Simultaneous Tokenization and Part-of-Speech Tagging for Arabic without a Morphological Analyzer

Seth Kulick
Linguistic Data Consortium
University of Pennsylvania
skulick@seas.upenn.edu

Abstract

We describe an approach to simultaneous tokenization and part-of-speech tagging that is based on separating the closed and open-class items, and focusing on the likelihood of the possible stems of the open-class words. By encoding some basic linguistic information, the machine learning task is simplified, while achieving state-of-the-art tokenization results and competitive POS results, although with a reduced tag set and some evaluation difficulties.

1 Introduction

Research on the problem of morphological disambiguation of Arabic has noted that techniques developed for lexical disambiguation in English do not easily transfer over, since the affixation present in Arabic creates a very different tag set than for English, in terms of the number and complexity of tags. In additional to inflectional morphology, the POS tags encode more complex tokenization sequences, such as *preposition* + *noun* or *noun* + *possessive pronoun*.

One approach taken to this problem is to use a morphological analyzer such as BAMA-v2.0 (Buckwalter, 2004) or SAMA-v3.1 (Maamouri et al., 2009c)[1], which generates a list of all possible morphological analyses for a given token. Machine learning approaches can model separate aspects of a solution (e.g., "has a pronominal clitic") and then combine them to select the most appropriate solution from among this list. A benefit of this approach is that by picking a single solution from the morphological analyzer, the part-of-speech and tokenization comes as a unit (Habash and Rambow, 2005; Roth et al., 2008).

In contrast, other approaches have used a pipelined approach, with separate models to first do tokenization and then part-of-speech tagging (Diab et al., 2007; Diab, 2009). While these approaches have somewhat lower performance than the joint approach, they have the advantage that they do not rely on the presence of a full-blown morphological analyzer, which may not always be available or appropriate as the data shifts to different genres or Arabic dialects.

In this work we present a novel approach to this problem that allows us to do simultaneous tokenization and core part-of-speech tagging with a simple classifier, without using a full-blown morphological analyzer. We distinguish between closed-class and open-class categories of words, and encode regular expressions that express the morphological patterns for the former, and simple regular expressions for the latter that provide only the generic templates for affixation. We find that a simple baseline for the closed-class words already works very well, and for the open-class words we classify only the possible stems for all such expressions. This is however sufficient for tokenization and core POS tagging, since the stem identifies the appropriate regular expression, which then in turn makes explicit, simultaneously, the tokenization and part-of-speech information.

2 Background

The Arabic Treebank (ATB) contains a full morphological analysis of each "source token", a whitespace/punctuation-delimited string from the source text. The SAMA analysis includes four fields, as shown in the first part of Table 1.[2] `TEXT` is the actual source token text, to be analyzed. `VOC` is the vocalized form, including diacritics. Each `VOC` segment has associated with it a `POS` tag and

[1]SAMA-v3.1 is an updated version of BAMA, with many significant differences in analysis.

[2]This is the analysis for one particular instance of *ktbh*. The same source token may receive another analysis elsewhere in the treebank.

ATB analysis for one source token:

TEXT:	ktbh		
VOC:	kutub	u	hu
POS:	NOUN	CASE_NOM	POSS_PRON_3MS
GLOSS:	books	[def.nom.]	its/his

Results in two ATB tree tokens:

TEXT:	ktb	h
VOC:	kutub+u	hu
POS:	NOUN+CASE_NOM	POSS_PRON_3MS

Current work recovers:

TEXT:	ktb	h
POS:	NOA	POSS_PRON

Table 1: Example analysis of one source token

NOUN, ADJ, NOUN.VN, ADJ.VN, NOUN_NUM, ADJ_NUM, NOUN_QUANT, ADJ_COMP, ABBREV	NOA (Noun or Adjective)
IV, IV_PASS	IV
PV, PV_PASS	PV
IVSUFF_DO, PVSUFF_DO	OBJ_PRON

Table 2: Collapsing of ATB core tags into reduced core tags

NOA	173938	PART	288
PREP	49894	RESTRIC_PART	237
PUNC	41398	DET	215
NOUN_PROP	29423	RC_PART	192
CONJ	28257	FOCUS_PART	191
PV	16669	TYPO	188
IV	15361	INTERROG_PART	187
POSS_PRON	9830	INTERROG_ADV	169
SUB_CONJ	8200	INTERROG_PRON	112
PRON	6995	CV	106
REL_PRON	5647	VOC_PART	74
DEM	3673	VERB	62
OBJ_PRON	2812	JUS_PART	56
NEG_PART	2649	FOREIGN	46
PSEUDO_VERB	1505	DIALECT	41
FUT_PART	1099	INTERJ	37
ADV	1058	EMPHATIC_PART	19
VERB_PART	824	CVSUFF_DO	15
REL_ADV	414	GRAMMAR_PROB	4
CONNEC_PART	405	LATIN	1

Table 3: The 40 "reduced core tags", and their frequencies in ATB3-v3.2. The total count is 402291, which is the number of tree tokens in ATB3-v3.2.

GLOSS. While "tokenization" can be done in different ways on top of this analysis, the ATB splits the VOC/POS/GLOSS segments up based on the POS tags to form the "tree tokens" necessary for treebanking. As shown in the second part of Table 1, the first two segments remain together as one tree token, and the pronoun is separated as a separate tree token. In addition, the input TEXT is separated among the two tree tokens.[3]

Each tree token's POS tag therefore consists of what can be considered an "ATB core tag", together with inflectional material (case, gender, number). For example, in Table 1, the "core tag" of the first tree token is NOUN. In this work, we aim to recover the separation of a source token TEXT into the corresponding separate tree token TEXTs, together with a "reduced core tag" for each tree token. By "reduced core tag", we mean an ATB core tag that has been reduced in two ways:

(1) All inflectional material [infl] is stripped off six ATB core tags: PRON[infl], POSS_PRON[infl], DEM[infl], [IV|PV|CV]SUFF_DO[infl]

(2) Collapsing of some ATB core tags, as listed in Table 2.

These two steps result in a total of 40 reduced core tags, and each tree token has exactly one such reduced core tag. We work with the ATB3-v3.2 release of the ATB (Maamouri et al., 2009b), which has 339710 source tokens and 402291 tree tokens, where the latter are derived from the former as discussed above. Table 3 lists the 40 reduced tags we use, and their frequency among the ATB3-v3.2 tree tokens.

3 Description of Approach

Given a source token, we wish to recover (1) the tree tokens (which amounts to recovering the ATB tokenization), and (2) the reduced core POS tag for each tree token. For example, in Table 1, given the input source token TEXT *ktbh*, we wish to recover the tree tokens *ktb*/NOA and *h*/POSS_PRON.

As mentioned in the introduction, we use regular expressions that encode all the tokenization and POS tag possibilities. Each "group" (substring unit) in a regular expression (regex) is assigned an internal name, and a list is maintained of the possible reduced core POS tags that can occur with that regex group. It is possible, and indeed usually the case for groups representing affixes, that more than one such POS tag is possible. However, it is crucial for our approach that while some given source token TEXT may match many regular expressions (regexes), when the POS tag is also taken into account, there can be only one match among all the (open or closed-class) regexes. We say a source token "pos-matches" a regex if the TEXT matches and POS tags match, and "text-matches" if the TEXT matches the regex regardless of the POS. During training, the pos-matching

[3]See (Kulick et al., 2010) for a detailed discussion of how this splitting is done and how the tree token TEXT field (called INPUT STRING in the ATB releases) is created.

(REGEX #1) [w|f]lm
  w: [PART, CONJ, SUB_CONJ, PREP]
  f: [CONJ, SUB_CONJ, CONNEC_PART, RC_PART]
  lm: [NEG_PART]
(REGEX #2) [w|f]lm
  w: and f: same as above
  lm: [REL_ADV,INTERROG_ADV]

Figure 1: Two sample closed-class regexes

regex for a source token TEXT is stored as the gold solution for closed-class patterns, or used to create the gold label for the open-class classifier.

We consider the open-class tags in Table 3 to be: NOUN_PROP, NOA, IV, CV, PV, VERB. A source token is considered to have an open-class solution if any of the tree tokens in that solution have an open-class tag. For example, *ktbh* in Table 1 has an open-class solution because one of the tree tokens has an open-class tag (NOA), even though the other is closed-class (POSS_PRON).

We encode the possible solutions for closed-class source tokens using the lists in the ATB morphological guidelines (Maamouri et al., 2009a). For example, Figure 1 shows two of the closed-class regexes. The text *wlm* can text-match either REGEX #1 or #2, but when the POS tag for *lm* is taken into account, only one can pos-match. We return to the closed-class regexes in Section 4.

We also encode regular expression for the open-class source tokens, but these are simply generic templates expressing the usual affix possibilities, such as:
`[wf] [blk] stem_NOA poss_pronoun`
where there is no list of possible strings for `stem_NOA`, but which instead can match anything. While all parts except for the stem are optional, we do not make such parts optional in a single expression. Instead, we multiple out the possibilities into different expressions with different parts (e.g., [wf]) being obligatory). The reason for this is that we give different names to the stem in each case, and this is the basis of the features for the classifier. As with the closed-class regexes, we associate a list of possible POS tags for each named group within a regular expression. Here the `stem_NOA` group can only have the tag NOA.

We create features for a classifier for the open-class words as follows. Each word is run through all of the open-class regular expressions. For each expression that text-matches, we make a feature which is the name of the stem part of the regular expression, along with the characters that match the stem. The stem name encodes whether there is a prefix or suffix, but does not include a POS tag. However, the source token pos-matches exactly one of the regular expressions, and the pos tag for the stem is appended to the named stem for that expression to form the gold label for training and the target for testing.

For example, Table 4 lists the matching regular expression for three words. The first, *yjry*, text-matches the generic regular expressions for *any string*/NOA, *any string*/IV, etc. These are summarized in one listing, *yjry*/all. The name of the stem for all these expressions is the same, just *stem*, and so they all give rise to the same feature, `stem=yjry`. It also matches the expression for a NOA with a possessive pronoun[4], and in this case the stem name in the regular expression is `stem_spp` (which stands for "stem with a possessive pronoun suffix"), and this gives rise to the feature `stem_spp=yjr`. Similarly, for *wAfAdt* the stem of the second expression has the name `p_stem`, for a prefix. The third example shows the different stem names that occur when there are both prefix and suffix possibilities. For each example, there is exactly one regex that not only text-matches, but also pos-matches. The combination of the stem name in these cases together with the gold tag forms the gold label, as indicated in column 3.

Therefore, for each source token TEXT, the features include the ones arising from the named stems of all the regexes that text-match that TEXT, as shown in column 4, and the gold label is the appropriate stem name together with the POS tag, as shown in column 3. We also include some typical features for each stem, such as first and last two letters of each stem, etc. For example, `wAfAdt` would also have the features `stem_fl=w`, `p_stem_fl=A`, indicating that the first letter of `stem` is `w` and the first letter of `p_stem` is `A`. We also extract a list of proper nouns from SAMA-v3.1 as a temporary proxy for a named entity list, and include a feature for a stem if that stem is in the list (`stem_in_list`, `p_stem_in_list`, etc.)

We do not model separate classifiers for prefix possibilities. There is a dependency between the

[4]The regex listed is slightly simplified. It actually contains a reference to the list of all possessive pronouns, not just *y*.

source TEXT	text-matching regular expressions	gold label	feature
yjry	yjry/all (*happens*)	stem:IV	stem=yjry
	yjr/NOA+y/POSS_PRON		stem_spp=yjr
wAfAdt	wAfAdt/all		stem=wAfAdt
	w + AfAdt/all (*and+reported*)	p_stem:PV	p_stem=AfAdt
lAstyDAHhm	lAstyDAHhm/all		stem=lAstyDAHhm
	l/PREP + AstyDAHhm/NOA		p_stem=AstyDAHhm
	l/PREP + AstyDAH/NOA + hm/POSS_PRON *for + request for clarification + their*	p_stem_spp:NOA	p_stem_spp=AstyDAH
	lAstyDAH/NOA + hm/POSS_PRON		stem_spp=lAstyDAH
	lAstyDAH/IV,PV,CV + hm/OBJ_PRON		stem_svop=lAstyDAH
	l/PREP,JUS_PART + AstyDAH/IV,PV,CV + hm/OBJ_PRON		p_stem_svop=AstyDAH

Table 4: Example features and gold labels for three words. Each text-matching regex gives rise to one feature shown in column 4, based on the stem of that regular expression. A p_ before a stem means that it has a prefix, _spp after means that it has a possessive pronouns suffix, and _svop means that it has a (verbal) object pronoun suffix. "all" in the matching regular expression is shorthand for text-matching all the corresponding regular expressions with NOA, IV, etc. For each word, exactly one regex also pos-matches, which results in the gold label, shown in column 3.

possibility of a prefix and the likelihood of the remaining stem, and so we focus on the likelihood of the possible stems, where the open-class regexes enumerate the possible stems. A gold label together with the source token TEXT maps back to a single regex, and so for a given label, the TEXT is parsed by that regular expression, resulting in a tokenization along with list of possible POS tags for each affix group in the regex.[5]

During training and testing, we run each word through all the open and closed regexes. Text-matches for an open-class regex give rise to features as just described. Also, if the word matches any closed-class regex, it receives the feature MATCHES_CLOSED. During training, if the correct match for the word is one of the closed-class expressions, then the gold label is CLOSED. The classifier is used only to get solutions for the open-class words, although we wish to give the classifier all the words for the sentence. The cross-product of the stem name and (open-class) reduced core POS tags, plus the CLOSED tag, yields 24 labels for a CRF classifier in Mallet (McCallum, 2002).

4 Experiments and Evaluation

We worked with ATB3-v3.2, following the training/devtest split in (Roth et al., 2008) on a previous release of the same data. We keep a listing (List #1) of all (source token TEXT, solution) pairs seen during training. For an open-class solution, "solution" is the gold label as described in Section 3. For a closed-class solution, "solution" is the name of the single pos-matching regex. In addition, for every regex seen during training that pos-matches some source token TEXT, we keep a listing (List #2) of all ((regex-group-name, text), POS-tag) tuples. We use the information in List #1 to choose a solution for all words seen in training in the Baseline and Run 2 below, and in Run 3, for words text-matching a closed-class expression. We use List #2 to disambiguate all remaining cases of POS ambiguity, wherever a solution comes from.

For example, if *wlm* is seen during testing, List #1 will be consulted to find the most common solution (REGEX #1 or #2), and in either case, List #2 will be consulted to determine the most frequent tag for *w* as a prefix. While there is certainly room for improvement here, this works quite well since the tags for the affixes do not vary much.

We score the solution for a source token instance as correct for tokenization if it exactly matches the TEXT split for the tree tokens derived from that source token instance in the ATB. It is correct for POS if correct for tokenization and if each tree token has the same POS tag as the reduced core tag for that tree token in the ATB.

For a simple baseline, if a source token TEXT is in List #1 then we simply use the most frequent stored solution. Otherwise we run the TEXT through all the regexes. If it text-matches any closed-class expression, we pick a random choice from among those regexes and otherwise from the open-class regexes that it text-matches. Any POS ambiguities for a regex group are disambiguated

[5]In Section 4 we discuss how these are narrowed down to one POS tag.

Solution	Baseline			Run 2			Run 3		
Origin	# tokens	Tok	POS	# tokens	Tok	POS	# tokens	Tok	POS
All	51664	96.0%	87.4%	51664	**99.4%**	**95.1%**	51664	99.3%	95.1%
Stored	46072	99.8%	96.6%	46072	99.8%	96.6%	16145	99.6%	96.4%
Open	5565	64.6%	11.6%	10	10.0%	0.0%	11	54.5%	0.0%
Closed	27	81.5%	59.3%	27	81.5%	63.0%	27	81.5%	63.0%
Mallet	0			5555	96.0%	83.8%	35481	99.1%	94.5%

Table 5: Results for Baseline and two runs. Origin “stored” means that the appropriate regex came from the list stored during training. Origins “open” and “closed” are random choices from the open or closed regexes for the source token. “Mallet” means that it comes from the label output by the CRF classifier.

using List #2, as discussed above. The results are shown in Table 5. The score is very high for the words seen during training, but much lower for open-class words that were not. As expected, almost all (except 27) instances of closed-class words were seen during training.

For run 2, we continue to use the stored solution if the token was seen in training. If not, then if the TEXT matches one or more closed-class regexes, we randomly choose one. Otherwise, if the CRF classifier has produced an open-class match for that token, we use that (and otherwise, in only 10 cases, use a random open-class match). There is a significant improvement in the score for the open-class items, and therefore in the overall results.

For run 3, we put more of a burden on the classifier. If a word matches any closed-class expression, we either use the most frequent occurence during training (if it was seen), or use a random maching closed-class expression (if not). If the word doesn’t match a closed-class expression, we use the mallet result. The mallet score goes up, almost certainly because the score is now including results on words that were seen during training. The overall POS result for run 3 is slightly less than run 2. (95.099% compared to 95.147%).

It is not a simple matter to compare results with previous work, due to differing evaluation techniques, data sets, and POS tag sets. With different data sets and training sizes, Habash and Rambow (2005) report 99.3% word accuracy on tokenization, and Diab et al. (2007) reports a score of 99.1%. Habash and Rambow (2005) reported 97.6% on the LDC-supplied reduced tag set, and Diab et al. (2007) reported 96.6%. The LDC-supplied tag set used is smaller than the one in this paper (24 tags), but does distinguish between NOUN and ADJ. However, both (Habash and Rambow, 2005; Diab et al., 2007) assume gold tokenization for evaluation of POS results, which we do not. The “MorphPOS” task in (Roth et al., 2008), 96.4%, is somewhat similar to ours in that it scores on a “core tag”, but unlike for us there is only one such tag for a source token (easier) but it distinguishes between NOUN and ADJ (harder).

We would like to do a direct comparison by simply runing the above systems on the exact same data and evaluating them the same way. However, this unfortunately has to wait until new versions are released that work with the current version of the SAMA morphological analyzer and ATB.

5 Future Work

Obvious future work starts with the need to include determiner information in the POS tags and the important NOUN/ADJ distinction. There are various possibilities for recovering this information, such as (1) using a different module combining NOUN/ADJ disambiguation together with NP chunking, or (2) simply including NOUN/ADJ in the current classifier instead of NOA. We will be implementing and comparing these alternatives. We also will be using this system as a preprocessing step for a parser, as part of a complete Arabic NLP pipeline.

Acknowledgements

We thank Ann Bies, David Graff, Nizar Habash, Mohamed Maamouri, and Mitch Marcus for helpful discussions and comments. This work was supported by the Defense Advanced Research Projects Agency, GALE Program Grant No. HR0011-06-1-0003 and by the GALE program, DARPA/CMO Contract No. HR0011-06-C-0022. The content of this paper does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred.

References

Tim Buckwalter. 2004. Buckwalter Arabic morphological analyzer version 2.0. Linguistic Data Consortium LDC2004L02.

Mona Diab, Kadri Hacioglu, and Daniel Jurafsky. 2007. Automatic processing of Modern Standard Arabic text. In Abdelhadi Soudi, Antal van den Bosch, and Gunter Neumann, editors, *Arabic Computational Morphology*, pages 159–179. Springer.

Mona Diab. 2009. Second generation tools (AMIRA 2.0): Fast and robust tokenization, pos tagging, and base phrase chunking. In *Proceedings of 2nd International Conference on Arabic Language Resources and Tools (MEDAR)*, Cairo, Egypt, April.

Nizar Habash and Owen Rambow. 2005. Arabic tokenization, part-of-speech tagging and morphological disambiguation in one fell swoop. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 573–580, Ann Arbor, Michigan, June. Association for Computational Linguistics.

Seth Kulick, Ann Bies, and Mohamed Maamouri. 2010. Consistent and flexible integration of morphological annotation in the Arabic Treebank. In *Language Resources and Evaluation (LREC)*.

Mohamed Maamouri, Ann Bies, Sondos Krouna, Fatma Gaddeche, Basma Bouziri, Seth Kulick, Wigdane Mekki, and Tim Buckwalter. 2009a. Arabic Treebank Morphological and Syntactic guidelines, July. http://projects.ldc.upenn.edu/ArabicTreebank.

Mohamed Maamouri, Ann Bies, Seth Kulick, Sondos Krouna, Fatma Gaddeche, and Wajdi Zaghouani. 2009b. Arabic treebank part 3 - v3.2. Linguistic Data Consortium LDC2010T08, April.

Mohammed Maamouri, Basma Bouziri, Sondos Krouna, David Graff, Seth Kulick, and Tim Buckwalter. 2009c. Standard Arabic morphological analyzer (SAMA) version 3.1. Linguistic Data Consortium LDC2009E73.

Andrew McCallum. 2002. Mallet: A machine learning for language toolkit. http://mallet.cs.umass.edu.

Ryan Roth, Owen Rambow, Nizar Habash, Mona Diab, and Cynthia Rudin. 2008. Arabic morphological tagging, diacritization, and lemmatization using lexeme models and feature ranking. In *Proceedings of ACL-08: HLT, Short Papers*, pages 117–120, Columbus, Ohio, June. Association for Computational Linguistics.