Memory-based morphological analysis generation and part-of-speech tagging of Arabic

Erwin Marsi, Antal van den Bosch
ILK, Tilburg University
P.O. Box 90153
NL-5000 LE Tilburg
The Netherlands
{E.C.Marsi,Antal.vdnBosch}@uvt.nl

Abdelhadi Soudi
Center for Computational Linguistics
Ecole Nationale de L'Industrie Minérale
Rabat,
Morocco,
asoudi@gmail.com/asoudi@enim.ac.ma

Abstract

We explore the application of memory-based learning to morphological analysis and part-of-speech tagging of written Arabic, based on data from the Arabic Treebank. Morphological analysis – the construction of all possible analyses of isolated unvoweled wordforms – is performed as a letter-by-letter operation prediction task, where the operation encodes segmentation, part-of-speech, character changes, and vocalization. Part-of-speech tagging is carried out by a bi-modular tagger that has a subtagger for known words and one for unknown words. We report on the performance of the morphological analyzer and part-of-speech tagger. We observe that the tagger, which has an accuracy of 91.9% on new data, can be used to select the appropriate morphological analysis of words in context at a precision of 64.0 and a recall of 89.7.

1 Introduction

Memory-based learning has been successfully applied to morphological analysis and part-of-speech tagging in Western and Eastern-European languages (van den Bosch and Daelemans, 1999; Daelemans et al., 1996). With the release of the Arabic Treebank by the Linguistic Data Consortium (current version: 3), a large corpus has become available for Arabic that can act as training material for machine-learning algorithms. The data facilitates machine-learned part-of-speech taggers, tokenizers, and shallow parsing units such as chunkers, as exemplified by Diab et al. (2004).

However, Arabic appears to be a special challenge for data-driven approaches. It is a Semitic language with a non-concatenative morphology. In addition to prefixation and suffixation, inflectional and derivational processes may cause stems to undergo infixational modification in the presence of different syntactic features as well as certain consonants. An Arabic word may be composed of a stem consisting of a consonantal root and a pattern, affixes, and clitics. The affixes include inflectional markers for tense, gender, and number. The clitics may be either attached to the beginning of stems (proclitics) or to the end of stems (enclitics) and include possessive pronouns, pronouns, some prepositions, conjunctions and determiners.

Arabic verbs, for example, can be conjugated according to one of the traditionally recognized patterns. There are 15 triliteral forms, of which at least 9 are in common. They represent very subtle differences. Within each conjugation pattern, an entire paradigm is found: two tenses (perfect and imperfect), two voices (active and passive) and five moods (indicative, subjunctive, jussive, imperative and energetic). Arabic nouns show a comparably rich and complex morphological structure. The broken plural system, for example, is highly allomorphic: for a given singular pattern, two different plural forms may be equally frequent, and there may be no way to predict which of the two a particular singular will take. For some singulars as many as three further statistically minor plural patterns are also possible.

Various ways of accounting for Arabic morphology have been proposed. The type of account of Arabic morphology that is generally accepted by (computational) linguists is that proposed by (McCarthy, 1981). In his proposal, stems are formed by a derivational combination of a root morpheme and a vowel melody. The two are arranged according to canonical patterns. Roots are said to interdigitate with patterns to form stems. For example, the Arabic stem *katab* (”he wrote”) is composed of the morpheme *ktb* (”the notion of writing”) and the vowel melody morpheme ’a-a’. The two are integrated according to the pattern CVCVC (C=consonant, V=vowel). This means that word structure in this morphology is not built linearly as is the case in concatenative morphological systems.

The attempts to model Arabic morphology in a two-level system (Kay’s (1987) Finite State Model, Beesley’s (1990; 1998) Two-Level Model and Kiraz’s (1994) Multi-tape Two-Level Model) reflect McCarthy’s separation of levels. It is beyond the scope of this paper to provide a detailed description of these models, but see (Soudi, 2002).

In this paper, we explore the use of memory-based learning for morphological analysis and part-of-speech (PoS) tagging of written Arabic. The next section summarizes the principles of memory-based learning. The following three sections describe our exploratory work on memory-based morphological analysis and PoS tagging, and integration of the two tasks. The final two sections contain a short discussion of related work and an overall conclusion.

2 Memory-based learning

Memory-based learning, also known as instance-based, example-based, or lazy learning (Aha et al., 1991; Daelemans et al., 1999), extensions of the k-nearest neighbor classifier (Cover and Hart, 1967), is a supervised inductive learning algorithm for learning classification tasks. Memory-based learning treats a set of labeled (pre-classified) training instances as points in a multi-dimensional feature space, and stores them as such in an *instance base* in memory. Thus, in contrast to most other machine learning algorithms, it performs no abstraction, which allows it to deal with productive but low-frequency exceptions (Daelemans et al., 1999).

An instance consists of a fixed-length vector of n feature-value pairs, and the classification of that particular feature-value vector. After the instance base is stored, new (test) instances are classified by matching them to all instances in the instance base, and by calculating with each match the *distance*, given by a distance kernel function. Classification in memory-based learning is performed by the k-NN algorithm that searches for the k ‘nearest neighbours’ according to the $\Delta(X, Y)$ kernel function[1].

The distance function and the classifier can be refined by several kernel plug-ins, such as feature weighting (assigning larger distance to mismatches on important features), and distance weighting (assigning a smaller vote in the classification to more distant nearest neighbors). Details can be found in (Daelemans et al., 2004).

3 Morphological analysis

We focus first on morphological analysis . Training on data extracted from the Arabic Treebank, we induce a morphological analysis generator which we control for undergeneralization (recall errors) and overgeneralization (precision errors).

3.1 Data

3.1.1 Arabic Treebank

Our point of departure is the Arabic Treebank 1 (ATB1), version 3.0, distributed by LDC in 2005, more specifically the “after treebank” PoS-tagged data. Unvoweled tokens as they appear in the original news paper are accompanied in the treebank by vocalized versions; all of their morphological analyses are generated by means of Tim Buckwalter’s Arabic Morphological Analyzer (Buckwalter, 2002), and the appropriate morphological analysis is singled out. An example is given in Figure 1. The input token (`INPUT STRING`) is transliterated (`LOOK-UP WORD`) according to Buckwalter’s transliteration system. All possible vocalizations and their morphological analyzes are listed (`SOLUTION`). The analysis is rule-based, and basically consists of three steps. First, all possible segmentations of the input string

[1]All experiments with memory-based learning were performed with TiMBL, version 5.1 (Daelemans et al., 2004), available from `http://ilk.uvt.nl`.

```
INPUT STRING: \331\203\330\252\330\250
LOOK-UP WORD: ktb
     Comment:
       INDEX: P2W38
  SOLUTION 1: (kataba) [katab-u_1] katab/PV+a/PVSUFF_SUBJ:3MS
     (GLOSS): write + he/it [verb]
* SOLUTION 2: (kutiba) [katab-u_1] kutib/PV_PASS+a/PVSUFF_SUBJ:3MS
     (GLOSS): be written/be fated/be destined + he/it [verb]
  SOLUTION 3: (kutub) [kitAb_1] kutub/NOUN
     (GLOSS): books
  SOLUTION 4: (kutubu) [kitAb_1] kutub/NOUN+u/CASE_DEF_NOM
     (GLOSS): books + [def.nom.]
  SOLUTION 5: (kutuba) [kitAb_1] kutub/NOUN+a/CASE_DEF_ACC
     (GLOSS): books + [def.acc.]
  SOLUTION 6: (kutubi) [kitAb_1] kutub/NOUN+i/CASE_DEF_GEN
     (GLOSS): books + [def.gen.]
  SOLUTION 7: (kutubN) [kitAb_1] kutub/NOUN+N/CASE_INDEF_NOM
     (GLOSS): books + [indef.nom.]
  SOLUTION 8: (kutubK) [kitAb_1] kutub/NOUN+K/CASE_INDEF_GEN
     (GLOSS): books + [indef.gen.]
  SOLUTION 9: (ktb) [DEFAULT] ktb/NOUN_PROP
     (GLOSS): NOT_IN_LEXICON
  SOLUTION 10: (katb) [DEFAULT] ka/PREP+tb/NOUN_PROP
     (GLOSS): like/such as + NOT_IN_LEXICON
```

Figure 1: Example token from ATB1

in terms of prefixes (0 to 4 characters long), stems (at least one character), and suffixes (0 to 6 characters long) are generated. Next, dictionary lookup is used to determine if these segments are existing morphological units. Finally, the numbers of analyses is further reduced by checking for the mutual compatibility of prefix+stem, stem+suffix, and prefix+stem in three compatibility tables. The resulting analyses have to a certain extent been manually checked. Most importantly, a star (*) preceding a solution indicates that this is the correct analysis in the given context.

3.1.2 Preprocessing

We grouped the 734 files from the treebank into eleven parts of approximately equal size. Ten parts were used for training and testing our morphological analyzer, while the final part was used as held-out material for testing the morphological analyzer in combination with the PoS tagger (described in Section 4).

In the corpus the number of analyses per word is not entirely constant, either due to the automatic generation method or to annotator edits. As our initial goal is to predict all possible analyses for a given word, regardless of contextual constraints, we first created a *lexicon* that maps every word to all analyses encountered and their respective frequencies From the 185,061 tokens in the corpus, we extracted 16,626 unique word types – skipping punctuation tokens – and 129,655 analyses, which amounts to 7.8 analyses per type on average.

```
= = = = = k t b = = = ka/PREP+;ka;k;ku
= = = = k t b = = = = a/PREP+t;uti;ata;t;utu
= = = k t b = = = = = ab/PV+a/PVSUFF_SUBJ:3MS+;
                      b/NOUN_PROP+;ub/NOUN+i/CASE_DEF_GEN+;
                      ub/NOUN+a/CASE_DEF_ACC+;
                      ub/NOUN+K/CASE_INDEF_GEN+;
                      ib/PV_PASS+a/PVSUFF_SUBJ:3MS+;
                      ub/NOUN+N/CASE_INDEF_NOM+;ub/NOUN+;
                      ub/NOUN+u/CASE_DEF_NOM+
```

Figure 2: Instances for the analyses of the word *ktb* in Figure 1.

3.1.3 Creating instances

These separate lexicons were created for training and testing material. The lexical entries in a lexicon were converted to *instances* suitable to memory-based learning of the mapping from words to their analyses (van den Bosch and Daelemans, 1999). Instances consist of a sequence of feature values and a corresponding class, representing a potentially complex morphological operation.

The features are created by sliding a window over the unvoweled look-up word, resulting in one instance for each character. Using a 5-1-5 window yields 11 features, i.e. the input character in focus, plus the five preceding and five following characters. The equal sign (=) is used as a filler symbol.

The instance classes represent the morphological analyses. The classes corresponding to a word's characters should enable us to derive all associated analyses. This implies that the classes need to encode several aspects simultaneously: vocalization, morphological segmentation and tagging. The following template describes the format of classes:

```
class = subanalysis; subanalysis; ...

subanalysis = preceding vowels & tags +
              input character +
              following vowels & tags
```

For example, the classes of the instances in Figure 2 encode the ten solutions for the word *ktb* in Figure 1. The ratio behind this encoding is that it allows for a simple derivation of the solution, akin to the way that the pieces of a jigsaw puzzle can be combined. We can exhaustively try all combinations of the subanalyses of the classes, and check if the right side of one subanalysis matches the left side of a subsequent subanalysis. This reconstruction process is illustrated in Figure 3 (only two reconstructions are depicted, corresponding to SOLUTION 1 and SOLUTION 4). For example, the subanalysis `ka` from the first class in Figure 2 matches the subanalysis `ata` from the sec-

```
ka                                  ku
 ata                                 utu
   ab/PV+a/PVSUFF_SUBJ:3MS             ub/NOUN+u/CASE_DEF_NOM
-------------------------- +        ------------------------ +
katab/PV+a/PVSUFF_SUBJ:3MS          kutub/NOUN+u/CASE_DEF_NOM
```

Figure 3: Illustration of how two morphological analyses are reconstructed from the classes in Figure 2.

ond class, which in turn matches the subanalysis `ab/PV+a/PVSUFF_SUBJ:3MS` from the third class; together these constitute the complete analysis `katab/PV+a/PVSUFF_SUBJ:3MS`.

3.2 Initial Experiments

To test the feasibility of our approach, we first train and test on the full data set. Timbl is used with its default settings (overlap distance function, gain-ratio feature weighting, $k = 1$). Rather than evaluating on the accuracy of predicting the complex classes, we evaluate on the complete correctness of all reconstructed analyses, in terms of precision, recall, and F-score (van Rijsbergen, 1979). As expected, this results in a near perfect recall (97.5). The precision, however, is much lower (52.5), indicating a substantial amount of analysis overgeneration; almost one in two generated analyses is actually not valid. With an F-score of only 68.1, we are clearly not able to reproduce the training data perfectly.

Next we split the data in 9 parts for training and 1 part for testing. The k-NN classifier is again used with its default settings. Table 1 shows the results broken down into known and unknown words. As known words can be looked up in the lexicon derived from the training material, the first row presents the results with lookup and the second row without lookup (that is, with prediction). The fact that even with lookup the performance is not perfect shows that the upper bound for this task is not 100%. The reason is that apparently some words in the test material have received analyses that never occur in the training material and vice versa. For known words without lookup, the recall is still good, but the precision is low. This is consistent with the initial results mentioned above. For unknown words, both recall and precison are much worse, indicating rather poor generalization.

To sum up, there appear to be problems with both the precision and the recall. The precision is low for known words and even worse for unknown words.

	#Wrds	Prec	Rec	F
Known with lookup	3220	92.6	98.1	95.3
Known without lookup	3220	49.9	95.0	65.5
Unknown	847	22.8	26.8	24.7

Table 1: Results of initial experiments split into known and unknown words, and with and without lookup of known words.

	#Wrds	Prec	Rec	F
Known	3220	15.6	99.0	26.9
Unknown	847	3.9	66.8	7.5

Table 2: Results of experiments for improving the recall, split into known and unknown words.

Analysis overgeneration seems to be a side effect of the way we encode and reconstruct the analyses. The recall is low for unknown words only. There appear to be at least two reasons for this undergeneration problem. First, if just one of the predicted classes is incorrect (one of the pieces of the jigsaw puzzle is of the wrong shape) then many, or even all of the reconstructions fail. Second, some generalizations cannot be made, because infrequent classes are overshadowed by more frequent ones with the same features. Consider, for example, the instance for the third character (l) of the word *jEl*:

```
= = = j E l = = = = =
```

Its real class in the test data is:

```
al/VERB_PERFECT+;ol/NOUN+
```

When the k-NN classifier is looking for its nearest neighbors, it finds three; two with a "verb imperfect" tag, and one with a "noun" tag.

```
{ al/VERB_IMPERFECT+ 2, ol/NOUN+ 1}
```

Therefore, the class predicted by the classifier is `al/VERB_IMPERFECT+`, because this is the majority class in the NN-set. So, although a part of the correct solution is present in the NN-set, simple majority voting prevents it from surfacing in the output.

3.3 Improving recall

In an attempt to address the low recall, we revised our experimental setup to take advantage of the complete NN-set. As before, the k-NN classifier is used,

	Prec	Rec	F
Known	58.6 (0.4)	66.6 (0.5)	62.4 (0.3)
Unknown	28.7 (3.7)	37.2 (1.2)	32.2 (2.5)
All	53.4 (1.2)	62.2 (0.6)	57.5 (0.8)

Table 3: Average results and SD of the 10-fold CV experiment, split into known and unknown words

but rather than relying on the classifier to do the majority voting over the (possibly weighted) classes in the k-NN set and to output a *single* class, we perform a reconstruction of analyses combining *all* classes in the k-NN set. To allow for more classes in k-NN's output, we increase k to 3 while keeping the other settings as before. As expected, this approach increases the number of analyses. This, in turn, increases the recall dramatically, up to nearly perfect for known words; see Table 2. However, this gain in recall is at the expense of the precision, which drops dramatically. So, although our revised approach solves the issues above, it introduces massive overgeneration.

3.4 Improving precision

We try to tackle the overgeneration problem by filtering the analyses in two ways. First, by ranking the analyses and limiting output to the n-best. The ranking mechanism relies on the distribution of the classes in the NN-set. Normally, some classes occur more frequently than others in the NN-set. During the reconstruction of a particular analysis, we sum the frequencies of the classes involved. The resulting score is then used to rank the analyses in decreasing order, which we filter by taking the n-best.

The second filter employs the fact that only certain sequences of morphological tags are valid. Tag bigrams are already implicit in the way that the classes are constructed, because a class contains the tags preceding and following the input character. However, cooccurrence restrictions on tags may stretch over longer distances; tag trigram information is not available at all. We therefore derive a frequency list of all tag trigrams occurring in the training data. This information is then used to filter analyses containing tag trigrams occurring below a certain frequency threshold in the training data.

Both filters were optimized on the fold that was used for testing so far, maximizing the overall F-score. This yieled an n-best value of 40 and tag frequency treshold of 250. Next, we ran a 10-fold cross-validation experiment on all data (except the held out data) using the method described in the previous section in combination with the filters. Average scores of the 10 folds are given in Table 3. In comparison with the initial results, both precision and recall on unknown words has improved, indicating that overgeneration and undergeneration can be midly counteracted.

3.5 Discussion

Admittedly, the performance is not very impressive. We have to keep in mind, however, that the task is not an easy one. It includes vowel insertion in ambiguous root forms, which – in contrast to vowel insertion in prefixes and suffixes – is probably irregular and unpredictable, unless the appropriate stem would be known. As far as the evaluation is concerned, we are unsure whether the analyses found in the treebank for a particular word are exhaustive. If not, some of the predictions that are currently counted as precision errors (overgeneration) may in fact be correct alternatives.

Since instances are generated for each type rather than for each token in the data, the effect of token frequency on classification is lost. For example, instances from frequent tokens are more likely to occur in the k-NN set, and therefore their (partial) analyses will show up more frequently. This is an issue to explore in future work. Depending on the application, it may also make sense to optimize on the correct prediction of unkown words, or on increasing only the recall.

4 Part-of-speech tagging

We employ MBT, a memory-based tagger-generator and tagger (Daelemans et al., 1996) to produce a part-of-speech (PoS) tagger based on the ATB1 corpus[2]. We first describe how we prepared the corpus data. We then describe how we generated the tagger (a two-module tagger with a module for known words and one for unknown words), and subsequently we report on the accuracies obtained on test material by the generated tagger. We conclude this

[2]In our experiments we used the MBT software package, version 2 (Daelemans et al., 2003), available from `http://ilk.uvt.nl/`.

```
w      CONJ
bdA    VERB_PERFECT
styfn  NOUN_PROP
knt    NOUN_PROP
nHylA  ADJ+NSUFF_MASC_SG_ACC_INDEF
jdA    ADV
,      PUNC
AlA    ADV
>n     FUNC_WORD
...
```

Figure 4: Part of an ATB1 sentence with unvoweled words (left) and their respective PoS tags (right).

section by describing the effect of using the output of the morphological analyzer as extra input to the tagger.

4.1 Data preparation

While the morphological analyzer attempts to generate all possible analyses for a given unvoweled word, the goal of PoS tagging is to select one of these analyses as the appropriate one given the context, as the annotators of the ATB1 corpus did using the * marker. We developed a PoS tagger that is trained to predict an unvoweled word in context, a concatenation of the PoS tags of its morphemes. Essentially this is the task of the morphological analyzer without segmentation and vocalization. Figure 4 shows part of a sentence where for each word the respective tag is given in the second column. Concatenation is marked by the delimiter +.

We trained on the full ten folds used in the previous sections, and tested on the eleventh fold. The training set thus contains 150,966 words in 4,601 sentences; the test set contains 15,102 words in 469 sentences. 358 unique tags occur in the corpus. In the test set 947 words occur that do not occur in the training set.

4.2 Memory-based tagger generator

Memory-based tagging is based on the idea that words occurring in similar contexts will have the same PoS tag. A particular instantiation, MBT, was proposed in (Daelemans et al., 1996). MBT has three modules. First, it has a lexicon module which stores for all words occurring in the provided training corpus their possible PoS tags (tags which occur below a certain threshold, default 5%, are ignored). Second, it generates two distinct taggers; one for known words, and one for unknown words.

The known-word tagger can obviously benefit from the lexicon, just as a morphological analyzer could. The input on which the known-word tagger bases its prediction for a given focus word consists of the following set of features and parameter settings: (1) The word itself, in a local context of the two preceding words and one subsequent word. Only the 200 most frequent words are represented as themselves; other words are reduced to a generic string – cf. (Daelemans et al., 2003) for details. (2) The possible tags of the focus word, plus the possible tags of the next word, and the *disambiguated* tags of two words to the left (which are available because the tagger operates from the beginning to the end of the sentence). The known-words tagger is based on a k-NN classifier with $k = 15$, the modified value difference metric (MVDM) distance function, inverse-linear distance weighting, and GR feature weighting. These settings were manually optimized on a held-out validation set (taken from the training data).

The unknown-word tagger attempts to derive as much information as possible from the surface form of the word, by using its suffix and prefix letters as features. The following set of features and parameters are used: (1) The three prefix characters and the four suffix characters of the focus word (possibly encompassing the whole word); (2) The possible tags of the next word, and the disambiguated tags of two words to the left. The unknown-words tagger is based on a k-NN classifier with $k = 19$, the modified value difference metric (MVDM) distance function, inverse-linear distance weighting, and GR feature weighting – again, manually tuned on validation material.

The accuracy of the tagger on the held-out corpus is 91.9% correctly assigned tags. On the 14155 known words in the test set the tagger attains an accuracy of 93.1%; on the 947 unknown words the accuracy is considerably lower: 73.6%.

5 Integrating morphological analysis and part-of-speech tagging

While morphological analysis and PoS tagging are ends in their own right, the usual function of the two modules in higher-level natural-language processing or text mining systems is that they jointly determine for each word in a text the appropriate single morpho-syntactic analysis. In our setup, this

	All words		Known words		Unknown words	
Part-of-speech source	Precision	Recall	Precision	Recall	Precision	Recall
Gold standard	70.1	97.8	75.8	99.5	30.2	73.4
Predicted	64.0	89.7	69.8	92.0	23.9	59.0

Table 4: Precision and recall of the identification of the contextually appropriate morphological analysis, measured on all test words and split on known words and unknown words. The top line represents the upper-bound experiment with gold-standard PoS tags; the bottom line represents the experiment with predicted PoS tags.

amounts to predicting the solution that is preceded by "`*`" in the original ATB1 data. For this purpose, the PoS tag predicted by MBT, as described in the previous section, serves to select the morphological analysis that is compatible with this tag. We employed the following two rules to implement this: (1) If the input word occurs in the training data, then look up the morphological analyses of the word in the training-based lexicon, and return all morphological analyses with a PoS content matching the tag predicted by the tagger. (2) Otherwise, let the memory-based morphological analyzer produce analyses, and return all analyses with a PoS content matching the predicted tag.

We first carried out an experiment integrating the output of the morphological analyzer and the PoS tagger, faking perfect tagger predictions, in order to determine the upper bound of this approach. Rather than predicting the PoS tag with MBT, we directly derived the PoS tag from the annotations in the treebank. The upper result line in Table 4 displays the precision and recall scores on the held-out data of identifying the appropriate morphological analysis, i.e. the solution marked by `*`. Unsurprisingly, the recall on known words is 99.5%, since we are using the gold-standard PoS tag which is guaranteed to be among the training-based lexicon, except for some annotation discrepancies. More interestingly, about one in four analyses of known words matching on PoS tags actually mismatches on vowel or consonant changes, e.g. because it represents a different stem – which is unpredictable by our method.

About one out of four unknown words has morphological analyses that do not match the gold-standard PoS (a recall of 73.4); at the same time, a considerable amount of overgeneration of analyses accounts for the low amount of analyses that matches (a precision of 30.2).

Next, the experiment was repeated with *predicted* PoS tags and morphological analyses. The results are presented in the bottom result line of Table 4. The precision and recall of identifying correct analyses of known words degrades as compared to the upper-bounds results due to incorrect PoS tag predictions. On unknown words the combination of heavy overgeneration by the morphological analyzer and the 73.6% accuracy of the tagger leads to a low precision of 23.9 and a fair recall of 59.0. On both known and unknown words the integration of the morphological analyzer and the tagger is able to narrow down the analyses by the analyzer to a subset of matching analyses that in about nine out of ten cases contains the "`* SOLUTION`" word.

6 Related work

The application of machine learning methods to Arabic morphology and PoS tagging appears to be somewhat limited and recent, compared to the vast descriptive and rule-based literature particularly on morphology (Kay, 1987; Beesley, 1990; Kiraz, 1994; Beesley, 1998; Cavalli-Sfora et al., 2000; Soudi, 2002).

We are not aware of any machine-learning approach to Arabic morphology, but find related issues treated in (Daya et al., 2004), who propose a machine-learning method augmented with linguistic constraints to identifying roots in Hebrew words – a related but reverse task to ours. Arabic PoS tagging seems to have attracted some more attention. Freeman (2001) describes initial work in developing a PoS tagger based on transformational error-driven learning (i.e. the Brill tagger), but does not provide performance analyses. Khoja (2001) reports a 90% accurate morpho-syntactic statistical tagger that uses the Viterbi algorithm to select a maximally-likely part-of-speech tag sequence over a sentence. Diab et al. (2004) describe a part-of-speech tagger based on support vector machines that is trained on tokenized data (clitics are separate tokens), reporting a tagging accuracy of 95.5%.

7 Conclusions

We investigated the application of memory-based learning (k-nearest neighbor classification) to morphological analysis and PoS tagging of unvoweled written Arabic, using the ATB1 corpus as training and testing material. The morphological analyzer was shown to attain F-scores of 0.32 on *unknown* words when predicting all aspects of the analysis, including vocalization (a partly unpredictable task, certainly if no context is available). The PoS tagger attains an accuracy of about 74% on unknown words, and 92% on all words (including known words). A combination of the two which selects from the set of generated analyses a subset of analyses with the PoS predicted by the tagger, yielded a recall of the contextually appropriate analysis of 0.90 on test words, yet a low precision of 0.64 largely caused by overgeneration of invalid analyses.

We make two final remarks. First, memory-based morphological analysis of Arabic words appears feasible, but its main limitation is its inevitable inability to recognize the appropriate stem of unknown words on the basis of the ambiguous root form input; our current method simply overgenerates vocalizations, keeping high recall at the cost of low precision. Second, memory-based PoS tagging of written Arabic text also appears to be feasible; the observed performances are roughly comparable to those observed for other languages. The PoS tagging task as we define it is deliberately separated from the problem of vocalization, which is in effect the problem of stem identification. We therefore consider the automatic identification of stems as a component of full morpho-syntactic analysis of written Arabic an important issue for future research.

References

D. W. Aha, D. Kibler, and M. Albert. 1991. Instance-based learning algorithms. *Machine Learning*, 6:37–66.

K. Beesley. 1990. Finite-state description of Arabic morphology. In *Proceedings of the Second Cambridge Conference: Bilingual Computing in Arabic and English*.

K. Beesley. 1998. Consonant spreading in Arabic stems. In *Proceedings of COLING-98*.

T. Buckwalter. 2002. Buckwalter Arabic morphological analyzer version 1.0. Technical Report LDC2002L49, Linguistic Data Consortium. available from `http://www.ldc.upenn.edu/`.

V. Cavalli-Sfora, A. Soudi, and M. Teruko. 2000. Arabic morphology generation using a concatenative strategy. In *Proceedings of the First Conference of the North-American Chapter of the Association for Computational Linguistics*, Seattle, WA, USA.

T. M. Cover and P. E. Hart. 1967. Nearest neighbor pattern classification. *Institute of Electrical and Electronics Engineers Transactions on Information Theory*, 13:21–27.

W. Daelemans, J. Zavrel, P. Berck, and S. Gillis. 1996. MBT: A memory-based part of speech tagger generator. In E. Ejerhed and I. Dagan, editors, *Proceedings of Fourth Workshop on Very Large Corpora*, pages 14–27. ACL SIGDAT.

W. Daelemans, A. van den Bosch, and J. Zavrel. 1999. Forgetting exceptions is harmful in language learning. *Machine Learning, Special issue on Natural Language Learning*, 34:11–41.

W. Daelemans, J. Zavrel, A. van den Bosch, and K. van der Sloot. 2003. MBT: Memory based tagger, version 2.0, reference guide. ILK Technical Report 03-13, Tilburg University.

W. Daelemans, J. Zavrel, K. van der Sloot, and A. van den Bosch. 2004. TiMBL: Tilburg memory based learner, version 5.1, reference guide. ILK Technical Report 04-02, Tilburg University.

E. Daya, D. Roth, and S. Wintner. 2004. Learning Hebrew roots: Machine learning with linguistic constraints. In *Proceedings of EMNLP'04*, Barcelona, Spain.

M. Diab, K. Hacioglu, and D. Jurafsky. 2004. Automatic tagging of arabic text: From raw text to base phrase chunks. In *Proceedings of HLT/NAACL-2004*.

A. Freeman. 2001. Brill's POS tagger and a morphology parser for Arabic. In *ACL/EACL-2001 Workshop on Arabic Language Processing: Status and Prospects*, Toulouse, France.

M. Kay. 1987. Non-concatenative finite-state morphology. In *Proceedings of the third Conference of the European Chapter of the Association for Computational Linguistics*, pages 2–10, Copenhagen, Denmark.

S. Khoja. 2001. APT: Arabic part-of-speech tagger. In *Proceedings of the Student Workshop at NAACL-2001*.

G. Kiraz. 1994. Multi-tape two-level morphology: A case study in semitic non-linear morphology. In *Proceedings of COLING'94*, volume 1, pages 180–186.

J. McCarthy. 1981. A prosodic theory of non-concatenative morphology. *Linguistic Inquiry*, 12:373–418.

A. Soudi. 2002. *A Computational Lexeme-based Treatment of Arabic Morphology*. Ph.D. thesis, Mohamed V University (Morocco) and Carnegie Mellon University (USA).

A. van den Bosch and W. Daelemans. 1999. Memory-based morphological analysis. In *Proceedings of the 37th Annual Meeting of the ACL*, pages 285–292, San Francisco, CA. Morgan Kaufmann.

C.J. van Rijsbergen. 1979. *Information Retrieval*. Buttersworth, London.