrapped}$). In Figure 7.13, MALTParser$_0^+$ returns the whole tree for a tweet, but Parasite$^+$ agrees only upon the sub tree in the box, so we only add that subtree to the training data. Finally, MALTParser$^+$ is retrained on $Treebank_{RB7957}^{bootstrapped}$ to produce a new model MALTParser$_1^+$ (Figure 7.11). On a sample of 97 tweets containing 1730 words, Parasite$^+$ produces a complete analysis for 62 (64%) of the tweets covering 1131 words, and assigns a role for 1432 (83%) of all the words. All analyses are legal according to the grammar.

One potential drawback of the bootstrapping technique is that the parser can reinforce its own bad behaviour. However, we control this by parsing a large amount of data and then by using the largest legal fragments according to the grammar for which a well-formed parse is obtained and added to the training pool. In this way, we make sure that the parser will not learn from bad data.

1. Create dependency trees for a seed set of 1K Arabic tweets by using Parasite$^+$ (an initial treebank).
2. Train MaltParser$^+$ (MaltParser$_0^+$) on the initial treebank.
3. Parse 20K Arabic tweets with MaltParser$_0^+$ and filter out all analyses which do not conform to the language rules by using Parasite$^+$.
4. Train a new model (MaltParser$_1^+$) on the silver treebank generated in step 3.
5. Test MaltParser$_1^+$ on the reserved 1K test set (created by Parasite$^+$).

Figure 7.11: Bootstrapping approach.
Figure 7.12: Bootstrapping a treebank by using rule-driven parser and data-driven parser.
In the second experiment, we ran Parasite\textsuperscript{+} on 1K tagged tweets (15K words) to create Treebank\textsuperscript{RB7957} and used this to train an initial interim parser, MALTParser\textsubscript{0+}. It took 1K sec-
Table 7.2: Constructing a treebank ($Treebank_{RB}^{7957\ bootstrapped}$) by bootstrapping.

<table>
<thead>
<tr>
<th>Size (words)</th>
<th>Strategy</th>
<th>Accuracy</th>
<th>Training time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7957</td>
<td>Parasite$^+$</td>
<td>64%</td>
<td>1K</td>
</tr>
<tr>
<td>123K</td>
<td>BootstrappingMALTParser$^+$ and Parasite$^+$</td>
<td>71%</td>
<td>4K</td>
</tr>
</tbody>
</table>

Ons to construct $Treebank_{RB}^{7957}$ (7957 words) and the accuracy of MALTParser$^0$ on the reserved data is 64%. Then, we ran MALTParser$^0$ on 20K Arabic tweets, using Parasite$^+$ to filter out all analyses which do not conform to the language rules. It took 4K seconds to construct $Treebank_{RB}^{7957\ bootstrapped}$ and the accuracy of MALTParser$_1^+$ after training on $Treebank_{RB}^{7957\ bootstrapped}$ is 71% (see Table 7.2). The whole bootstrapping method, including running Parasite$^+$, training MALTParser$^0$ and running MALTParser$^0$ to construct $Treebank_{RB}^{7957\ bootstrapped}$, took 5K seconds to create a reasonable size treebank. Detailed investigation by the authors of Parasite$^+$’s grammar suggests that in over 95% of cases the analyses are indeed legal interpretations of input text. In many cases Prepositional Phrase (PP) and other attachments are wrong, but that is inescapable – even if the attachments were all correct, there is not enough data for the deterministic parser to learn the right attachments, since this takes at least hundreds of millions of words of correctly annotated data. Examples of the output of Parasite$^+$ are listed in Appendix H.

In both experiments, we are able to create a dependency treebank from unlabelled tweets without any manual intervention. Results of experiments show that using MALTParser$^+$ and Parasite$^+$ in a bootstrapping approach to construct a large training set is better than constructing it by using just Parasite$^+$ in terms of the speed of training and constructing the treebank (20K seconds to construct a 162K treebank just using Parasite$^+$, 5K seconds to construct a 123K treebank using Parasite$^+$ to analyse 15K words and MALTParser$^+$ filtered by Parasite$^+$ to analyse 20K Arabic tweets) and the accuracy of parsing.

---

$^3$Personal communication with Prof. Allan Ramsay and Dr. Hanady Ahmed
The results of the tests on our parsing approach achieved 71% unlabelled attachment accuracy. We have compared our parsing accuracy and the size of the treebank with similar work for English tweets and French social media data. Those two parsers yield unlabelled attachment accuracies 80% and 67.8% respectively and the size of our treebank $Treebank_{RB}^{bootstrapped}$ is much larger than their treebanks. Our system is thus more accurate than the system developed by Seddah et al. (2012) for parsing French social media, and is comparable to that reported by Kong et al. (2014) for parsing English tweets. Moreover, we did not use any manual intervention for creating our treebank whereas they used human annotated data. We repeated the same experiment by running the MALTParser$+$ model on 40K Arabic tweets, but they did not give a performance increase which suggests no potential benefit from more bootstrapped data (see Figure 7.14).

7.7 Summary

In this chapter, we have explored two approaches to the task of developing a dependency treebank for Arabic tweets: using rule-based parser only and using rule-based parser and data-
driven parser in a bootstrapping technique. The bootstrapping technique uses a rule-based parser to construct a small treebank of Arabic tweets based on an existing POS-tagged corpus, which then trains a data-driven parser on this treebank to parse a much larger pool of unlabelled Arabic tweets to create a large treebank. The results reported from the evaluation of this approach show that it can make a reasonable size silver dependency treebank \((\text{Treebank}^{RB}_{\text{bootstrapped}})\) that conforms to the rules of the rule-based parser and improves the speed of training and the accuracy of the parsing. This method does not require annotated data or human-supervised training. We also explained our choice of parsing strategy and syntactic representation.
Chapter 8

Conclusion and Future Work

This research aims to shed light on the field of Arabic tweets NLP, to analyse the linguistic structure of Arabic tweets, and to extract the important linguistic features from their text. To achieve this aim, we have studied a number of standard Arabic processing tools and have highlighted their limitations when manipulating Arabic tweets, and we then developed a POS tagger, stemmer and parser aimed specifically at Arabic tweets. We have also automatically created the first dependency treebank for Arabic tweets. This chapter provides a summary of the thesis, which is followed by the main contributions which we have made during this research. Finally, the most promising future directions in this area are discussed.

8.1 Thesis summary

In this thesis, we have focused on the Twitter genre in order to investigate the morphosyntax of Arabic tweets. Our studies were aimed at analysing the linguistic structure of Arabic tweets. This task involves three linked subtasks, tagging, stemming and parsing. To achieve this aim, the following related aspects were covered in this thesis: (1) the challenges of processing Arabic tweets; (2) NLP tools for MSA; (3) the research corpus; (4) NLP tools for Arabic
tweets: POS tagging, stemming and parsing.

We have investigated the challenges of processing Arabic tweets, the sources of ambiguity in natural languages with particular focus on Arabic, and the properties of Arabic tweets and Twitter phenomena. We found that the Arabic tweets text has similar challenges to any ordinary language such as lexical and structural ambiguities as well as more genre-specific issues such as that tweets contain acronyms, slang, that they are not always written maintaining proper spelling and grammar and that they have new elements. All these issues should be taken into consideration when processing Arabic tweets.

Many tools have been developed for processing different components of the Arabic NLP pipeline, with tools such as parsers being trained on MSA corpora in order to be used in an Arabic context. We analysed the behaviour of existing NLP tools for MSA text before developing new tools for Arabic tweets. We have described the POS tagging for Arabic, including different POS tagsets and the state-of-the-art taggers and stemmers. We have also discussed different syntactic parsing frameworks, algorithms and strategies and examples of parsers.

To undertake our experiments, we need a suitable corpus. Despite the growing interest within the NLP community into building Arabic social media corpora by harvesting the web, none of these resources are publicly available yet. They also do not contain all phenomena of tweets as they appear in their original forms in Twitter and they have been built to be used mainly in sentiment analysis. Hence, we built our own corpus which preserves all phenomena of Arabic tweets. We used the Twitter Stream API to retrieve tweets from the Arabian Peninsula. The corpus consists of a million Arabic tweets (10 million tokens).

We experimentally evaluated the performance of state-of-the-art POS taggers for MSA (i.e. AMIRA, MADA and the Stanford tagger) on Arabic tweets. Off-the-shelf taggers present poor success rates on Arabic tweets due to genre noisiness. They failed to deal with Twitter phenomena as shown in the detailed error analysis (Section 4.2.6). As a result, their outcomes when applied to the Arabic tweets text are not useful as inputs to any parser. Therefore, we
have described the steps involved in the development of an Arabic tweets tagger. We introduced approaches for avoiding the noisiness of the genre by combining normalisation and external knowledge to boost the taggers’ performance. Then, we generated training data by applying agreement-based bootstrapping on heterogeneous tagger outputs to retrain the Stanford tagger on it. These combined to improve POS tagging for Arabic tweets. Our techniques yield a very fast and robust POS tagger for Arabic tweets. By using a pool of bootstrapped data combined with the Penn Arabic Treebank to train the augmented version of Stanford tagger, we are able to improve its accuracy from 49% to 74% on Arabic tweets while preserving its speed.

During POS tagging, we used a tagset obtained by unifying the tagsets from AMIRA, MADA and the Stanford tagger. The resulting tagset consists of the main POS tags, both coarse- and fine-grained, in addition to Twitter-specifics tags, and at this point affixes have not been identified. We have decided to use a finer-grained set for all affixes (e.g. pronouns and prepositions) because we believe we can split them accurately so a parser can have more information, which will help it to produce the right analysis. Therefore, we developed a light stemmer which achieves 77.9% accuracy. This result shows improvements over the performance of three well-known stemmers for Arabic.

We have described two approaches to the task of developing a dependency treebank for Arabic tweets: using a rule-based parser, Parasite*, only and using Parasite* and MALTParser+ in a bootstrapping technique. In both approaches, we create a dependency treebank from unlabelled tweets without any manual intervention. Experimental results show that using MALTParser+ and Parasite* in a bootstrapping approach to construct a large training set is better than constructing it by using just Parasite* in terms of both the speed of training and the accuracy of parsing (its accuracy is 71%). In our experiments, we used a dataset from the above POS-tagged and stemmed corpus.

The limitations to the general approaches used in this work are basically that tweets often do not obey any systematic rules at all. They include made up words, compound words
and elongated words of arbitrary lengths, which makes it extremely difficult to stem them accurately using the techniques described here (or, indeed, using any of the other approaches discussed in the literature). The fact that many tweets simply do not have a grammatical structure similarly means it is impossible to either develop a gold standard dataset for parsing or write a program that assigns a sensible syntactic structure. Furthermore, a data-driven parser will assign structures even for tweets that do not have a sensible structure. It may therefore be necessary to supplement the use of robust data-driven parsers by a rule-based parser to detect cases where the tweets are simply ungrammatical nonsense, since it is hard to see how a robust data-driven parser can detect such cases.

8.2 Main contributions

In this project we have made the following main contributions to the field:

1. We have made three state-of-the-art POS taggers for MSA (i.e. AMIRA, MADA and the Stanford tagger) more robust towards noise through pre- and post-processing steps to the tagging when applied to the Arabic tweets text. We also developed the first fast and robust POS tagger for Arabic tweets.

2. We have created the first POS-tagged corpus of Arabic tweets which contains all phenomena of tweets as they appear in their original forms in Twitter.

3. We have developed two approaches to stemming Arabic tweets’ words: a heavy stemmer and a light stemmer. The light stemmer outperforms three well-known stemmers for Arabic in performance. We found that the light stemmer is the most suitable approach for stemming Arabic tweets because it does not use dictionaries, is fast and yields better accuracy compared with the heavy stemmer and MSA stemmers.
4. We have automatically created the first dependency treebank for Arabic tweets and then we trained MALTParser on it to evaluate its usefulness.

To conclude, the hypothesis of this research, which is that developing a POS tagger, stemmer and parser that take into account the properties of Arabic tweets could result in producing efficient and accurate tagging, stemming and parsing solutions for such material compared with the state-of-the-art Arabic taggers, stemmers and parsers, was confirmed by the contributions mentioned above. It is worth mentioning that while these initial results seem promising, they are, of course, substantially behind those which have been achieved with MSA, but this is at least partially because of the extra issues that occur because tweets are noisy, tend to be ungrammatical and have an open vocabulary.

8.3 Future directions

The main focus of this study is to analyse the linguistic structure of Arabic tweets. The required NLP tools to achieve this aim have been developed during this research. As noted, these initial results, while promising, are significantly below the performance of the similar tools on well-formed genres such as newswire. In this Section, we provide the following suggestions for future work:

1. POS tagging: We have done our experiments on a corpus which contains only Arabic tweets from the Arabian Peninsula (Section 4.2.1). We would like to repeat our experiments on a dataset which has no geographical restriction. We expect that the new dataset will contain a high percentage of Romanised Arabic “Arabizi”. Tobaili (2016) showed that 4.9% and 5.7% of Twitter data is Arabizi from Lebanon and Egypt respectively. The new dataset will also contain other Arabic dialects as well as the Gulf dialect. So, we will use approaches similar to Darwish (2013) and Zaidan and Callison-Burch (2014) as
pre-processing steps to convert Arabizi and dialect words to their MSA equivalents.

2. Syntactic parsing: using MALTParser+ and Parasite+ in a bootstrapping approach achieves 71% parsing accuracy on Arabic tweets, which we considered as a promising initial result, and the speed of constructing the Arabic tweets dependency treebank is reasonably fast. Exploring alternative approaches to create training data and further experimental investigations are needed to improve the treebank quality and parsing accuracy over the initial result:

- As we have seen in Section 7.6.4, Parasite+ was used as a filter during parsing so all dependency analyses which do not conform to the defined language rules are omitted. Alternatively, Parasite+ could be used to find the largest legal fragments of tweets instead of omitting them. By doing this, we will increase the training pool size by adding more legal dependency trees without having to parse the whole corpus by using Parasite+.

- Would using a different search strategy give us better results? The version of MALTParser (MALTParser+) that we have used in our experiments is a deterministic parser which uses decision tree ID3 as a machine learning algorithm. It is employed as a deterministic search strategy which makes a single decision at each step. So, decisions cannot be reconstructed later which leads to error propagation. Beam search is another search strategy which has been employed successfully in many NLP tasks (Collins and Roark 2004; Koehn et al. 2003), and achieved a positive effect on their accuracy compared to deterministic approaches (Zhang and Clark 2008). This strategy does not make decisions too early; instead it keeps track of multiple analyses and chooses the analysis which has the overall best score as the right analysis. So, one possible future work is to re-implement MALTParser+ to use beam search strategy instead of simple deterministic search during the parsing
process.

- Would using a different machine learning algorithm give us better results? As mentioned above, the ID3 machine learning algorithm has been used in MALT Parser[^1]. It is fast to train which makes doing experiments easy, but we recognise that ID3 may not be the best machine learning algorithm. Therefore, it is sensible to investigate other machine learning algorithms such as support vector machines and (deep) neural networks (NNS) learning algorithms. The combination of NNS algorithm and beam search strategy has, for instance, been used in SyntaxNet (Andor et al., 2016). Its authors claim that it outperforms all published state-of-the-art dependency parsers in performance. Using a similar approach might give better results.

3. We intend to explore using the tools which we have developed during this research in various NLP applications such as information retrieval and machine translation. Arabic tweets linguistic analyses using these tools have already been started by our research group. Ahmad and Ramsay (2016) use our tagger and stemmer in a task which aims at linking tweets to news.

4. We intend to make the annotated corpus and the dependency treebank available to the research community, thus other researchers can duplicate our experiments and compare our approaches and results with alternatives techniques.

[^1]: Personal communication with Prof. Joakim Nivre, who is one of the MALT Parser developers.
References


Language Resources and Evaluation (LREC), Portoro, Slovenia, pp. 1500–1506. European Language Resources Association (ELRA).


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


Appendix A

An example of Buckwalter Arabic morphological analyser output

<table>
<thead>
<tr>
<th>Input sentence:</th>
<th>شارك عشرة طلاب في الندوة.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Ark E$rp TlAb fy Alndwp .</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WORD $Ark</th>
</tr>
</thead>
<tbody>
<tr>
<td>bw:+$Arak/PV+a/PVSUFF_SUBJ:3MS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WORD E$rp</th>
</tr>
</thead>
<tbody>
<tr>
<td>bw:+Ea$or/NOUN_NUM+ap/NSUFF_FEM_SG+i/CASE_DEF_NOM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WORD TlAb</th>
</tr>
</thead>
<tbody>
<tr>
<td>bw:+TulAb/NOUN+N/CASE_INDEF_NOM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WORD fy</th>
</tr>
</thead>
<tbody>
<tr>
<td>bw:+fiy/PREP+ya/PRON_1S</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WORD Alndwp</th>
</tr>
</thead>
<tbody>
<tr>
<td>bw:Al/DET+nadow/NOUN+ap/NSUFF_FEM_SG+i/CASE_DEF_GEN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WORD .</th>
</tr>
</thead>
<tbody>
<tr>
<td>bw:./PUNC</td>
</tr>
</tbody>
</table>
# Appendix B

## The Penn Arabic Treebank Tagset

<table>
<thead>
<tr>
<th>Tag</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABBREV</td>
<td>IVSUFF_SUBJ:2FS_MOOD:ISJ</td>
<td>POSSE_PRON_3FF</td>
</tr>
<tr>
<td>ADJ</td>
<td>IVSUFF_SUBJ:D_MOOD:I</td>
<td>POSSE_PRON_3FF</td>
</tr>
<tr>
<td>ADV</td>
<td>IVSUFF_SUBJ:D_MOOD:ISJ</td>
<td>POSSE_PRON_3FS</td>
</tr>
<tr>
<td>CONJ</td>
<td>IVSUFF_SUBJ:FP</td>
<td>POSSE_PRON_3MS</td>
</tr>
<tr>
<td>DEM_PRON_F</td>
<td>IVSUFF_SUBJ:MP_MOOD:I</td>
<td>PREP</td>
</tr>
<tr>
<td>DEM_PRON_FD</td>
<td>IVSUFF_SUBJ:MP_MOOD:JS</td>
<td>PRON_1P</td>
</tr>
<tr>
<td>DEM_PRON_FS</td>
<td>NEG_PART</td>
<td>PRON_1S</td>
</tr>
<tr>
<td>DEM_PRON_MD</td>
<td>NO_FUNC</td>
<td>PRON_2FS</td>
</tr>
<tr>
<td>DEM_PRON_MP</td>
<td>NON_ALPHABETIC</td>
<td>PRON_2MP</td>
</tr>
<tr>
<td>DEM_PRON_MS</td>
<td>NON_ARABIC</td>
<td>PRON_2MS</td>
</tr>
<tr>
<td>DET</td>
<td>NOUN</td>
<td>PRON_3D</td>
</tr>
<tr>
<td>EMPHATIC_PARTICLE</td>
<td>NOUN_PROP</td>
<td>PRON_3FP</td>
</tr>
<tr>
<td>EXCEPT_PART</td>
<td>NSUFF_FEM_DU_ACCGEN</td>
<td>PRON_3FS</td>
</tr>
<tr>
<td>FUNC_WORD</td>
<td>NSUFF_FEM_DU_ACCGEN_POSS</td>
<td>PRON_3MP</td>
</tr>
<tr>
<td>FUT</td>
<td>NSUFF_FEM_DU_NOM</td>
<td>PRON_3MS</td>
</tr>
<tr>
<td>INTERJ</td>
<td>PVSUFF_SUBJ:1P</td>
<td>NUM</td>
</tr>
<tr>
<td>INTERROG_PART</td>
<td>PVSUFF_SUBJ:1S</td>
<td>NUM</td>
</tr>
<tr>
<td>IV1P</td>
<td>PVSUFF_SUBJ:D_Mood:1P</td>
<td>NUM</td>
</tr>
<tr>
<td>IV1S</td>
<td>PVSUFF_SUBJ:D_Mood:1S</td>
<td>NUM</td>
</tr>
<tr>
<td>IV2D</td>
<td>NSUFF_MASC_DU_ACCGEN</td>
<td>NUM</td>
</tr>
<tr>
<td>IV2FS</td>
<td>NSUFF_MASC_DU_ACCGEN_POSS</td>
<td>NUM</td>
</tr>
<tr>
<td>IV2MP</td>
<td>NSUFF_MASC_DU_NOM</td>
<td>NUM</td>
</tr>
<tr>
<td>IV2MS</td>
<td>NSUFF_MASC_DU_NOM_POSS</td>
<td>NUM</td>
</tr>
<tr>
<td>IV3FD</td>
<td>NSUFF_MASC_PL_ACCGEN</td>
<td>NUM</td>
</tr>
<tr>
<td>IV3FF</td>
<td>NSUFF_MASC_PL_NOM</td>
<td>NUM</td>
</tr>
<tr>
<td>IV3FS</td>
<td>NSUFF_MASC_PL_NOM_POSS</td>
<td>NUM</td>
</tr>
<tr>
<td>IV3MD</td>
<td>NSUFF_MASC_SZ_ACC_INDEF</td>
<td>NUM</td>
</tr>
<tr>
<td>IV3MP</td>
<td>PVSUFF_SUBJ:3PP</td>
<td>NUM</td>
</tr>
<tr>
<td>IV3MS</td>
<td>PVSUFF_SUBJ:3FS</td>
<td>NUM</td>
</tr>
<tr>
<td>IVSUFF_DO:1P</td>
<td>PVSUFF_SUBJ:3PF</td>
<td>NUM</td>
</tr>
<tr>
<td>IVSUFF_DO:1S</td>
<td>PVSUFF_SUBJ:3PF</td>
<td>NUM</td>
</tr>
<tr>
<td>IVSUFF_DO:2MP</td>
<td>PVSUFF_SUBJ:3PF</td>
<td>NUM</td>
</tr>
<tr>
<td>IVSUFF_DO:2MS</td>
<td>PVSUFF_SUBJ:3PF</td>
<td>NUM</td>
</tr>
<tr>
<td>IVSUFF_DO:3D</td>
<td>PVSUFF_SUBJ:3PF</td>
<td>NUM</td>
</tr>
<tr>
<td>IVSUFF_DO:3FS</td>
<td>PVSUFF_SUBJ:3PF</td>
<td>NUM</td>
</tr>
<tr>
<td>IVSUFF_DO:3MP</td>
<td>PVSUFF_SUBJ:3PF</td>
<td>NUM</td>
</tr>
<tr>
<td>IVSUFF_DO:3MS</td>
<td>RESULT_CLAUSE_PARTICL</td>
<td>NUM</td>
</tr>
</tbody>
</table>

Source: [Maamouri et al. ](maamouri2004)
## Appendix C

**MADA tagsets**

<table>
<thead>
<tr>
<th>Part-of-speech</th>
<th>LABEL</th>
<th>MADA 3.2</th>
<th>MADA 2.32</th>
<th>PATB Equivalent</th>
<th>CATiB Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nouns</td>
<td>noun</td>
<td>N</td>
<td>NN / NNS</td>
<td>NOM</td>
<td></td>
</tr>
<tr>
<td>Number Words</td>
<td>noun_num</td>
<td>N</td>
<td>NN / NNS</td>
<td>NOM</td>
<td></td>
</tr>
<tr>
<td>Proper Nouns</td>
<td>noun_prop</td>
<td>PN</td>
<td>NNP / NNPS</td>
<td>PROP</td>
<td></td>
</tr>
<tr>
<td>Adjectives</td>
<td>adj</td>
<td>AJ</td>
<td>JJ</td>
<td>NOM</td>
<td></td>
</tr>
<tr>
<td>Adverbs</td>
<td>adv</td>
<td>AV</td>
<td>RB</td>
<td>NOM</td>
<td></td>
</tr>
<tr>
<td>Pronouns</td>
<td>pron</td>
<td>PRO</td>
<td>PRP</td>
<td>NOM</td>
<td></td>
</tr>
<tr>
<td>Verbs</td>
<td>verb</td>
<td>V</td>
<td>VBN / VBP / VBD</td>
<td>VRB or VRB-PASS</td>
<td></td>
</tr>
<tr>
<td>Particles</td>
<td>part</td>
<td>P</td>
<td>IN</td>
<td>PRT</td>
<td></td>
</tr>
<tr>
<td>Prepositions</td>
<td>prep</td>
<td>P</td>
<td>IN</td>
<td>PRT</td>
<td></td>
</tr>
<tr>
<td>Abbreviations</td>
<td>abbrev</td>
<td>AB</td>
<td>NN</td>
<td>PROP</td>
<td></td>
</tr>
<tr>
<td>Punctuation</td>
<td>punc</td>
<td>PX</td>
<td>PUNC</td>
<td>PNX</td>
<td></td>
</tr>
<tr>
<td>Conjunctions</td>
<td>conj</td>
<td>C</td>
<td>CC</td>
<td>PRT</td>
<td></td>
</tr>
<tr>
<td>Interjections</td>
<td>interj</td>
<td>IJ</td>
<td>UH</td>
<td>PRT</td>
<td></td>
</tr>
<tr>
<td>Digital Numbers</td>
<td>digit</td>
<td>NUM</td>
<td>CD</td>
<td>NOM</td>
<td></td>
</tr>
<tr>
<td>Foreign/Latin</td>
<td>latin</td>
<td>F</td>
<td>IN</td>
<td>NOM</td>
<td></td>
</tr>
</tbody>
</table>

Source: [Habash et al., 2009](#)
Appendix D

Twitter Stream API (Java implementation)

```java
package stream;
import twitter4j.*;
import twitter4j.FilterQuery;
import twitter4j.Status;
import twitter4j.StatusDeletionNotice;
import twitter4j.StatusListener;
import twitter4j.TwitterStream;
import twitter4j.TwitterStreamFactory;
import twitter4j.conf.ConfigurationBuilder;
import java.io.IOException;
import java.util.Scanner;

public class Stream {

    public static void main(String[] args) throws TwitterException, IOException {

        ConfigurationBuilder cb = new ConfigurationBuilder();
        cb.setDebugEnabled(true);
        // application’s API keys are used to authenticate requests to the Twitter Platform.
        // shouldn’t shared with others
        // I used my Twitter account to create them
        cb.setOAuthConsumerKey("****");
        cb.setOAuthConsumerSecret("****");
        cb.setOAuthAccessToken("****");
        cb.setOAuthAccessTokenSecret("****");
        TwitterStream twitterStream = new TwitterStreamFactory(cb.build()).getInstance();

        // System.out.println("Please enter numbers of tweets you want to retrieve or -1 for continuous tweets stream");
    }
}
```
APPENDIX D. TWITTER STREAM API (JAVA IMPLEMENTATION)  170

// Scanner scan = new Scanner(System.in);
// int nofTweets = scan.nextInt();

StatusListener listener = new StatusListener() {
    WriteToXmlFile wx = new WriteToXmlFile();

    public void onException(Exception arg0) {
        // TODO Auto-generated method stub
    }

    public void onDeletionNotice(StatusDeletionNotice arg0) {
        // TODO Auto-generated method stub
    }

    public void onScrubGeo(long arg0, long arg1) {
        // TODO Auto-generated method stub
    }

    int count=1;
    public void onStatus(Status status) {
        // if (nofTweets != -1)
        // if (count > nofTweets) System.exit(0);
        // write tweet to xml file

        wx.writeToXmlFile(status);
        System.out.println("No of tweets in the file "+count);
        count++;
    }

    public void onTrackLimitationNotice(int arg0) {
        // TODO Auto-generated method stub
        System.out.println("onTrackLimitationNotice "+"\n");
    }

    public void onStallWarning(StallWarning arg0) {
        // TODO Auto-generated method stub
        System.out.println("onStallWarning" +"\n");
    }
    };

    // filter Twitter stream by setting desired parameters
    FilterQuery fq = new FilterQuery();
    //GCC region
    double lat =24.73;
    double longitude = 46.71;
double lat1 = lat - 5;
double longitude1 = longitude - 10;
double lat2 = lat + 5;
double longitude2 = longitude + 10;
twitterStream.addListener(listener);
double[][] bb = {{longitude1, lat1}, {longitude2, lat2}};

fq.locations(bb);
// twitterStream.addListener(listener); //without filtering

twitterStream.filter(fq);
// twitterStream.sample(); //without filtering

Appendix E

Pre- and Post-processing code

//detect Twitter specific items as a pre-processing step
Scanner scanner = new Scanner(System.in);
System.out.print("Enter input file path ");
String inputFile = scanner.nextLine();
System.out.print("Enter output folder path ");
String outputFolder = scanner.nextLine();
try (BufferedReader br1 = new BufferedReader(new FileReader(inputFile)))
{
  String sCurrentLine1;
  while ((sCurrentLine1 = br1.readLine()) != null) {

    File file = new File(outputFolder +"/
    Preprocessing.txt");
    File file2 = new File(outputFolder+"/Token.txt");
    File file3= new File(outputFolder+"/Tag.txt");

    FileWriter fw = new FileWriter(file,true);
    BufferedWriter bw = new BufferedWriter(fw);
    FileWriter fw2 = new FileWriter(file2,true);
    BufferedWriter bw2 = new BufferedWriter(fw2);
    FileWriter fw3 = new FileWriter(file3,true);
    BufferedWriter bw3 = new BufferedWriter(fw3);

    //detect all emojis
    String emo Regex = "\\u1F60-\\u1F64|\\u2702-\\u27B0|\\u1F68-\\u1F6C|\\u1F30-\\u1F70|\\u2600-\\u26FF";
    Matcher matcher = Pattern.compile(emo Regex).matcher(sCurrentLine1);
    String original = sCurrentLine1;

    //replace links and urls by "LINK"
    if (sCurrentLine1.matches("(.*)http(.*)")
    {
APPENDIX E. PRE- AND POST-PROCESSING CODE

sCurrentLine1 = sCurrentLine1.replaceAll("http \S*","LINK");

// replace user name by "USERN"
if (sCurrentLine1.matches("(.*)@[a-zA-Z0-9_]+(.*)")
    ) sCurrentLine1 = sCurrentLine1.replaceAll("@[a-zA-Z0-9_]+","@USERN");

// detect all English words
String EngWord_regex= "[a-zA-Z0-9_]+";
Matcher matcherE = Pattern.compile(EngWord_regex).matcher(sCurrentLine1);
String[] latin = sCurrentLine1.split(" ");
sCurrentLine1="";
for (int i=0;i<latin.length;i++)
{
    // replace English word by "LATIN" if it is not "LINK" or "USERN"
    if (!(latin[i].equals("LINK")||latin[i].equals("USERN")))
        if (\b\w+\b).matches("\\b\\w+\\b")
            latin[i]= "LATIN";
    sCurrentLine1= sCurrentLine1+" "+latin[i];
}

// replace emoji by "EMOJ"
matcher = Pattern.compile(emoj_regex).matcher(sCurrentLine1);
while (matcher.find())
{
    sCurrentLine1 = sCurrentLine1.replace(matcher.group(), "EMOJ");
}

// replace emoticons by "EMOT". Below just examples of them
sCurrentLine1 = sCurrentLine1.replaceAll("(D:\|3>|;:-D|:-D|:\)|:\(=|O.o|:-\)|:\)|:\(|;\)|:\(|:>|:<)\)|","EMOT");

// write a tweet after replacing Twitter specific items (if any)
bw.write(sCurrentLine1);

String[] words = original.split(" ");
String[] replacTwitterSpe = sCurrentLine1.split(" ");
System.out.println(replacTwitterSpe.length);
bw.newLine();
bw.close();
for (int i=0;i<words.length;i++)
{
    if ((replacTwitterSpe[i].matches("\\@"))|(
            replacTwitterSpe[i].matches("LATIN")) |(
            replacTwitterSpe[i].matches("LINK")) |(
            replacTwitterSpe[i].matches("EMOJ")|
            replacTwitterSpe[i].matches("REP")|(
    {
        //save the original token
        bw2.write( words[i]+",");
        //save its tag
        bw3.write(replacTwitterSpe[i]+",");
    }
    bw2.newLine();bw3.newLine();bw2.close();
    bw3.close();
}
//dealing with elongated words
try {
    try (BufferedReader br1 = new BufferedReader(new FileReader("C:/Users/albogamf/Desktop/Preprocessing2.txt"));
        BufferedReader br1 = new BufferedReader(new FileReader(outputFolder+"/Preprocessing.txt"));
    {
        String sCurrentLine1;
        String sCurrentLine2;
        // File file = new File("C:/Users/albogamf/Desktop/Preprocessing3.txt");
        File file = new File(outputFolder+"/Preprocessing.txt");
        FileWriter fw = new FileWriter(file,true);
        BufferedWriter bw = new BufferedWriter(fw);
        while ((sCurrentLine1 = br1.readLine()) != null) {
            String[] words = sCurrentLine1.split(" ");
            for (int i=0;i<words.length;i++)
            {
                // read file if only the word contains at least two repeated chars
                int j = 0;
                boolean found=false;
                for (j = 0; j < words[i].length()-1; j++) {

if (words[i].charAt(j) == words[i].charAt(j+1)) {
    found=true;
    break;
}
if(found) {
    BufferedReader br2 = new BufferedReader(new FileReader(outputFolder+"/cnn-arabic-wordlist.txt");
    Boolean exist=false;
    while ((sCurrentLine2 = br2.readLine()) != null) {
        sCurrentLine2= sCurrentLine2.replaceAll("\s+"," ").replaceAll("^\s+|\s+$", "");
        if (words[i].equals(sCurrentLine2))
            exist=true; break;
    }
    if (exist)
        {bw.write(words[i]+" ");
    else {bw.write(words[i].replaceAll("(.)*\1{1,}", "$1")+" "); //System.out.
        println(words[i].replaceAll("(.)*\1{1,}", "$1")+" ");
    }
    }else {bw.write(words[i]+" ");
    }
    bw.newLine();
    bw.close();
//dealing with character deletion and slang words
try {
    BufferedReader br1 = new BufferedReader(new FileReader(outputFolder+"/Preprocessing.txt");
    String sCurrentLine1;
    String sCurrentLine2;
    File file = new File(outputFolder+"/Preprocessing5.txt"); // Ready for tagging!
    FileWriter fw = new FileWriter(file, true);
    BufferedWriter bw = new BufferedWriter(fw);
    while ((sCurrentLine1 = br1.readLine()) != null) {
        String[] words = sCurrentLine1.split(" ");
    }
for (int i=0;i<words.length;i++)
{
    switch (words[i]) {
    // dealing with character deletion.
    // characters and words have written
    // by using transliteration.
    case "y":words[i]="yA";break;
    case "E":words[i]="ElY";break;
    case "f":words[i]="fy";break;
    case "m":words[i]="mA";break;
    // dealing with slang words
    case "Ally":words[i]="Al*y";break;
    case "Aly":words[i]="Al*y";break;
    case "bs":words[i]="lkn";break;
    case "mw":words[i]="lys";break;
    case "m$":words[i]="lys";break;
    case "w$":words[i]="lmA*A";break;
    case "ly$":words[i]="lmA*A";break;
    case "EsAn":words[i]="l>n";break;
    case "ElsAn":words[i]="l>n";break;
    case "mdry":words[i]="lAdry";break;
    case "zy":words[i]="mvl";break;
    case "Ayh":words[i]="nEm";break;
    case "Ay$":words[i]="mA*A";break;
    case "lyn":words[i]="HtY";break;
    case "dh":words[i]="h*A";break;
    case "$lwn":words[i]="kyf";break;
    }
    bw.write(words[i]+" ");
    }
    bw.newLine();
}
bw.close();

// dealing with hashtags
try (BufferedReader br1 = new BufferedReader(new FileReader(outputFolder+"/Preprocessing.txt")))
{
    String sCurrentLine1;
    while ((sCurrentLine1 = br1.readLine()) != null) {
        File file = new File(outputFolder+"/Hash.txt");
        FileWriter fw = new FileWriter(file, true);
        fw.write(sCurrentLine1 + " ");
        fw.newLine();
        fw.close();
    }
}
BufferedWriter bw = new BufferedWriter(
   fw);
String[] original = sCurrentLine1.split(" ");
int c=0;
int j=0;
int i=0;
int counter=0;
int temp=0;
int t=0;
for ( i=c;i<original.length;i++){
temp=0;
if (original[i].matches("#")){
   if ((counter >0)) {temp=i-(t);}else {temp=i;}
bw.write(Integer.toString(temp)+",");
j=1;t=t+1;
counter++;
   for (int k=i+2;k<original.length;k+=2)
      if (original[k].matches("_"))(j++;t++;
   else { ; bw.write(Integer.toString(j)+" ");j=0;c=k;break;}
}counter=0;
if ((i+2 >= original.length)&& j>0) bw.write(
   Integer.toString(j)+" ");
bw.newLine();
bw.close();

//call the Stanford tagger to tag the corpus
MaxentTagger tagger = new MaxentTagger(
   "taggers/arabic2.tagger");
try {
   BufferedReader br1 = new BufferedReader(new FileReader(
     outputFolder+"/Preprocessing.txt");
   String sCurrentLine1;
   while ((sCurrentLine1 = br1.readLine()) != null) {
      sCurrentLine1= sCurrentLine1.replaceAll("\s+"," ");
      .replaceAll("\s+\|\s+"," ");
      String tagged = tagger.tagString(sCurrentLine1);
      File file = new File(outputFolder+"/TaggedCorpus.
      txt");
      FileWriter fw = new FileWriter(file,true);
BufferedWriter bw = new BufferedWriter(fw);
bw.write(tagged);
bw.newLine();
bw.close();
}

// Post-processing
// replace all original tokens

try {
    BufferedReader br1 = new BufferedReader(new FileReader(outputFolder+"/TaggedCorpus.txt"));
    BufferedReader br2 = new BufferedReader(new FileReader(outputFolder+"/Token.txt"));
    BufferedReader br3 = new BufferedReader(new FileReader(outputFolder+"/Tag.txt"));

    String sCurrentLine1;
    String sCurrentLine2;
    String sCurrentLine3;

    while ((sCurrentLine1 = br1.readLine()) != null) {
        File file = new File(outputFolder+"/TaggedWithToekns.txt");
        FileWriter fw = new FileWriter(file, true);
        BufferedWriter bw = new BufferedWriter(fw);
        String[] taggedWords = sCurrentLine1.split(" ");

        if ((sCurrentLine2 = br2.readLine()) != null) {
            sCurrentLine3 = br3.readLine();
            if (sCurrentLine2.trim().equals("")) {
                bw.write(sCurrentLine1);
                bw.newLine();
            } else {
                String[] Tokens = sCurrentLine2.split(",");
                String[] Tags = sCurrentLine3.split(",");

                int j=0; int k=0;
                String str;
                for (int i=0;i<Tags.length;i++){
                    for ( j=k; j<taggedWords.length;j++)
                    {
                        // do something
                    }
                }
            }
            bw.write(sCurrentLine1);
            bw.newLine();
        } else {
            PrintWriter bw = new PrintWriter(outputFolder+"/TaggedWithToekns.txt");
            bw.write(sCurrentLine1);
            bw.newLine();
        }
    }
}
if ((taggedWords[j].contains(Tags[i]))) {
    taggedWords[j]=Tokens[i]+","+Tags[i];
    str=Tokens[i];
    if (Tags[i].equals("USERN"))
        if (str.length()>0) if (Character.
            toString(str.charAt(str.length()
            -1)).equals(":"))taggedWords[j
            -1]="@","+"RET";
        else if (i==0)taggedWords[j-1]="@
            "+","+"REP";else taggedWords[j
            -1]="@","+"MEN";
    k =j+1;
    break; }

if ((taggedWords[j].contains(Tags[i]))) {
    taggedWords[j]=Tokens[i]+","+Tags[i];k =j
    +1;break; }
}

for (int d=0; d<taggedWords.length;d++)
{
    bw.write(taggedWords[d]+" ");
}
bw.newLine();

bw.close();

//dealing with named entities as a post-processing step
try
{
    BufferedReader br1 = new BufferedReader(new FileReader(outputFolder
    +"/TaggedWithToekns.txt");

    String sCurrentLine1;
    String sCurrentLine2;

    while ((sCurrentLine1 = br1.readLine()) != null) {

        //File file = new File("C:/Users/albogamf/
        Desktop/tagger/TaggedWithNNP.txt");
        File file = new File(outputFolder+
            "+/TaggedWithToekns.txt");
        // File file = new File("C:/Users/Fahad/Dropbox
        //EXperiments/Feb2015/AMIRA/Tagging/
TaggedWithHashTokens1.txt);
FileWriter fw = new FileWriter(file, true);
    BufferedWriter bw = new BufferedWriter(fw);
sCurrentLine1= sCurrentLine1.replaceAll("\\s+"," ").replaceAll("^\\s+|\\s+$", "");
String[] taggedWords = sCurrentLine1.split(" ");
boolean found=false;
for (int d=0; d<taggedWords.length;d++)
{
    BufferedReader br2 = new BufferedReader(new
    FileReader(outputFolder+"/ProperNames.txt ");
    found=false;
    String[] word = taggedWords[d].split(",");
    while ((sCurrentLine2 = br2.readLine()) != null)
    {
        sCurrentLine2= sCurrentLine2.replaceAll("\\s +"," ").replaceAll("^\\s+|\\s+$", "");
        if ((word[0]).equals(sCurrentLine2))
        {found=true; break;}
    }
    if (found) {bw.write(word[0]+","+"NNP"+" ");
            }
    else bw.write(taggedWords[d]+" ");
}
    bw.newLine();
    bw.close();
}
//dealing English words
//call the Stanford tagger for English to tag English words
try {
    BufferedReader br1 = new BufferedReader(new FileReader(
    outputFolder+"/TaggedWithNNP.txt");
    String sCurrentLine1;
    MaxentTagger taggerE = new MaxentTagger(
    "taggers/english.tagger");
    while ((sCurrentLine1 = br1.readLine()) != null) {
    File file = new File(outputFolder+"/TaggedE.txt");
    FileWriter fw = new FileWriter(file, true);
    BufferedReader bw = new BufferedWriter(fw);
sCurrentLine1= sCurrentLine1.replaceAll("\\s+"," ").replaceAll("^\\s+|\\s+$", "");
    String[] taggedWords = sCurrentLine1.split(" ");
    boolean found=false;
    int length;
    for (int d=0; ditaggedWords.length;d++)
length = 0;
found = false;
String[] word = taggedWords[d].split(",");

if ((word[1]).equals("LATIN")) {
    length = word[0].length();
    String tagged = taggerE.tagString(word[0]);
    bw.write(tagged);
} else { bw.write(taggedWords[d] + " "); }

bw.newLine();
bw.close();
Appendix F

Stemmer source code

```python
import re

"""
(?P<name>xyna*b) will match any sequence like "xyaaaab" and call it "name"

(?{name}(abcd)|(pqrs)): if some group called "name" has already been
matched, then this
will match "abcd", otherwise it will match "pqrs"
"""

patterns = {
"ART": "Al|",
"CONJ": "w|b",
"NSTEM": ".{3,}?",
"VSTEM": ".{2,}?",
"PRON": "k|kmA|km|kn|h(A+)|hm(A*)|hn|hm|nA|y",
"VPRON": "PRON|NY",
"NY": "ny",
"PREP": "b|k|l",
"FUT": "s",
"TNS_PRES": "(?P<ON>0|n)|(?P<AY>A|y)|(?P<t>t)",
"TNS_PAST": "(?P<past>)",
"TENSE": "(?FUT)?(TNS_PRES)",
"PERSON": "(?(ON)AGR-ON|(?(AY)AGR-AY|(?t)AGR-T|(?(past)AGR-PAST)))",
"AGR-ON": "",
"AGR-AY": "(A|An|wA|wn|n)",
"AGR-T": "(yn|An|wn|n)",
"AGR-PAST": "(tmA|tm|tn|t|A|wA|n)",
"AGREE": "wn|yn|p|An|At",
"NOUN": ":CONJ|(PREP)|(ART)\{P<stem>NSTEM\}AGREE\{PRON\}",
"VERB": "\{CONJ\}\{TENSE\}\{P<stem>VSTEM\}\{PERSON\}\{VPRON\}"}

"""

For each pattern, replace anything that is in uppercase, possibly with
digits, and is in the set of patterns, by the expansion of its value in
the set, i.e. replace "CONJ" in "\{CONJ\}\{DEFART\}\{STEM\}\{AGREE\}\{PRON\}"
by "w|B". Do this recursively (e.g. VERBs contain TENSEs, but TENSEs
"""

182
APPENDIX F. STEMMER SOURCE CODE

```python
contain FUTs and TNS1s)

def expandpattern(p, patterns=patterns):
    for i in re.compile("([A-Z0-9-]+)").finditer(p):
        i = i.group(0)
        if i in patterns:
            p = p.replace(i, "(?P<%s>%s)"%(i.replace("-", ""),
                                      expandpattern(patterns[i], patterns)))
    return p

Do that for all your patterns.

Once done it, replace the values by compiled regexes that have `^` and `$`
to make sure that they match the whole string

```python
def expandpatterns(patterns=patterns):
    epatterns = {p:expandpattern(p, patterns) for p in patterns}
    for p in epatterns:
        try:
            epatterns[p] = re.compile("^"+epatterns[p]+"$")
        except:
            pass
    return epatterns

Now just lookup the tag in the set of patterns and see if it matches your
string

```python
>>> m = lookupword("wsyktbwnh", "VERB")
>>> m = lookupword("wsyktbwnh", "NOUN")

This will give us a "match object". We can ask it what its stem is

```python
>>> m.group("stem")
'ktb'

Or you we ask it what all its groups are:

```python
>> m.groups()
('w', 'sy', 's', 'y', 'ktb', 'wn', 'h')

```python
def lookupword(w, tag, expandedpatterns=expandpatterns()):
    return expandedpatterns[tag].match(w)
```
Appendix G

Source code for calling Parasite$^+$ and MaltParser$^+$ to build a silver treebank

```python
#!/usr/bin/python

import re, sys, os, subprocess, shutil
from useful import *
import a2bw, tidytraining, tidycorpus
reload(a2bw)
reload(tidycorpus)
reload(tidytraining)
import test_wholeprogram as wp
reload(wp)

corpus = "TaggedTweets"

def conll2prolog(corpus="TaggedTweets", N1=-1):
    tidycorpus.tidyCorpus("%s.txt"%(corpus), out="%stidy.txt"%(corpus))
    """ Now turn that into N Prolog clauses """
    tidycorpus.corpus2prolog("%stidy.txt"%(corpus), out="%ssentences.pl"%(corpus, corpus), N=N1)
    """ Split sentences into manageable segments """
    process = "'/usr/local/bin/sicstus -r parser.sav --goal consult('%ssentences.pl'),ppandhalt('%ss segments')."%(corpus, corpus)
    sys.stdout.flush()
    subprocess.Popen(process.split(" "), stdout=sys.stdout, stderr=subprocess.STDOUT).wait()

"""
Use the rule-based parser to create the initial treebank
"""

def makeBaseTreeBank(corpus):
    sys.stdout.flush()
    subprocess.Popen("'/usr/local/bin/sicstus -r parser.sav --goal consult('%s segments'),parse2conll('%sbasetreebank.cnll'),halt.'%(corpus, corpus)).split(" "), stdout=sys.stdout, stderr=subprocess.STDOUT).
```
We sometimes create CNLL files with the wrong offsets. This sorts that out.

def tidyCNLL(corpus):
    tidytraining.tidytraining("%s.cnll"%(corpus), "%s.tidy.cnll"%(corpus))
    os.remove("%s.cnll"%(corpus))
    os.rename("%s.tidy.cnll"%(corpus), "%s.cnll"%(corpus))

Read TaggedTweets file, convert it to a list of [word, tag] pairs to be fed
directly to the parser.

def text2conll(ifile):
    rawtext = []
    for l in codecs.open(ifile, encoding="utf-8").read().split("\n"):
        l1 = []
        for x in l.split(" "):
            x = x.split(",")
            if len(x) == 2:
                x[0] = a2bw.convert(x[0]).strip()
                try:
                    x[1] = str(x[1]).strip()
                except:
                    print x[1].strip()
                    raise Exception("AAA")
                l1.append(x)
            rawtext.append(l1)
    return rawtext

def train(corpus):
    makeBaseTreeBank(corpus)
    tidyCNLL('%sbasetreebank'%(corpus))
    return wp.wholething(f="%sbasetreebank.cnll"%(corpus)

def makeCompleteTreeBank(corpus, parser, outsink=sys.stdout, N2=-1):
    alltrees = []
    conll = text2conll("%stidy.txt"%(corpus))
    for sentence in conll:
        print "%s %.3f"%(len(alltrees), len(conll), float(len(alltrees))
        /float(len(conll)))
        try:
            if len(sentence) > 2:
                tree = parser.parse(sentence)
                alltrees.append(tree)
                if len(alltrees) == N2:
                    break
        except:
            pass
    with safeout(outsink) as write:
for i, tree in enumerate(alltrees):
s = u"constrained(%s, [%"%(i
sep = u"
offset = tree.leaves[0].position+1
for w in tree.leaves:
    try:
        wroot = u"%s"%(w.root).replace("", "\")
        if wroot == u"":
            wroot = "???"
    except:
        wroot = "???"
    try:
        wform = u"%s"%(w.form).replace("", "\")
        if wform == u"":
            wform = "???"
    except:
        wform = "???"
    s += u"%s('%s', '%s', '%s')"%(sep, wroot, wform, w.tag)
    sep = u", 
    s += u"], [%"%(offset
    sep= u""
    for link in tree.parsed.values():
        s += u"\%s\\%s\"%(sep, i, link.dtr+1, link.hd +1)
    sep = u", 
    s += u"\n\n"
if not tidycorpus.NONARABIC.search(s):
    try:
        write(str(s))
    except:
        write(s)
def retrain(corpus, parser, N2=-1):
    makeCompleteTreeBank(corpus, parser, outsink="%sconstrained.pl"%(corpus , N2=N2)
sys.stdout.flush()
subprocess.Popen("/usr/local/bin/sicstus -r parser.sav --goal consult ('%sconstrained'),parseConstrained('%sreparsed.cnll'),halt."%(corpus, corpus)).split(" "), stdout=sys.stdout, stderr=subprocess.STDOUT).wait()
tidyCNLL("%sreparsed"%(corpus)
return wp.wholething(f="reparsed.cnll")

def doItAll(corpus="TaggedTweets", N1=1000, N2=20000):
    ***
    Convert from tagged format to cnll and Prolog, break
    into N1 sentences and break these into segments
    ***
    T0 = now()
APPENDIX G. TREEBANK DEVELOPMENT SOURCE CODE

print box("START TIME %s"%(T0))
print box("./end2end.py corpus=%s N1=%s N2=%s"%(corpus, N1, N2))
conll2prolog(corpus=corpus, N1=N1)

***
Use the rule-based parser to make a treebank from the
segments we’ve just made and train a MALTParser+ on this
***
T1 = now()
parser = train(corpus)
T2 = now()
print box("Time for first round of training %s"%(timediff(T2, T1)))

***
Parse N2 sentences using this parser, make a new
treebank out of the examples that the rule-based parser (Parasite+)
likes, train a new parser on this.
***
parser = retrain(corpus, parser, N2=N2)
T3 = now()
print box("Time for retraining %s"%(timediff(T3, T2)))
TN = now()
print box("END TIME %s, total time %s"%(TN, timediff(TN, T0)))
return parser

if "end2end.py" in sys.argv[0]:
  N1=1000
  N2=1000
  corpus = "TaggedTweets"
for flag in sys.argv[1:]:
  flag = flag.split("=")
  if len(flag) > 1:
    [flag, value] = flag
    print flag, value
    if flag=="N1":
      N1 = int(value)
    elif flag=="N2":
      N2 = int(value)
    elif flag=="corpus":
      corpus = value
  else:
    print "No such flag %s"%(flag)
    sys.exit()
else:
  flag = flag[0]
  if "-redirect".startswith(flag):
    try:
      sys.stdout = open("end2end.py corpus=%s N1=%s N2=%s.log"%(corpus, N1, N2), "w")
      except Exception as e:
        print e
        sys.exit()
  elif "-help".startswith(flag):
print "end2end N1=<segments for base treebank> N2=<trees for retraining> corpus=<TaggedTweets or similar> - redirect"
sys.exit()
else:
    print "No such flag %s"%(flag)
sys.exit()

doItAll(corpus=corpus, N1=N1, N2=N2)
Appendix H

Examples of Parasite+ annotation

1. @ RooryG t slm y yA qlb y
   ت سلم ي يا قلب ي

   (i39
   + [@#0,
     ((user , RooryG#1)),
     ((topic
       , [slm:#3,
         {{tns , t:#2}},
         {{subj , zero#4}},
         {{dobj , y:#4}},
         {{voc
           , [yA:#5,
             {generalcomment , [qlb:#7]}]}]})))
   )

2. $} mn Al <nsAnyt >n t HAwl >n t Hqq $y} AF mn ]nsAnyt k Alty t xS Al gyr . . .
   HynmA twn sdyq AF jyd AF . . . ♥ . . . http://t.co/inRKySur2Q
   التي ت خص ال غير . . . أن ت حاول أن ت حقق شئ أ من إنسانيت ل شئ من ال إنسانية
   حينما تكون صديق أ جيد أ . . . . . . . . . ♥ . . . http://t.co/inRKySur2Q

   (i1434
   + [@$):#0,
     ((ppmod
       , [mn:#1,
         {{comp , [<nsAnyp:#2]]}})),
     ((pred
       , [HAwl:#7,
         {{tns , t:#6}},
         {{subj , zero#5}}),
         {{voc
           , [yA:#5,
             {generalcomment , [qlb:#7]}]}]})))
   )
APPENDIX H. EXAMPLES OF PARASITE* ANNOTATION

3. mfy$ AHly mn AHsAs An k t bqA brp mSr bjd

مقين احلي من احساس أن لت بقا برة مصر بعيد

}191
    + [bqA:#7,
        {(tns , t:#6)},
        {(subj
            , [mfy$:#0,
                {(nmod

}}
APPENDIX H. EXAMPLES OF PARASITE* ANNOTATION

4. @ alabbas75 @ DrA_Farouk235 h*A Hq w nHn b Hmd Allh k mSr yn n Hb Allh w rswl h mHmd w kl nbyAQ w Al rsl w ntAjr mE Allh w lEn Allh xwArj Al ESr

@ alabbas75 @ DrA_Farouk235 مصر ين حب الله هذا حق و تخن ب حمد الله ل 255 و رسول ح محمد وكل نبي و ال رسول و نتاجر مع الله و لن الله خوارة ال عصر

(i5 + [@#2, {mentioned, DrA_Farouk235#3}])

(i2 + [h*A:#1])

(i1428
  + [w:#11,
    ((conj1
      , [Hb:#9,
        {{tns, n:#8}},
        {{subj, [mSr:#7]}},
        {{dobj, Allh:#10}}),
      , [w:#15,
        ((conj1, mHmd:#14),
          ((conj2
            , [kl:#17])),
          ((conj2, [rsl:#19]))))},
    ((ppmod
      , [b:#2,
        {{comp
          , [Hmd:#3,
            {{nmod, [Allh:#5]}},
          , [nmod, [Allh:#5]]}}]))]))


APPENDIX H. EXAMPLES OF PARASITE* ANNOTATION

(i10
+ [w:#0,
   (conj0
    , [ntAjr:#1,
      (subj , zero#0),
      (ppmod , [mE:#3])])])})

(i26
+ [w:#0,
   (conj0
    , [lEn:#1,
      (subj , [Allh:#3]),
      (dobj , [ESr:#4])])])})})

5. @ NI_AF2013 hlAQ
   @ NI_AF2013 هلال، 2013

   (i4 + [@#0, {user, NI_AF2013#1}, {topic, hlAQ:#2}])

6. <Tfrt k y Eny >Hb k gyr j*y AnsY kl Aly AHb hm ATfr hm :
   إذا طفقت ل ي عني أحب ل غير جذل انتي كل الي احب هم اطرف هم :

   (i32
    + [<*A:#0,
      (antecedent
       , [Tfrt:#2])]),
    (consequent
     , [Eny:#4,
       (tns , y:#3),
       (subj , zero#8),
       (xcomp
        , [Hb:#6])])])})

   (i98
    + [AnsY:#2,
      (subj , zero#2),
      (xcomp
       , [ATfr:#7,
         (subj
          , [kl:#3,
            (whmod
             , [AHb:#5,
               (subj , Aly:#4),
               (dobj , hm:#6)])]),
             (dobj , hm:#8)])])})
APPENDIX H. EXAMPLES OF PARASITE* ANNOTATION

7. @ BernardShawQ8 mA fy hA $y . . . ymkn mw nAsy fDI hm Ely h . . . Hlyb w syrylAk ! !
   ما في ها شيء ... يمكن معناه فضل هم علي ه ... حليب و سيريلاك ! !
   (i8 + [hA:#5])

   (i21
   + [ymkn:#0,
      ((subj , zero#0)),
      (xcomp
       , [nAsy:#2,
          {(nmod, [fDI:#4])},
          {(pred, [Ely:#6])},
          {(advmod, mw:#1)})])))

   (i7
   + [!!#3,
      ((frag
       , [w:#1,
          {(conj1, Hlyb:#0)},
          {(conj2, syrylAk:#2)})])])

8. Aw mtnY
   او متي
   (i1 + mtnY:#1)

9. mn dwAEy srwr y HDwr y l Hflt km Al mqAm h b Al >ms ElY Srf AlzbAly w Alty >qym t
   brqy w mkAnp $xS km Al krym
   من دوائي سروري حضور ي ل حفلت كم ال مقام ه ب ال أمس على شرف الزبالي و التي
   أقيم ت برقي و مكانة شخيص كم ال كريم
   (i2580
   + [HDwr:#4,
      {(subj, zero#1)},
      {(dobj, y:#5)},
      {(ppmod
       , [l:#6,
          {(comp
           , [Hflt:#7,
              {(nmod, km:#8)}},
              (nmod
              , [km:#9})],
              (nmod
              , [al:#10},
              (nmod
              , [al:#11})],
              (nmod
              , [al:#12})]})

   (i241
   + [aEv:#9,
      {(conj1, HDwr:#4)},
      {(conj2, mtnY:#1)})

   (i18
   + [w:#1,
      {(conj1, Hlyb:#0)},
      {(conj2, syrylAk:#2)})]})
APPENDIX H. EXAMPLES OF PARASITE\textsuperscript{*} ANNOTATION

10. frE mTEm Albyk Al jdyd fy brydp http://t.co/m1BG2yVNMc

فرع مطعم الـبيك الـجديد في بريدة http://t.co/m1BG2yVNMc
APPENDIX H. EXAMPLES OF PARASITE* ANNOTATION

+ [frE:#0,
  ((nmod
    , [mTEm:#1,
      {(nmod
        , [Albyk:#2,
          {(nmod , [jdyd:#3])},
          {(ppmod , [fy:#6])})}]
      ,(pred , http://t.co/m1BG2yVNMc#7)]))}]