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

Antal van den Bosch*, Erwin Marsi*, and Abdelhadi Soudi**
* ILK / Dept. of Language and Information Science
Faculty of Arts, Tilburg University
P.O. Box 90153, NL-5000 LE Tilburg, The Netherlands
email: {Antal.vdnBosch,E.C.Marsi}@uvt.nl
** Ecole Nationale de l'Industrie Minérale
Rabat, Morocco
email: asoudi@gmail.com

## 1 Introduction

Memory-based learning has been successfully applied to morphological analysis and part-of-speech tagging in Western and Eastern-European languages (Daelemans et al., 1996; Van den Bosch and Daelemans, 1999; Zavrel and Daelemans, 1999). With the release of the Arabic Treebank by the Linguistic Data Consortium, 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 (Diab, Hacioglu, and Jurafsky, 2004); cf. Chapter 9.

However, as argued and illustrated throughout this book, Arabic offers special challenges for data-driven and knowledge-based approaches alike. An Arabic word may be composed of a stem consisting of a consonantal root and a pattern, and may furthermore contain affixes and clitics. Arabic verbs, for instance, can be conjugated according to one of the traditionally recognized patterns. There are 15 triliteral forms, of which at least 9 are 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.

In this chapter 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. Section 3 describes the data used throughout the study for both tasks. The subsequent three sections describe our work on memory-based morphological analysis (Section 4) and its integration with part-of-speech tagging (Section 5). The final Section 6 contains a short discussion of related work and offers an overall conclusion.

## 2 Memory-based learning

Memory-based learning, also known as instance-based, example-based, or lazy learning (Aha, Kibler, and Albert, 1991; Daelemans, Van den Bosch, and Zavrel, 1999), based on 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 naturally allows it to deal with productive but low-frequency exceptions (Daelemans, Van den Bosch, and Zavrel, 1999).

An instance consists of a fixed-length vector of $n$ feature-value pairs, and an information field containing 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 function $\Delta(X, Y)$ between the new instance $X$ and the memory instance $Y$. The most primitive distance function is the "overlap" function $\Delta(X, Y) = \sum_{i=1}^{n} w_i \, \delta(x_i, y_i)$, where $n$ is the number of features, $w_i$ is a weight for feature $i$, and $\delta = 0 \; if \; x_i = y_i, \; else \; 1$ is the distance per feature. In this chapter we use the somewhat more complex Modified Value Difference Metric (MVDM) distance function (Stanfill and Waltz, 1986; Cost and Salzberg, 1993), which determines the similarity of pairs of values of a feature by looking at the conditional probabilities (estimated on co-occurrence counts) of the two values conditioned on the classes: $\delta(v_1, v_2) = \sum_{i=1}^{n} |P(C_i|v_1) - P(C_i|v_2)|$.

In this chapter the weight (importance) of a feature $i$, $w_i$, is estimated by computing its *gain ratio*, $GR_i$. To compute a feature's $GR$, we first compute its *information gain* $IG_i$, which is the difference in uncertainty (entropy) within the set of cases between the situations without and with knowledge of the value of that feature: $IG_i = H(C) - \sum_{v \in V_i} P(v) \times H(C|v)$, where $C$ is the set of class labels, $V_i$ is the set of values for feature $i$, and $H(C) = -\sum_{c \in C} P(c) \log_2 P(c)$ is the entropy of the class labels. The probabilities are estimated from frequency counts in the training set. To derive the GR, the feature's IG is divided by the entropy of the feature values, the *split info* $si_i = -\sum_{v \in V_i} P(v) \log_2 P(v)$: $GR_i = \frac{IG_i}{si_i}$.

Classification in memory-based learning is performed by the $k$-NN algorithm that searches for the $k$ 'nearest neighbors' according to the $\Delta(X, Y)$ function. The majority class of the $k$ nearest neighbors then determines the class of the new case. With symbolic feature values, distance ties can occur when two nearest neighbors mismatch with the test instance on the same feature value, while all three instances have different values. In the $k$-NN implementation[1] we used, equidistant neighbors are taken as belonging to the same $k$, so this implementation is effectively a $k$-nearest distance classifier. This implies that when $k = 1$, more than one nearest neighbor may be found, all at the same distance to the test instance.

## 3 Data

Our point of departure is the Arabic Treebank 1 (ATB1), version 2.0, distributed by LDC[2], more specifically the "after treebank" POS-tagged data, consisting of 166,068 tagged words. Tokens are vocalized and morphologically analyzed by means of Tim Buckwalter's Arabic Morphological Analyzer (Buckwalter, 2002). 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

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

[2] LDC: `http://www.ldc.upenn.edu/`.

```
INPUT STRING: كتب

LOOK-UP WORD: ktb
     Comment:
       INDEX: P2W20
  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 according to Buckwalter's transliteration (cf. Table 1 in Chapter 2).

portion of the solution relevant to our experiments is the segmentation, with plus markers (+) as segment boundary markers, and slashes (/) as delimiters between the character strings and the part-of-speech information related to that string. For example, the segmented string `kutub/NOUN+a/CASE_DEF_ACC` represents the substring `kutub` being a noun, and `a` carrying the morpho-syntactic function `CASE_DEF_ACC` (underscores occur within multi-word tags, and have no relation with segmentation). The resulting analyses have been manually annotated with a preceding star (*), marking the correct solution in the given context.

Throughout the corpus the number of analyses listed per word is not constant, probably as a result of additions and/or deletions by the annotators. As our goal is to predict all possible analyses for a given word, we first created a lexicon that maps every word to all analyses encountered and their respective frequencies; see Figure 2 for an example. In the course of this process, we removed all vowels, since we aim at analyzing unvoweled words, producing unvoweled analyses. Vowel generation ultimately implies stem identification, a complication which we do not address here; our goal is the segmentation of unvoweled strings and an appropriate identification of the morpho-syntactic function of each of the parts, so that the output can later be integrated with a part-of-speech tagger (see Section 5).

Also, we chose to re-attach clitic tokens (e.g. determiners and prepositions) to their host,

```
ktb
==> ktb/PV+/PVSUFF_SUBJ:3MS+ 7
==> k/PREP+tb/NOUN_PROP+ 7
==> ktb/PV_PASS+/PVSUFF_SUBJ:3MS+ 7
==> ktb/NOUN+/CASE_DEF_ACC+ 7
==> ktb/NOUN_PROP+ 7
==> ktb/NOUN+/CASE_DEF_NOM+ 8
==> ktb/NOUN+K/CASE_INDEF_GEN+ 7
==> ktb/NOUN+ 7
==> ktb/NOUN+N/CASE_INDEF_NOM+ 7
==> ktb/NOUN+/CASE_DEF_GEN+ 8
```

Figure 2: Example of a preprocessed lexical entry for the word ktb كتب, carrying ten morphological analyses, using Buckwalter's transliteration (cf. Table 1 in Chapter 2).

storing them together as a single word form. The lexicon derived from the full corpus, excluding punctuation markers and words without a stored solution, contains 16,626 unique words and 113,105 analyses; an average of 6.8 analyses per word.

To evaluate our system, we need data which can be regarded as realistic test material, including a typical amount of unknown words, representing any new document of text the system may be applied to. More realistically, we require any news article of the type that the ATB1 corpus is composed of. To this purpose, we split the complete part-of-speech tagged ATB1 corpus into eleven partitions of near-equal size. The data is shuffled randomly at the article level. One of these eleven partitions is set apart as a held-out set for later use, described further below. Subsequently, ten pairs of training and test sets are generated using the ten remaining 1/11th partitions. Each training set consists of a concatenation of nine of the ten partitions, while each tenth partition is taken as the test set accompanying the training set. Repeating this ten times, we thus create ten overlapping training sets that each consist of 9/11th of the original ATB1 tagged corpus, and ten corresponding non-overlapping test sets that each represent a 1/11 portion of the original ATB1 corpus. With this 10-fold cross-validation setup, we use the same training-test set partitions for training both the morphological analysis system as well as the part-of-speech tagger described in the next section.

The eleventh held-out set, the remaining 1/11th portion of the shuffled ATB1 corpus, is used in both experimental sections to estimate the generalization performance of both modules individually, and also to estimate the joint generalization performance of the part-of-speech tagger and morphological analyzer together. We use this single held-out split for methodological reasons; both the morphological analyzer and the part-of-speech tagger are tested using a range of algorithmic parameter values on the 10-fold partitionings (e.g., the value of $k$ of the $k$-nearest neighbor classifier). Since an estimated generalization performance of both modules cannot be based on the optimized test performance in a range of experiments, a fully unknown test set is needed - hence the eleventh set.

We conclude the section with some lexical statistics. Taken separately, each 1/11th partition contains about 15,100 tokens and about 4,010 unique words. The ten 9/11th training sets on average contain about 135,900 tokens, and about 15,100 unique words. Importantly, the number of unknown word tokens in the ten test sets that do not occur in their respective

training sets is 977 on average, most of them occuring once (76%): about 21% of all unique words (i.e., word types). Since most unknown words have a low frequency, they only represent about 6.5% of all tokens in a test set. On the difference between word tokens and word types, the following should be noted. As will be argued in the subsequent sections, we define morphological analysis as a task on word types, and part-of-speech tagging as a task on word tokens. Henceforth, we will refer to "words" when we mean word tokens, and "word types" when we mean word types.

# 4 Morphological analysis

The goal of the memory-based morphological analysis system we describe here is to generate no more and no less than all possible morphological analyses of an unvoweled input word that has not occurred before in the analyzer's training material. For all occurrences of a word we want to generate the same analyses; hence, this is a task at the word type level. An analysis consists of a proper segmentation of clitics, stems, and suffixes, brought to their canonical orthographic form, and with the correct identification of the morpho-syntactic part-of-speech tag of each segment.

This means our system is literally a morphological analysis generator for word types. To measure its ability to generate no more and no less than all possible analyses, we employ the dual concepts of precision (the percentage of generated analyses that is actually correct) and recall (the percentage of target analyses that the analyzer was able to generate). Also, as said before, we focus solely on the capability of the analyzer to generate analyses for word types it has not seen before, henceforth referred to as *unknown words*. We assume that a typical morphological analysis system has a lexicon at hand, allowing the system to reproduce all morphological analyses of known word types flawlessly. The problem is, of course, that typically a non-trivial portion of all words in a text are unknown words. In our experiments an unknown set of texts contains about 6.5% unknown words, or 24% of the word types.

In this section we first describe how we created the data used in our experiments. We then describe the experiments performed on these data, focusing our analyses on the precision and recall scores on unknown words. We conclude the section with a critical discussion of our results.

The experiments on training a morphological analysis system need an additional processing step, namely the extraction of a lexicon from each training set. Each lexicon contains for each word type in the training set all possible morphological analyses. This is done as described above. The same procedure is followed for the test set, except that here we ignore the words already in the training set; we focus on unknown words only. Hence, each test set is converted to a small lexicon of all unknown words that do not occur in the corresponding training set, listing for each unknown word the one or more analyses which they actually get in the annotated corpus. The goal of our morphological system is to generate precisely these analyses.

## 4.1 Creating instances for morphological analysis

The lexical entries generated by preprocessing both the training set and the test set of each partition are converted to instances suitable to memory-based learning of the mapping from words to their analyses (Van den Bosch and Daelemans, 1999). Instances represent input (the orthographic word) and their corresponding output (the morphological analysis). Since

```
= = = = = k t b = = =  _-/NOUN-/NOUN
= = = = k t b = = = =  /NOUN-/NOUN-DK/NOUN/CASE_INDEF_GEN
= = = k t b = = = = =  /NOUN-DK/NOUN/CASE_INDEF_GEN-_
```

Figure 3: Instances for the analyses of the word ktb كتب in Figure 2.

instances need to be of a fixed length and since they need to be general enough to generalize from known to unknown words, instances do not map entire words to entire analyses (which would render them case-specific), but rather represent partial fixed-width snapshots of words mapping to subsequences of the analysis. More specifically, the mapping is broken down into smaller letter-by-letter classification tasks.

The input of each instance, consisting of a fixed number of *features*, is created by sliding a window over the input word, resulting in one instance for each letter. Using a 5-1-5 window yields 11 features, i.e. the input letter in focus, plus the five preceding and five following letters. The equality mark (=) is used as a filler symbol for positions beyond the beginning or end of the word. To illustrate this, consider the seventh analysis of the word ktb كتب in Figure 2, `ktb/NOUN+K/CASE_INDEF_GEN+`, representing a stem (`ktb`, noun = kutub "books"), followed by a suffix (`K`) carrying the `CASE_INDEF_GEN` function. The `K` suffix is not realized in the surface form. This three-letter word with this particular analysis then results in the three instances displayed in Figure 3.

The class of the first instance, `_-/NOUN-/NOUN`, is a trigram (with the character `-` as the delimiter), marking the fact that the letter `k` is the first letter of a noun stem, and that the second letter `t` is also inside the same noun stem. The class of the second instance, `/NOUN-/NOUN-DK/NOUN/CASE_INDEF_GEN`, signals in the rightmost part of the trigram that the third letter, `b`, marks the end of the noun stem and also carries the unrealized `CASE_INDEF_GEN` function; at the same time, the `DK` code denotes that a `K` was deleted. Hence, to reconstruct the analysis, a `K` needs to be reinserted. In general, the class codes making up the three parts of the class trigram always encode the part-of-speech tag of the morpheme the corresponding letter belongs to. Optionally this tag is preceded by a code representing the insertion, deletion, or replacement of one or more letters that are required to generate the underlying forms of the morphemes in the analysis, coded by the letters `I`, `D`, and `R`, respectively, followed by the letters themselves. Segmentation is encoded implicitly; whenever a letter is associated to another part-of-speech tag than its preceding neighbor, then a morpheme segmentation boundary exists between them. Based on this complex information, a complete analysis can be constructed.

In principle, unigram classes could already be used for this purpose. If predicted correctly, an analysis would follow from the straightforward concatenation of position-specific classes. However, we are faced with an average of 6.8 analyses per word. The example word ktb كتب has ten analyses, as illustrated in Figure 2. The first letter is associated within these ten analyses with five different unigram classes: `PV`, `PREP`, `PV_PASS`, `NOUN`, and `NOUN_PROP`. The second letter is linked to four tags: `PV`, `NOUN`, `PV_PASS`, and `NOUN_PROP`. Finally, the third letter is associated with seven different tags. If predicted in isolation, the system would have the task to pick the correct ten analyses from the maximal cartesian product of $5 \times 4 \times 9 = 180$ combined analyses. Due to their redundant overlap, the trigrams offer the key to this search. Since the $k$-nearest distances classifier is used, it will always produce all ambiguous analyses at

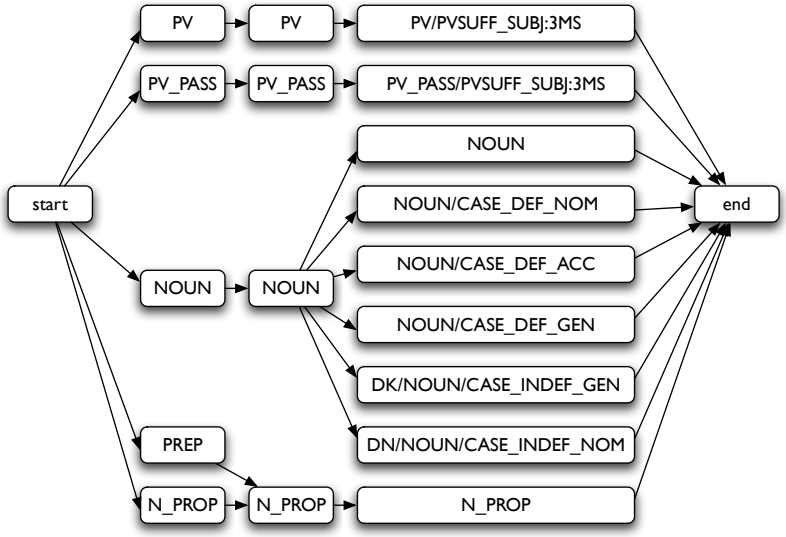


Figure 4: The lattice formed by the overlapping trigrams generated by three consecutive position-specific classifications, using all nearest neighbors at distance $k = 1$. The lattice contains exactly the ten possible paths encoding the ten morphological analyses of ktb كتب.

the same distance. Hence, all 5, 4, and 9 position-specific overlapping trigrams will be present in the ideal case that the classifier maps to the correct nearest neighbors. The trigrams then span up a lattice, illustrated in Figure 4, in which exactly ten paths are possible.

While this example is perfect, less complete lattices are possible when a trigram does not match with one of the trigrams generated on the next letter. This will never occur with known words, but as said, we focus on unknown words, and then it is quite possible that a nearest-neighbor classification of the first letter of the word yields a completely different trigram than the nearest-neighbor classification of the second word. In these cases, paths in the lattice do not connect to the end node since they are generated by a trigram classification that does not match on the right hand side, or do not start in the start node since they are generated by a trigram that does not match on the left hand side. We apply the hard constraint that a valid path, encoding a complete analysis, must lead from the start node to the end node; hence, a mismatching trigram generates at least one invalid path. This also means that completely mismatching trigrams may lead to a lattice without valid paths, and no morphological analysis is generated.

Figure 5 displays an actual lattice generated for the unknown word AHtfAl احتفَال; eight of the 24 intermediate nodes are part of invalid paths. Two of these paths do not connect to either the start or the end node; these are verbal analyses generated while focusing on the fourth letter, f ف. Eventually, six paths are valid, i.e., six analyses can be generated. Five of them are correct; the analysis with `DN/NOUN` misses a `CASE_INDEF_NOM` tag, and there is also an analysis with `CASE_INDEF_GEN` that cannot be generated from this lattice.

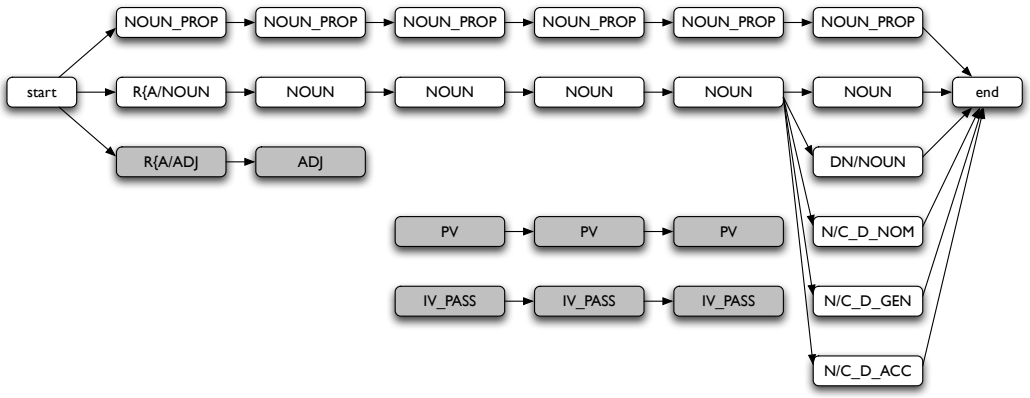


Figure 5: The lattice formed by the overlapping trigrams generated by six consecutive position-specific classifications, using the five $k$-nearest distances, generated for the unknown word AHtfAl احتفَال. Grey nodes in the lattice represent nodes in invalid paths, i.e. they are not used for the generation of analyses. "N/C_D" stands for `NOUN/CASE_DEF`.

### 4.2 Evaluation

Performance is evaluated on the generated complete analyses (i.e. all valid paths in a generated lattics), where an analysis is considered complete if all of its part-of-speech and spelling change information is fully correct. Note again that the analyses concerns word types; the morphological analyzer is trained on word types from the training set, and is applied to unknown word types in the test set. Since we are concerned with measuring the degree of undergeneration or overgeneration of analysis generation, we quantitatively measure the performance of our system in terms of precision, recall, and F-score (Van Rijsbergen, 1979). Precision (1), the ratio of true positives (TP) over the total number of true and false positives (FP), measures to what extent overgeneration of analyses occurs, whereas recall (2), the ratio of true positives over the sum of true positives and false negatives (FN), measures the amount of undergeneration. The F-score (3), the harmonic average between precison and recall, represents an overall fit between real and predicted analyses.

$$precision = \frac{TP}{TP + FP} \tag{1}$$

$$recall = \frac{TP}{TP + FN} \tag{2}$$

$$Fscore = \frac{2 * precision * recall}{precision + recall} \tag{3}$$

### 4.3 Experiments

We performed 10-fold cross-validation experiments, measuring the precision and recall on the generated analyses for the unknown words in the test sets. Figure 6 displays the precision-recall curve with different values of the $k$-parameter of the $k$-nearest neighbor classifier. Starting at $k = 1$, when the nearest-neighbor classifier just uses the nearest-distance instances in

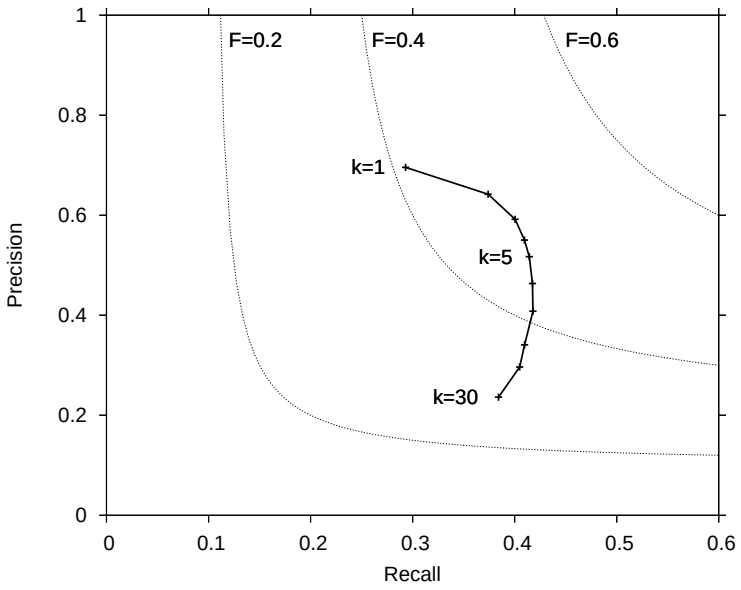


Figure 6: Precision-recall curve on the generation of morphological analyses, with increasing values of $k$, from 1 up to 30. The F-isolines represent the lines in the space with a particular value of F in steps of 0.2.

memory to generate the analysis, a precision level of 0.70 is attained; this means that about seven out of ten generated analyses are correct. The downside is that at $k = 1$, recall is only 0.29, meaning that the system is only able to generate slightly under one in three of the analyses it should generate. Recall can be increased to 0.42 with higher values of $k$, peaking at $k = 10$ (precision at that point is down to 0.40), but slightly decreasing with $k > 10$. The peak F-score of 0.47 (the highest harmonic mean of precision and recall) is attained at $k = 3$.

Clearly this performance is not impressive. We have to keep in mind, however, that the task is not an easy one: the evaluation concerns *unknown* words that were not in the learner's training material.

# 5 Integration with 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[3]. 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.

## 5.1 Data preparation

While the morphological analyzer attempts to generate all possible analyses for a given word, the goal of POS tagging is to select one of these analyses as the appropriate one given the

[3]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
h       PRON_3MS
Agtsl   VERB_PERFECT
w       CONJ
...
```

Figure 7: Part of an ATB1 sentence with words (left) and their respective POS tags (right), according to Buckwalter's transliteration (cf. Table 1 in Chapter 2).

context, as the annotators of the ATB1 corpus did manually with the * marker. We developed a POS tagger that is trained to predict, for any given word, a concatenation of the POS tags of its morphemes. This is essentially the morphological analysis of a word in which segmentation information and letter transformations are discarded. Figure 7 shows part of a sentence where for each word the respective tag is given in the second column.

Concatenation is encoded by the delimiter +. Some words have no solution at all, but annotator comments usually tag them as proper nouns – hence we gave all of these words a NOUN_PROP tag.

Due to the concatenation of all morpho-syntactic information, the tag set is quite substantial: 306 different tags occur in the ten folds extracted from the ATB1 corpus. The five most frequent tags are PREP (13%), PUNC (10%), NOUN_PROP (7%), CONJ (4%), and DET+NOUN+CASE_DEF_GEN (4%). About 10% of the tags occur only once (and will therefore never be predicted correctly), and about 33% of all tags occur less than 10 times.

We used the same partitionings of the ATB1 corpus as used to develop the morphological analyzer; we also run a 10-fold cross validation experiment. Evaluation differs, though: we measure the percentage of word tokens that are given the correct tag, i.e. a straightforward accuracy score. We also split this score into one on known words (already encountered in the training data) and unknown words (not encountered in the training data, i.e., the same words as focused on when evaluating the output of the morphological analysis system). Different from morphological analysis, we cannot assume that we are able to perform perfectly on words that are already known from a training set; in part-of-speech tagging, having seen a word in training only means that the tagger knows some of the tags (hopefully, all of the tags) that a word may have in different contexts. The tagger still needs to decide which tag is appropriate for every word token.

## 5.2 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 by Daelemans et al. (1996).

| Accuracy (% correct POS tags) | | |
|---|---|---|
| Known words | Unknown words | All words |
| 93.3 | 66.4 | 91.5 |
| ±0.6 | ±2.1 | ±0.6 |

Table 1: POS tagging accuracies on known words, unknown words, and all test data (% correctly tagged test words), with standard deviations.

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. If a word has one unique tag in the training set, it will likely have the same tag when reoccurring in test material. If a word has a handful of possible tags, then the tagger's task is reduced to selecting the appropriate one from this limited list.

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 one of the test partitions.

The unknown-word tagger obviously does not know the possible tags the unknown word can have. Instead, it 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 letters and the four suffix letters 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 one test set.

### 5.3 Evaluating the tagger

Table 1 lists the average accuracy (percentage of correctly tagged test words) of MBT as measured in the 10-fold cross-validation experiment. The general accuracy, 91.5%, is reasonable; known words are tagged with an accuracy of 93.3%. The unknown-words tagger has a lower accuracy than the known-words tagger, but it is able to tag 66.4% of the 6.5% unknown test words correctly nevertheless.

### 5.4 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 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.

As a concluding experiment, we trained MBT with optimized settings on the complete concatenated ten folds of our original experiment, and tested the tagger on the eleventh held-out partition. As can be expected with somewhat more training data available, we observe a slightly higher overall tagging accuracy on the held-out data of 92.0%, with 93.4% on known words, and 71.0% on the 672 unknown words in this partition.

Subsequently we also trained a morphological analyzer on the full concatenated ten folds, and tested it on the held-out set, with the $k = 10$ setting that yielded the highest recall in the 10-fold cross-validation experiment. Recall is more important than F-score or precision, since higher recall will generally improve the likelihood of a match with the part-of-speech tags. Training the analyzer on the full ten folds and testing on the held-out set yields a precision of 0.41, a recall of 0.43, and an F-score of 0.42.

Finally, we computed how well the generated morphological analyses for unknown words could be integrated with the predicted tags for these words. We first computed an upper bound score by comparing the generated analyses against the gold standard annotated tags of all unknown words. It turned out that in 77.5% of all unknown words one of the analyses generated actually matches with the correct gold-standard part-of-speech tag. To illustrate this, consider the unknown word *yxfy*, which is tagged as `IV3MS+IV+IVSUFF_MOOD:I`. The three analyses generated for this word, reduced to only the concatenation of the part-of-speech tags (leaving out letter transformations and the specific segmentation position information) are

1. `NOUN_PROP`
2. `IV3MS+IV_PASS`
3. `IV3MS+IV+IVSUFF_MOOD:I`

Since one of the analyses matches the gold-standard tag, this word is counted as one of the 69.1% of the unknown words of which the full analysis could be generated, i.e., of which the solution marked with a `*` in the ATB1 corpus could be identified.

The actual agreement with the predicted tag is 75.0%, which is also high; however, the most relevant score is the intersection between the cases where both the tagger is correct, and one of the morphological analyses matches with the tag. In 82.2% of the cases in which the predicted tag is correct, a matching morphological analysis is also generated. Measured at the level of all unknown words, including the words that receive an incorrect tag, we find that of all unknown words in the held-out set 58.1% are assigned both a correct tag and a completely correct and matching morphological analysis, including segmentation and letter transformations.

## 6 Discussion and conclusion

In this chapter we investigated the application of memory-based learning ($k$-nearest neighbor classification) to morphological analysis and POS tagging of written Arabic, using the ATB1 corpus as training and testing material. The morphological analyzer, when optimized on recall, was shown to attain a precision of 0.41, a recall of 0.43, and an F-score of 0.42 on *unknown* word types in held-out data when predicting all aspects of the analysis: part-of-speech tags of the segments, the positions of the segmentations, and all letter transformations between the surface form and the analysis. The POS tagger, in turn, attained an accuracy of 66.4% on unknown words, and 91.5% on all words (including known words) in held-out data. A combination of the two which selects one full morpho-syntactic analysis out of the generated analyses, matching the part-of-speech predicted by the tagger, yields a joint accuracy of 58.1% fully correctly predicted tags and corresponding full analysis for unknown words.

Extrapolating this number to a score on all words in a text, i.e. including the known words, we assume (safely) that our training-set-based lexicon always produces a matching analysis for all known words, for which we have observed the 93.3% accuracy of the part-of-speech tagger. Since known words make up 93.5% of test data, on average, we estimate that we can generate correct tags with complete morphological analyses for $(0.935 \times 0.933) + (0.065 \times 0.581) = 91.0\%$ of all words in unseen text.

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; Soudi, 2002). We are not aware of any machine-learning approach to Arabic morphology. POS tagging, on the other hand, seems to have attracted some focus. 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. In Chapter 9 of this book, Diab *et al.* describe a part-of-speech tagger based on support vector machines that is trained on tokenized data, reporting a tagging accuracy of 95.5%.

The use of trigrams in the output space has been proposed by Van den Bosch and Daelemans (2006), and demonstrated on morphological analysis of Dutch and English words by Van den Bosch, Schuurman, and Vandeghinste (2006). In this chapter, trigram classes are used differently, however; here, the problem is not only in optimizing the class label prediction, but also in limiting the overgeneration of analyses.

We make two final remarks. First, memory-based morphological analysis of Arabic words is feasible, but its main limitation is its inability to recognize the stem of an unknown word, and consequently the appropriate vowel insertions. Also, its guess on the possible POS tags of an unknown word turned out to be less useful in our tagging approach than using the raw prefix and suffix letters of the words themselves, as witnessed by the scores on unknown words of the POS subtagger specialized in unknown words. Second, memory-based POS tagging of written Arabic text appears to be successful, because performance is comparable to that for other languages. The POS tagging task as we define it, is deliberately separated from the problem of vowel insertion, 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.

**Acknowledgements**

The work of the first two authors is funded by the Netherlands Organisation for Scientific Research (NWO). The authors wish to thank Walter Daelemans and Sander Canisius for discussions and feedback.

# References

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

Beesley, K. 1990. Finite-state description of Arabic morphology. In *Proceedings of the Second Cambridge Conference: Bilingual Computing in Arabic and English*, page no pagination.

Beesley, K. 1998. Consonant spreading in Arabic stems. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Montréal, Quebec, Canada*, pages 117–123.

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

Cost, S. and S. Salzberg. 1993. A weighted nearest neighbour algorithm for learning with symbolic features. *Machine Learning*, 10:57–78.

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

Daelemans, W., 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.

Daelemans, W., 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 the Fourth Workshop on Very Large Corpora*, pages 14–27. ACL SIGDAT.

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

Daelemans, W., J. Zavrel, K. Van der Sloot, and A. Van den Bosch. 2004. TiMBL: Tilburg Memory Based Learner, version 5.1.0, reference guide. Technical Report ILK 04-02, ILK Research Group, Tilburg University.

Diab, M., K. Hacioglu, and D. Jurafsky. 2004. Automatic tagging of arabic text: From raw text to base phrase chunks. In *Proceedings of HLT-NAACL 2004*, pages 149–152, Boston, MA.

Freeman, A. 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. Available on: `http://www.elsnet.org/acl2001-arabic.html`.

Kay, M. 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.

Khoja, S. 2001. APT: Arabic part-of-speech tagger. In *Proceedings of the Student Workshop at NAACL-2001*, pages 20–25.

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

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

Stanfill, C. and D. Waltz. 1986. Toward memory-based reasoning. *Communications of the ACM*, 29(12):1213–1228, December.

Van den Bosch, A. 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.

Van den Bosch, A. and W. Daelemans. 2006. Improving sequence segmentation learning by predicting trigrams. In *Proceedings of the Ninth Conference on Natural Language Learning, CoNLL-2005*, pages 80–87, Ann Arbor, MI.

Van den Bosch, A., I. Schuurman, and V. Vandeghinste. 2006. Transferring PoS-tagging and lemmatization tools from spoken to written Dutch corpus development. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation, LREC-2006*, Trento, Italy.

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

Zavrel, J. and W. Daelemans. 1999. Recent advances in memory-based part-of-speech tagging. In *VI Simposio Internacional de Comunicacion Social*, pages 590–597.