Automated part-of-speech analysis of Urdu: conceptual and technical issues

Andrew Hardie

Department of Linguistics and English Language
Lancaster University, UK

a.hardie@lancaster.ac.uk

Abstract

Part-of-speech (POS) tagging is the process of labelling tokens in a text with tags that indicate their morphosyntactic category, and has a wide range of applications in computational and corpus linguistics, such as the production of corpus-based dictionaries and grammars. This paper describes an experiment in extending POS tagging to a hitherto untagged language, Urdu.

The most challenging task in POS tagging is *disambiguation*, i.e. the resolution of the contextual ambiguity of a token for which more than one tag is possible. Three important approaches to disambiguation have been developed: approaches based on rules devised by a linguist; probabilistic approaches based on the application of corpus-derived statistics in a mathematical model such as a Markov model; and Brill (1995)'s approach where rules are learned automatically from a corpus. However, given that only a small amount of pre-tagged data was available for Urdu, only the rule-based approach was appropriate for the Urdu tagger described here. A rule-based tagger for Urdu was created within the *Unitag* architecture, together with the requisite language-specific resources for Urdu (including a tagset, an analyser, a lexicon, and a rule list).

An evaluation of the tagger suggests that it performs at a level of accuracy notably below that commonly reported for languages such as English. However, this poor performance is primarily attributable to the small size of the lexicon, which is attributable to the small quantity of training data available. The rule-based disambiguation rules was more successful.

1. Introduction

In this paper, an experiment in part-of-speech (POS) tagging for Urdu[1] is described. After a brief discussion of the background of past work in POS tagging, then choice of tagging methodology for Urdu texts is justified, and the software and language-specific resources of which the tagging system is composed are outlined. Finally, the results of the experiment, in terms of the success rate of the tagger, are evaluated and, insofar as there are shortcomings, these are explained.

2. Background

POS tagging[2] may be defined as the process of assigning to each word in a running text a label which indicates the status of that word within some system of categorising the words of that language according to their morphological and/or syntactic properties (frequently known as a "tagset")[3]. These "categories" are often similar to, or subdivisions of, the eight parts of speech recognised by grammarians in the Latin/Greek tradition (see Voutilainen 1999a: 3-4).

POS tagging has played an important part in the development of corpus linguistics. A variety of analyses and statistics can be obtained from a corpus – for

[1] The research reported in this paper is discussed in greater detail in Hardie (2004).
[2] The process of "part-of-speech (POS) tagging" is also frequently referred to as "morphosyntactic annotation" (e.g. by Leech and Wilson 1999), or "(syntactic) wordclass tagging" (e.g. by authors in van Halteren 1999). I use these terms interchangeably.
[3] Similar definitions are given by Leech (1987: 8) and van Halteren (1999: xiii).

example word frequency counts, concordances, collocations and keywords/key categories. However, the value of a corpus is considerably greater when it is marked up with POS tags. For instance, with POS tags, it is possible for a concordancer to distinguish between *cut*_NOUN and *cut*_VERB, rather than treating these homonyms as the same word.

There are a wide variety of applications for POS tagging software and tagged text. These include information retrieval, spelling- and grammar-checking, speech processing, handwriting recognition, machine translation, production of corpus-based dictionaries and grammars, and applications in the teaching of foreign languages and knowledge of grammar (see Leech and Smith 1999). POS tagging is a key component of human language technology and corpus research. This paper describes an experiment in extending this tried and tested technique to a language it has not hitherto been applied to: Urdu.

3. Tagging methodologies

This section provides a brief overview of how various different methods of POS tagging work, before going on to justify the choice of a tagging methodology for use with Urdu tests and corpora.

There are three main sub-tasks that a full automatic tagging system must accomplish, which van Halteren and Voutilainen (1999: 109) describe as follows: "segmentation of the text into tokens; assignment of potential tags to tokens, usually resulting in ambiguity; determination of the contextual appropriateness of each potential tag, usually in order to remove the less appropriate tags". We may refer to these tasks as *tokenisation*, *analysis* and *disambiguation* respectively.

This conceptual division is often an actual division in terms of the software used: a separate program or module undertakes each task. This is, for example, true of the CLAWS tagger (Garside, Leech and Sampson 1987). Tokenisation and analysis are straightforward conceptually, if not always practically. However, various methodologies exist for the process of disambiguation. The three best known are rule-based disambiguation, probabilistic disambiguation, and the transformation-based error-driven learning technique.

The basic principle of rule-based approaches is that the knowledge base consists of a set of linguistic generalisations, known most commonly as *rules* or *constraints*. Each rule contains instructions for an operation to be performed, and a context describing where that rule should be applied. The operation alters the list of tags associated with an ambiguously-tagged word in such a way that one or more potential tags are eliminated from consideration, reducing the ambiguity. For instance, going back to the example of the verb/noun *can* discussed above, a rule for an English tagger might state that the modal verb tag should be removed *if* the preceding word is an article, and the noun tag removed *if* the preceding word is a pronoun. This rule takes advantage of the likely local indicators of a noun as opposed to an operator verb in English.

Taking a "rule-based" approach to disambiguation in tagging does not imply using grammar rules as traditionally formulated by linguists. Disambiguation, rule-based or otherwise, typically makes use of short-range, surface-level information: normally no more than the form of preceding or following words, or the tags that these words have. Rarely is much use made of more abstract syntactic concepts such

as the noun phrase, the verbal group, or the clause[4]. As Brill (1995: 544), among others, has pointed out, restricting disambiguation to this minimal information can be highly effective. But such "rules" are a far cry from the theoretical model proposed by many researchers into syntax (compare for instance the clause/phrase structures and processes described by Chomsky[5]).

The earliest approaches to tagging were rule-based (Klein and Simmons 1963; Greene and Rubin 1971). But the most prominent recent rule-based tagger is the Constraint Grammar (CG) system developed at the University of Helsinki (see Karlsson et al. 1995). This represents the state of the art in rule-based disambiguation. Within CG, disambiguation is performed by the application of *constraints*. These are characterised by Voutilainen (1999b) as rules that "perform operation X on target Y in context Z". The operation may be SELECT (delete all analyses but one, which is taken to be correct) or REMOVE (an incorrect analysis). Contexts can refer to words or analyses on words preceding or following the target word by any distance, or to clause boundaries. The constraints in CG are manually written, via a combination of intuition and trial-and-error experimentation (Voutilainen 1999b: 226). While a majority of the published literature concerns the application of the CG methodology to the tagging of English, the approach has been applied to several other languages, for instance French (Chanod and Tapanainen 1995a, 1995b).

The basic principle of probabilistic approaches is that statistical information concerning the frequency with which sequences of tags occur is gathered from long stretches of running text. This data is used to deduce which of the optional analyses of an ambiguously tagged word is the more likely to be correct. Usually, a mathematical model known as a *Markov model* is used to take account of as many adjacent tags as possible in calculating the most likely tag. The mathematics of Markov models are discussed in some detail by Charniak et al. (1993). The most immediate advantage of a probabilistic system over rule-based systems is that the linguist does not have to write an effective set of rules to produce an effective system. The process of *training* a Markov model for POS tagging consists of estimating the parameters of the model. These parameters are the *tag transition probabilities*, which are based on counts of how frequently each tag in the tagset is followed by each other tag in the tagset. These counts are derived from a tagged corpus. Disambiguation using a Markov model consists of applying those probabilities to the task of choosing a single tag from a set of potential tags: the transition probabilities for different potential sequences are multiplied together to identify the most likely sequence of tags. Probabilistic taggers using Markov models have been very successful, for instance the CLAWS tagger (see Garside, Leech and Sampson 1997) and the Xerox tagger (Cutting et al. 1992).

The third common disambiguation technique was developed by Eric Brill in the 1990s and is referred to by him as *transformation-based error-driven learning* (Brill 1995). Transformations, like constraints, are a type of rule; but whereas constraints specify what analyses should be removed from a list of possibilities, transformations change one analysis into another to reduce the number of tagging errors in an unambiguously tagged text. For this reason, it is strictly speaking not a disambiguation technique, but what will be referred to here as an "improvement" technique. Rather than being manually written, in Brill's approach transformations are automatically learnt by an algorithm running on a tagged corpus. Brill argues that this is preferable to training a Markov model on tagged data because whereas the

[4] An exception here is Constraint Grammar (Karlsson et al. 1995), discussed below, which makes use of clause boundaries.

[5] See for example Chomsky (1957).

transition probabilities that probabilistic systems store are nothing like linguistic knowledge as linguists formulate it, the transformations learned in Brill's system benefit from "the perspicuity of a small set of meaningful rules" (Brill 1992: 152).

It can be difficult to compare the success rates of these different tagging methodologies, as differences in the success rates of taggers based on each method may be due to other factors (e.g. the nature of the text, the tagset, the performance of the analyser that provides the candidate tags, and so on; see also Abney 1997). There have been claims that rule-based tagging can outperform probabilistic disambiguation (see, for instance, Chanod and Tapanainen 1995a). But the difference, if it exists, is small, so there are few absolute grounds for preferring one methodology over another for constructing the Urdu tagger described here. However, in practice the relative unavailability of tagged Urdu data effectively rules out both the disambiguation methodologies which require training on pre-tagged data. Only 50,000 words of manually tagged text was available for this study. The methods in question– those based on Markov models and on transformation-based error-driven learning – are usually trained on much larger quantities of data than this. Many studies on English have used the tagged Brown Corpus (1 million words) as training and test data[6], and quantities of text on this scale may be considered typical.

It must be assumed that without the quantities of training data available to the researchers who have produced impressive results with Markov models, those results could not be replicated by an Urdu tagger. The same applies to a transformation-based tagger. Some Markov model taggers can be trained, albeit less effectively, on untagged data (see Merialdo 1994). However, a lexicon containing the possible tags for at least a significant proportion of the words is still necessary for the analysis stage. In the case of Urdu, no such lexicon exists. As lexicons are typically derived from tagged corpora, the need for tagged data is not obviated in this way.

By comparison, when disambiguation is performed using rules created by a linguist, this restriction does not apply. There is no need for a vast quantity of tagged data. Some tagged text is still required, as a benchmark is required against which to test the system, but a much smaller amount will suffice here. On this basis, it is possible to rule out the use of a Markov model or a transformation-based tagging system for the disambiguation module of an Urdu tagger. The only remaining option is disambiguation based on hand-crafted rules.

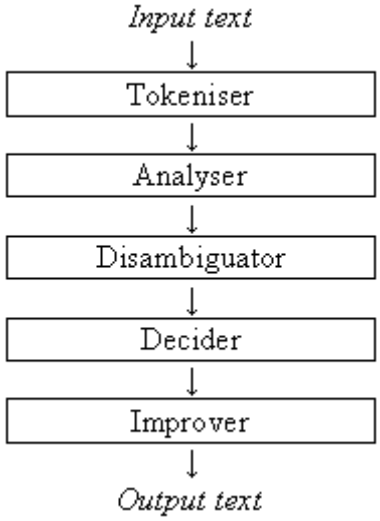
4. Software design and implementation: Unitag

The tagger for Urdu was implemented within a new tagging system called *Unitag*, developed by the author for the purpose of processing two-byte Unicode text (whereas most earlier taggers can process only ASCII text). Unitag is an architecture in which different programs can be assembled to create a tagging system. It consists of a functional typology of disambiguation systems (for not all perform the same task), a structure into which the programs fit, a formalism for these programs to communicate with one another, and a system for declaring which programs are to be used at each stage. The rationale for this is that, as stated above, it is common to divide tagging into the stage of tokenisation, initial tag analysis, and tag disambiguation. This computational task is more tractable to the programmer if handled by several different programs, each called in turn by Unitag. However,

[6] For example, Charniak et al. (1993), Kupiec (1992), Church (1988), Brill and Pop (1999), and Cutting et al. (1992) all utilise the Brown Corpus, or a large percentage of it, as their training data.

handling each of these tasks by means of a separate program also allows them to be used independently where this is of benefit. It should be noted that, although it was written to form the basis of the Urdu tagging system, Unitag itself is entirely language-independent.

The typology of disambiguation systems devised for Unitag is based on the type of input and output a given system has. A *disambiguator* receives ambiguous input and removes some of the ambiguity, without necessarily selecting a single tag (e.g. the CG tagging system). A *decider* receives ambiguous input and selects a single tag per token (e.g. most Markov model taggers). An *improver* receives unambiguously tagged input and alters the tagging in some way to reduce the number of errors (e.g. the basic form of Brill's system). These three types are not incompatible. As Chanod and Tapanainen (1995a) demonstrate, different types of disambiguation system can be linked in serial; hybrid taggers such as CLAWS typically do this (Garside, Leech and Sampson 1987). Unitag can therefore optionally call all three types, as shown below:



Whilst the tokeniser and analyser are compulsory, all the other units are optional. In the Urdu tagger, only a disambiguator is used, because, as has been explained, a rule-based procedure is to be used. However, it is the facility of future work on tagging Urdu and related languages will be greatly enhanced by creating the Urdu tagger in a framework which is ultimately very open to having new components added to it. A parameter file is used to specify the modules to be used, and any special options for their configuration, in an instantiation of Unitag.

The Unitag file format is the formalism used within Unitag for the various programs to communicate with one another. Information about each token is stored in a particular layout, which each program can then interpret. The layout is vertical, i.e. each token is put on a separate line, as in the example lines below:

```
s00004 w001 <body>                    *VE NULL
s00010 w001 چاہیں                     *VE VVSV2 VC2
s00010 w001 چاہیں                     *LE VVSV2/50 VC2/50
```

At the start of the line are two serial numbers which identify the token for an analyst (though they need not be unique), followed by a space. Then, there is a single

horizontal tab character. There follows a three-character code for "last modified by": this allows errors to be tracked to the actions of specific modules.

After the responsibility marker is another space, and then the tag or tags which are currently assigned to that token. If the "token" is actually an SGML tag, it receives the special NULL tag. The tags must consist of characters other than control characters, white space, the underscore character or the forward slash character. Otherwise any Unicode characters may be used. Between each tag is a space. Tags may be followed by a forward slash and a two-digit number indicating how likely that token is to have that tag (a probability expressed as a percentage). Probabilities are optional: a tagged file need not contain any, and if it does then not every line nor every tag on a given line need have them. Where no probability is given for a tag, it is assumed to have an equal share of the probability remaining on that line. So for example, if two tags are given on the line and neither has a stated probability, then both are assumed to have a probability of 50%.

Unitag modules must comply to certain rules to ensure they fit within the system: i.e. they must handle text in the Unitag file format, they must accept command-line arguments in a certain format, they must handle one text file at a time. Several different modules, including tokenisers, analysers, disambiguators and a probabilistic decider, have been developed for Unitag, based on various algorithms described in the literature. However, only the key module in the Urdu tagger, namely the rule-base disambiguator, will be discussed here. This program is called *Unirule*.

Any rule-based disambiguator requires some kind of formalism in which its rules can be expressed. This formalism will necessarily restrict the type of rules that the system can implement. The formalism devised for Unirule was based on a brief survey of best practice as exemplified by the rule formalisms of Constraint Grammar and Brill's transformation-based tagger (both discussed in the previous section). A number of features are shared by the formalisms of these two taggers, which are otherwise quite distinct in their methodology. Both constraints and transformations consist of some operation to be carried out on the analyses given to a token *if* a given set of conditions is fulfilled. One of these conditions refers to the analysis to be changed. The other conditions may refer to analyses on nearby tokens (ambiguously or unambiguously), or to the wordforms of nearby tokens, or (in the case of Constraint Grammar) to the lemma of a nearby token. There is provision for multiple conditions linked by logical OR and logical AND. The operations that may be performed include imposing a specified analysis, selecting a specified analysis from those currently given to the token in question, or deleting a specified analysis from those currently given to the token in question. Although to actually use either the Constraint Grammar formalism or Brill's formalism for Unirule would have been computationally unfeasible, these shared features were used as the guiding principles of the Unirule formalism.

In Unirule, a rule consists of *conditions* and an *action*. The action is the operation which is performed on the "current" token[7] if the conditions are fulfilled. There are four possible actions, *assign*, *select*, *delete* and *deletenot*. An action is specified as follows in the Unirule formalism:

```
a select NNMM1N
```

[7] That is, the token that Unirule is currently analysing in the course of sequentially examining every token in the file.

The initial "a" indicates that this is an action, the following word specifies the type of action that is required, and the tag given at the end is the argument of the action. The possible actions are:

assign	The token is assigned the tag as given. All other tags are deleted.
select	If one of the tags listed for that token matches the tag as given, then all other tags are deleted. If more than one tag listed for that token matches the tag as given, then the first such tag in the list is selected, and all other tags deleted. If no tags match the tag as given, then no action is taken.
delete	Any tag matching the tag as given is deleted, unless it is the only tag remaining on the list.
deletenot	Any tag *not* matching the tag as given is deleted, unless it is the only tag remaining on the list.

Unirule recognises two wildcard characters. The asterisk (*) can represent any single character. For instance, NNM*1N can be used to represent either NNMM1N or NNMF1N, (M and F being the only two characters that can appear in this context in the Urdu tagset). The second wildcard character is #. This may only appear at the end of a string and represents *any zero or more characters up to the next white-space character*. This feature has been particularly designed to take advantage of hierarchical tagsets[8], so N# can refer to all nouns, NN*M# to all masculine nouns, J# to all adjectives, JD# to all determiner-like adjectives, and so on.

The conditions state what features the context must have for the action to be performed on the current token. A condition consists of an instruction on what type of comparison to carry out, the range of the comparison, and what is being looked for. For example:

```
c ifnextwordis 1 be
```

The "c" specifies that this is a condition. The following word specifies the comparison type; it is followed by an integer indicating the range (up to 25 in the current version of Unirule), and then by the string which is the basis for comparison. This example condition will be fulfilled if the first word after the current token is the same as the defined string (in this case, the English word "be").

There are 9 basic comparison types, as follows:

ifthiswordis	Fulfilled if the word-form of **this** token[9] matches
ifthistagis	Fulfilled if all the tags on **this** token match
ifthistaginc	Fulfilled if at least one of the tags on **this** token matches
ifprevwordis x	Fulfilled if the word-form of the x^{th} token **before** this token matches
ifprevtagis x	Fulfilled if all the tags on the x^{th} token **before** this token match
ifprevtaginc x	Fulfilled if at least one of the tags on the x^{th} token **before** this token matches
ifnextwordis x	Fulfilled if the word-form of the x^{th} token **after** this token matches
ifnexttagis x	Fulfilled if all the tags on the x^{th} token **after** this token match

[8] Note, however, that Unirule is still entirely compatible with non-hierarchical tagsets.
[9] That is, the token that Unirule is currently disambiguating.

ifnexttaginc x Fulfilled if at least one of the tags on the x^{th} token **after** this token matches

There is also a logical negative for each of these condition types, i.e. *ifthiswordisnot*, *ifprevtagisnot*, *ifnexttagincnot*, and so on. The terms "previous" and "next" have been used rather than "left" and "right", to avoid terminology rooted in any particular writing system which is not universally applicable. The conditions in Unirule are rather wordier than the conditions in – say – Constraint Grammar. However, this makes them more perspicuous to the untrained reader.

A single rule consists of an action, and all its accompanying conditions, which are listed directly before the action (and after the immediately preceding action). If an action has no conditions, it will be triggered on every token. If the action has more than one condition, then all the conditions must be fulfilled for the action to take effect (logical AND). In the interests of simplicity there is no logical OR, though this effect could easily be achieved by adding another rule with the same action and a different condition.

A Unirule rule file is a single Unicode text file consisting of a string of conditions and actions. Comment lines may be included (beginning in /) and empty lines are passed over. There follows an example of a very short set of rules (for Urdu):

```
/ postpositions follow (pro)nouns, therefore delete verb, adjective
/ and adverb tags
c ifnexttagis 1 II#
a delete V#
c ifnexttagis 1 II#
a delete J#
c ifnexttagis 1 II#
a delete R#
```

When Unirule runs, it loads in the rules from file at time of running. The entire set of rules is applied, in the order they appear in the rule file, to each token in the text file in turn. Unirule can be instructed to make a multiple passes of a single file, to allow rules to take advantage of the unambiguous context that the application of other rules has provided.

5. Urdu language resources

The Urdu tagger described in this paper uses Unirule as its disambiguator. However, a language-specific resource, namely a rule-list, is needed for Unirule to work. Other language-specific resources are required for the Urdu tagger, namely a tagset, a lexicon and an analyser for Urdu text. The creation of these resources is discussed below.

5.1. Tagset

A scheme for categorising the wordforms of a language into approporiate morphosyntactic categories is a *sine qua non* of automated POS tagging; developing a tagset is usually the first step in creating a tagger for a previously untagged language. The creation of a tagset for Urdu has been discussed extensively elsewhere (Hardie 2003, 2004) and will not be discussed here[10] except to note that it was based on the

[10] All the tags mentioned in this article are listed in the Appendix.

Urdu grammar of Schmidt (1999) and the EAGLES international standard for POS tagsets (see Leech and Wilson 1999).

5.2. Analyser

The program used to assign an initial set of contextually ambiguous tags to each token in an Urdu text is a language-specific tool called *Urdutag*. It uses several means of analysis, including lexical lookup, character type analysis, and morphological analysis.

Lexical lookup in Urdutag is fairly basic. A lexicon (discussed below) is loaded and held in memory by the analyser. The first step in the analysis of a given token is to look its wordform up on the lexicon list[11]. If it is found, all the tags given to that wordform in the lexicon are assigned to the token in the text. No attempt is made to lemmatise wordforms.

If no tags are assigned from the lexicon, Urdutag attempts to allocate a tag by analysing the characters that make up the token. This algorithm is currently very primitive, detecting only the JDNU and FX categories. If all the characters in the token are numerals (Arabic or Latin), the tag JDNU is assigned. If the token contains characters from outside the Arabic alphabet, the tag FX is assigned. If no tag has been assigned at this point, the word is morphologically analysed[12]. It is in this area that a great deal of specific linguistic knowledge has been built in to the Urdutag program.

The Urdu grammar of Schmidt (1999) was used as a model of the language to develop an algorithm for morphological analysis of Urdu tokens. Urdutag reads characters from the word, one by one, starting from the end, and matches the longest suffix it can. When it has matched a suffix, it assigns a set of tags associated with that suffix (or a default set of tags if no suffix is matched). These sets can be quite large, as there is a great deal of ambiguity among suffixes in use in Urdu. This means that inevitably, the ambiguity inherent in the wordform is reproduced in the output of the analyser. The suffixes that the program identifies, and the tags those suffixes indicate, are listed below. The suffixes are transliterated in a non-standard way to indicate more clearly the actual letters that occur in the Indo-Perso-Arabic text; the corresponding standard transliterations may be inferred from the examp[le words. *Chōtī yē*, ی, which represents variously *y*, *ī*, and *ē* and *ai*, is shown as Y; *vāō*, و, pronounced *v*, *ū*, *ō*, or *au*, is shown as V; *baRī yē*, ے, is indicated by E, and *chōTī yē* with superscript *hamza*, ۓ, is shown as [hoy] (for "hamza over yē").

Suffix + example[13]	**Tag indicated**
Noun suffixes	
ā, *laRkā*	NNMM1N
h, *baccah*	NNMM1N
Yh, *rūpayah*	NNMM1N
gāh, *ibādatgāh*	NNUF1N NNUF1O NNUF1V
stān, *inglistān*	NNUM1N NNUM1O NNUM1V NNUM2N
pn, *bacpan*	NNUM1N NNUM1O NNUM1V NNUM2N

[11] Vowel diacritics are stripped from the word prior to the process of analysis, so a single lexicon can be used for both text without vowel marks and the much rarer vowelled texts.

[12] The morphological analysis stage also splits off clitics as separate tokens. The clitics identified are *al–*, *-gunā*, and the various clitic forms of *hī* and *kō*.

[13] Example words are from Schmidt (1999) and Bhatia and Koul (2000).

pā, *buRhāpā*	NNMM1N
Y, *laRkī*	NNMF1N NNMF1O NNMF1V
Yā, *ciriyā*	NNMF1N NNMF1O NNMF1V
āhT, *ghabrāhaT*	NNUF1N NNUF1O NNUF1V
āVT, *rukāvaT*	NNUF1N NNUF1O NNUF1V
Yt, *insāniyat*	NNUF1N NNUF1O NNUF1V
E, *laRkē*	NNMM1O NNMM1V NNMM2N
[hoy]E, *rūpae*	NNMM1O NNMM1V NNMM2N
Yā~, *laRkiyā~*	NNMF2N
YV~, *laRkiyō~*	NNMF2O
Y~, *kitābe~*	NNUF2N
V~, *laRkō~*	NNMM2O NNUM2O NNUF2O
V, *laRkō*	NNMM2V NNMF2V NNUM2V NNUF2V
Adjective suffixes	
ā, *chōTā*	JJM1N
Y, *chōTī*	JJF1N JJF1O JJF2N JJF2O
E, *chōTē*	JJM1O JJM2N JJM2O RRJ
Vā~	JDNM1N
VY~	JDNM1O JDNM2N JDNM2O JDNF1N JDNF1O JDNF2N JDNF2O
Adverb suffixes	
[Unicode 064B], *faurān*	RR (*Arabic loans ending in the "tanvīn" character*)
Verb suffixes	
nā, *likhnā*	VVNM1N
nE, *likhnē*	VVNM1O VVNM2
nY, *likhnī*	VVNF1 VVNF2
tā, *likhtā*	VVTM1N
tE, *likhtē*	VVTM1O VVTM2N VVTM2O
tY, *likhtī*	VVTF1N VVTF1O VVTF2N VVTF2O
tY~, *likhtī~*	VVTF2N
ā, *likhā*	VVYM1N
Yā, *likhā*	VVYM1N
E, *likhē*	VVYM1O VVYM2N VVYM2O
Y, *likhī*	VVYF1N VVYF1O VVYF2N VVYF2O
Y~, *likhī~*	VVYF2N
V~, *likhū~*	VVSM1
E, *likhē*	VVST1 VVSV1
Y~, *likhē~*	VVSM2 VVSV2
V, *likhō*	VVST2 VVIT2
[hoy]E, *likhiē*	VVIA
[hoy]YE, *jāiyē*	VVIA

Some endings (especially *Y*, *E*, *ā, Y~* and *V*) are highly polysemous. This means that the ambiguity in the analysis is quite high. For instance, the tags for *Y* are NNMF1N NNMF1O NNMF1V JJF1N JJF1O JJF2N JJF2O VVYF1N VVYF1O VVYF2N VVYF2O. Urdutag always matches the longest suffix it can; therefore, if a word ends in *pā*, Urdutag will assign the set of tags for *pā*, *not* the set of tags for *ā*.

Exceptions to the generalisations in the table above must be stored in the lexicon, so that they are not incorrectly tagged by the morphological analyser.

If no tag is assigned at this stage, a default set of tags, listed below, is assigned to the token:

```
JJU, NNUM1N, NNUM1O, NNUM1V, NNUM2N, NNUF1N, NNUF1O, NNUF1V, RR,
VVIT1, VV0
```

5.3. Lexicon

Urdutag is designed to use lexicons in a format similar to the Unitag file format. A line of a lexicon file looks like this:

```
i000365 کرنی                    VVNF1 VVNF2
i000365 کرنی                    VVNF1/50 VVNF2/50
```

The serial number begins with "i" (for "item") and is followed by 6 figures (to distinguish lexicons from tagged files). This is followed by a space, and then the wordform. There is then a horizontal tab character, followed by a list of all possible tags for that wordform separated by spaces, followed by a carriage return. As with the Unitag format, the lexicon format may contain probabilities.

The simplest way to create a lexicon is automatically, by acquiring a list of wordforms and possible tags from pre-tagged data. The *Unilex* program, an adjunct to Unitag, was created to do this. However, the lexicon actually used by Urdutag is a combination, compiled using Unilex, of two separate lexicons. The first is the lexicon acquired automatically. The other was written manually and contains words which are exceptions to the morphological principles used by Urdutag: for example, the adjective *purānā*, "old", which has a formal resemblance to an infinitive verb, as well as all the wordforms that fall into closed categories.

A key parameter for the function of Unilex which acquires lexicons from text files is the *frequency threshold*. This is an integer, selected by the user, which sets a minimum number of occurrences below which a word will not be included in the lexicon. So if the threshold is set at 3, and the word *kitāb* occurs twice in the training data, it will not be included in the lexicon.

The advantage of a low threshold is clear: the lexicon will provide a tag for more wordforms. But a low threshold may not always give the best result. It may result in words that occur only once or twice in the training corpus being stored in the lexicon with only some of the tags they can conceivably have. For example, if the word *sunī~/sunē~*[14], "hear (perfective participle/subjunctive)" occurs once in the training text, with the tag VVYF2N, every instance of *sunī~/sunē~* will be given the tag VVYF2N. But *sunī~/sunē~* may also be VVSV2 or VVSM2. In this case, Urdutag would have achieved better accuracy by letting the suffix analyser work on the form of the word, to produce the following list of tags: NNUF2N, VVSM2, VVSV2, VVYF2N. This list includes all the correct readings (plus an extraneous noun tag). A higher threshold can prevent low-frequency wordforms blocking the operation of the morphological analyser in this way.

To discover the effect of the lexicon threshold, Urdutag was run on the training data and two test texts using 11 different lexicons, created using automatic

[14] These two words (though not homophonous) are identical in Indo-Perso-Arabic script.

lexicons with thresholds of 1 to 10 and, in addition, the manual lexicon on its own. Fig. 1 shows the resulting accuracy rates.

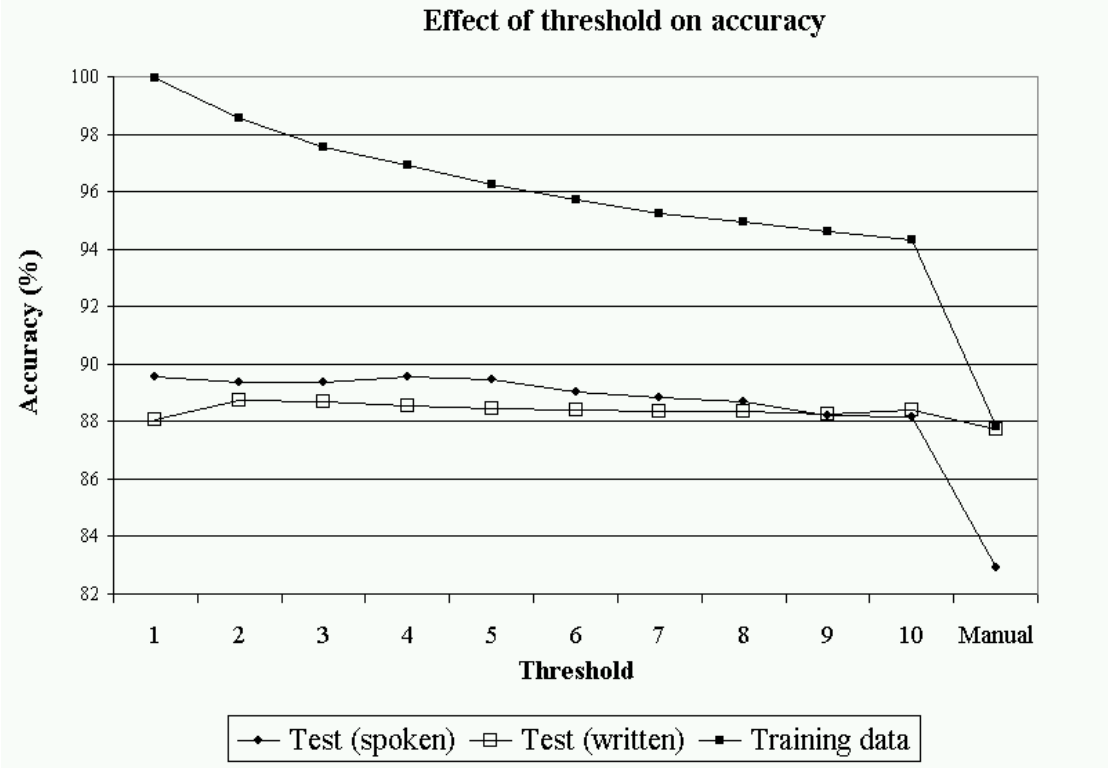


Fig. 1

Apart from the clear indication that a lexicon derived from the data being tagged is very useful (thus the much better performance achieved on the training data), several points arise from these results. The manual lexicon alone produces poor results. A threshold 1 lexicon produces excellent results for the training data from which it was derived, but on other data does not necessarily perform better than a lexicon with a higher threshold. The threshold 1 and threshold 4 lexicons do equally well on the spoken test data, and a threshold 2 lexicon does best on the written data.

It is clear from this that the "blocking" effect described above is real, and problematic for tagging, the experiment does not give sufficient evidence to select an optimum threshold, since the two test texts behaved somewhat differently. Therefore, another approach was used to get around the blocking problem. Instead of removing the offending entries from the lexicon by raising the threshold, it seemed logical to attempt to enrich the lexicon in an attempt to add to those entries the tags that they were missing.

A dedicated program, *Growurdulex*, has been used to enrich automatically acquired lexicons. It does this by adding tags to the entries in the lexicon on the basis of the tags that are already there. Groups of tags are designated that apply to forms which are morphologically identical. If an entry in the lexicon has any member of a group, then the other members are added if not already there. An example of a group is JJM1O~JJM2N~JJM2O, all of which apply to words with the suffix the suffix *–ē*.

When the experiment described above was redone with enriched lexicons, the results were as shown in Fig. 2.

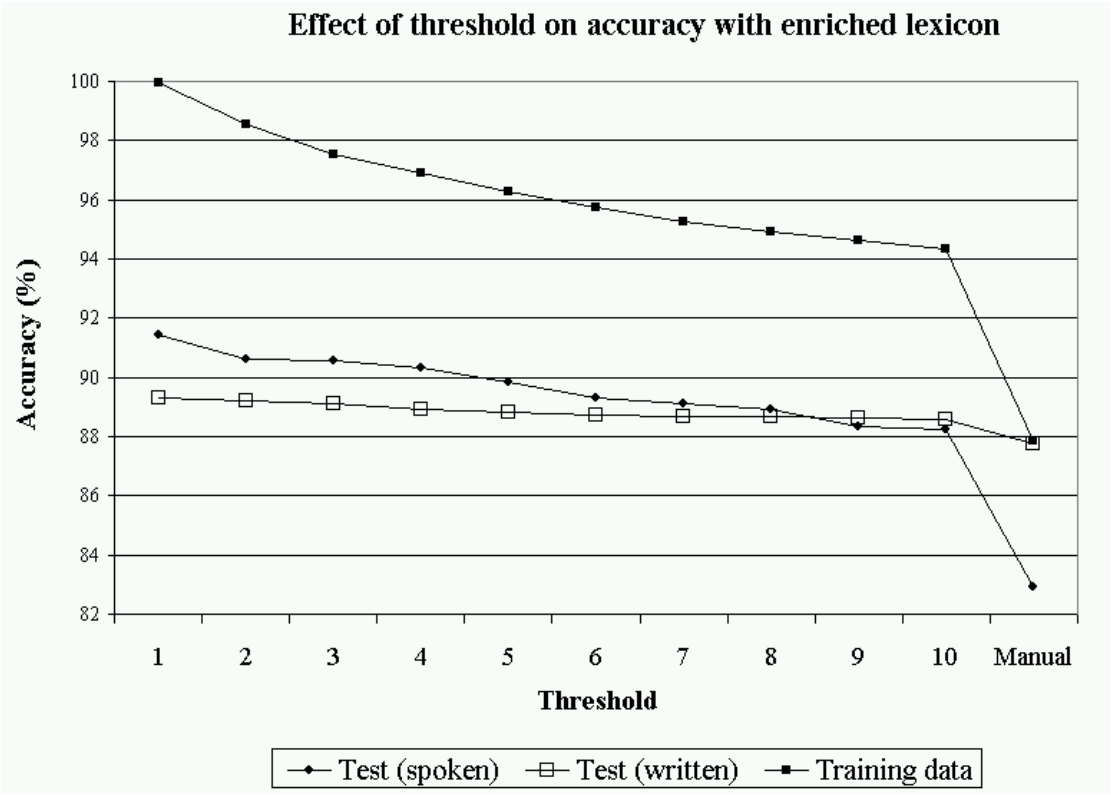


Fig. 2

As can be seen, the blocking effect vanishes, so the threshold 1 lexicon performs best for all three datasets. There is also a noticeable gain in overall best accuracy: 1.9% for the spoken text, 0.6% for the written text. Moreover this comes at a relatively small cost in extra ambiguity. Therefore, the capacity to enrich a lexicon resolves the problem of finding an optimal threshold: it is 1.

There is evidence to suggest that the type of the text from which the lexicon is derived is critical. The spoken test text was at all points easier for Urdutag to handle than the written text, since the training data is also spoken Urdu.

5.4. Rules

When writing disambiguation rules, the analyst may use their knowledge of the language in question to find generalisations about sequences of parts-of-speech which they anticipate will reduce ambiguity. Rules are then devised to encode these generalisations. Alternatively, they may come up with rules by examining the errors and ambiguity in the data.

The rules that underlie the Urdu tagger described here were initially based on the model of Urdu grammar presented by Schmidt (1999). This allowed work to be done on the rule list before the training data became available. However, these rules were all corrected, and further rules developed, by examining output from Unirule for errors and remaining ambiguity.

In developing a rule, it is necessary to move from a generalisation about the surface structure of the language to one, or in most cases more than one, rules in the Unirule formalism. The first 105 rules were based on the following generalisations extracted from Schmidt (1999):

- Postpositions follow nouns and verb infinitives (4)[15]
- Nouns and pronouns take the oblique case before postpositions (6)
- The particle *hī* is clitic after the oblique case of plural personal pronouns, but not the nominative (2)
- In the phrase *āp kā / kē / kī*, *āp* is reflexive, not honorific (1)
- In a compound postposition consisting of *kē* plus adverb, *kē* is IIM1O (1)
- Words with adjective-like inflection (i.e. marked adjectives, the marked postposition *kā*, marked determiner-like adjectives such as ordinal numerals, the adjectival particle *sā* , and possessive pronouns) agree with a following noun for case, gender and number (78)
- The auxiliary *rahā* is preceded by a verb in the root form (1)
- General auxiliary verbs and the verbal postposition *kē* always follow a verb in the root form (3)
- The future auxiliary follows a subjunctive verb (1)
- An infinitive before *cāhiē~* (VC2) is not singular (1)
- *kyā* at the start of a clause is a question marker rather than an interrogative pronoun[16] (2)
- The principle that Unirule should always delete an F* tag, especially FU, if another analysis was available was also adopted (5)

As an example of how these principles translate into rules, let us take the principle that nouns and pronoun are oblique before postpositions. This rule could be stated as "if the next word is unambiguously tagged as a postposition, remove any tags indicating a nominative or vocative noun". This translates into the Unirule formalism as:

```
c ifnexttagis 1 II#
a delete N****N
c ifnexttagis 1 II#
a delete N****V
```

This only covers nouns. Pronouns (whose tags begin with P) are more complicated. Personal pronouns are nominative, not oblique, before the postposition *nē*. Furthermore, pronoun tags vary in length, so the same wildcard template will not fit all pronoun tags, as is the case for nouns. The rules for pronouns are as follows:

```
c ifnexttagis 1 II#
c ifnextwordisnot 1 نے
a delete PP**N
c ifnexttagis 1 II
c ifnextwordis 1 نے
a delete PP**O
c ifnexttagis 1 II#
a delete P**N
c ifnexttagis 1 II#
a delete PNN
```

These 105 rules were applied to data analysed using Urdutag and the optimal lexicon (see above). Before the application of the rules, accuracy was 100% and ambiguity 2.89; afterwards, they were 97.8% and 2.55, respectively. This

[15] Numbers in brackets indicate the number of rules related to a given principle.
[16] This principle turned out to be unreliable in practice and was removed at a later stage.

performance was then improved by developing additional rules using the training data, as described below, to reach 99.0% and 1.73. Ultimately a total of 274 rules were written. Some of these disambiguate only one or two tokens each; others disambiguate hundreds of tokens.

6. Evaluation: the success rate of the Urdu tagger

When all components of the system described above are run on the training dataset, the result is 99.0% accuracy with an ambiguity of 1.73 tags per word. However, running on test texts which were not part of the data that the system was trained on, the same system achieves 90.6% / 2.20 (spoken text) and 88.1% / 2.97 (written text). It seems clear that the Urdu tagger does not match up to the mainstream of taggers for languages such as English. Markov model taggers for English regularly score above 95% accuracy with ambiguity 1, for instance.

It may be asked why this should be. The primary cause of the tagger's poor performance is an inadequate lexicon. Running on its training data, where it benefits from a lexicon containing all the words, it performs well. On the test data, where many tokens are not in the lexicon, accuracy drops by 9-10%, and ambiguity increases drastically. This suggests that the common core of Urdu vocabulary which needs to be in the lexicon for the tagger to cope with unseen text has *not* been captured by deriving a lexicon from the training data. This is also suggested by the size of the lexicon – circa 3,900 items. As a point of comparison, the English handcrafted tagging lexicons used by the CLAWS tagger (Smith 1997: 141-144) contains 15,000-23,000 items, and some automatically derived lexicons rise to 45,000 items. So the small size of the lexicon, a result of the scarcity of training data, hamstrings the Urdu tagger from the outset.

However, it seems clear that given an appropriate lexicon, the disambiguation rules devised for Urdu do work well. On the training data, they reduce ambiguity from 2.55 to 1.73 tags per word (removing over half the ambiguity in the initial analysis) at a cost of only 1.0% accuracy. It is on the test data, where due to the lexicon the analysis is poor to begin with, that the disambiguation rules cause an unfortunately large number of errors (decreasing accuracy from 89.9% to 88.1% on the written test data, and from 92.5% to 90.6% on the spoken test data).

It would therefore seem that of the resources created during this study, the Urdu lexicon is the weakest and least adequate. Unfortunately, this leads to a comparable inadequacy in the tagger as a whole. However, the software tools that created the lexicon would provide a means to acquire a far superior lexicon, if only adequate training data were available. But to create such data would be a time-intensive and expensive procedure.

While the disambiguation rules were more successful, they left many tokens with two or more tags. This remaining ambiguity was very difficult to remove without causing large numbers of errors. Some of this was down to categories which have identical forms – such as the various tags for feminine adjectives. JJF1N, JJF1O, JJF2N and JJF2O are impossible to tell apart from their form. Another problem was disambiguating words that could be adjectives or adverbs (i.e. JJU/RR or JJM1O/RRJ), with little in the immediate context to indicate the difference. There were also individual problematic words. For instance سو (*sau*, "seven", or *sō*, a multiple homonym) was extremely difficult to disambiguate. Virtually all instances of

this word retain the tags JDNU[17], CC and RR. It is possible that additional work on rule-writing might clear some of the remaining ambiguity. Alternatively, reformulating the tagset into a scheme that the computer is more likely to succeed in annotating, without losing the key features that would be required for analysis of Urdu texts, is another potential path to improvements.

7. Conclusion: lessons for the future

The research reported here has successfully demonstrated that automated part-of-speech tagging of Urdu text is possible using pre-existing knowledge, techniques and standards – in particular the rule-based disambiguation methodology. Nevertheless, as pointed out above, there are flaws and room for improvement in the resulting tagger. While comparison between different tagging systems is very difficult, it is nonetheless very clear that the Urdu tagger described here does not approach the levels of accuracy and ambiguity that have been achieved for languages like English. It is not to be expected that a single small-scale project such as this could match the result of at least two decades' intensive research. But it *has* been possible to create a working system capable of producing a usable output. Furthermore, the experience gained with Urdu provides a good starting point for attempting automated morphosyntactic annotation in other Indo-Aryan languages, which have the same prior requirements as Urdu in terms of Unicode-compliant software frameworks and resource creation.

8. References

Abney, S (1997) Part-of-speech tagging and partial parsing. In: Young, S and Bloothooft, G (eds.) (1997) *Corpus-based methods in language and speech processing*. Dordrecht: Kluwer Academic Publishers.

Bhatia, TK and Koul, A (2000) *Colloquial Urdu*. London: Routledge.

Brill, E (1992) A simple rule-based part of speech tagger. In: *Proceedings of the Third Conference on Applied Natural Language Processing (ANLP'92)*. Trento.

Brill, E (1995) Transformation-based error-driven learning and Natural Language Processing: a case study in part-of-speech tagging. In: *Computational Linguistics*, 21 (4): 543-565.

Brill, E and Pop, M (1999) Unsupervised learning of disambiguation rules for part of speech tagging. In: Armstrong et al. (1999).

Chanod, J-P and Tapanainen, P (1995a) Tagging French – comparing a statistical and a constraint-based method. In: *Proceedings of the Seventh Conference of the European Chapter of the ACL*. Dublin: Association for Computational Linguistics.

Chanod, J-P and Tapanainen, P (1995b) Creating a tagset, lexicon and guesser for a French tagger. In: *Proceedings of the European Chapter of the Association for Computational Linguistics EACL-SIGDAT Workshop*. Dublin.

Charniak, E, Hendrickson, C, Jacobson, N and Perkowitz, M (1993) Equations for part of speech tagging. In: *Proceedings of the Eleventh National Conference on Artificial Intelligence*. Menlo Park: AAAI Press/MIT Press.

Chomsky, N (1957) *Syntactic structures*. Mouton.

[17] Any word homonymous with a numeral is difficult to fully disambiguate since Urdu numerals may occur in a great variety of syntactic settings.

Church, K (1988) A stochastic parts program and noun phrase parser for unrestricted text. In: *Proceedings of the second conference on Applied Natural Language Processing, ACL.*

Cutting, D, Kupiec, J, Pederson, J, and Sibun, P (1992) A practical part-of-speech tagger. In: *Proceedings of the third conference on Applied Natural Language Processing*, ACL.

Garside, R, Leech, G and Sampson, G (eds.) (1987) *The computational analysis of English*. London: Longman.

Greene, BB and Rubin, GM (1971) *Automatic grammatical tagging of English*. Providence, Rhode Island: Brown University Department of Linguistics.

van Halteren, H (ed.) (1999) *Syntactic wordclass tagging*. Dordrecht: Kluwer Academic Publishers.

van Halteren, H and Voutilainen, A (1999) Automatic taggers: an introduction. In: van Halteren (1999).

Hardie, A (2004) *The computational analysis of morphosyntactic categories in Urdu*. PhD thesis, University of Lancaster.

Hardie, A (2003) Developing a tagset for automated part-of-speech tagging in Urdu. In: Archer, D, Rayson, P, Wilson, A, and McEnery, T (eds.) (2003) *Proceedings of the Corpus Linguistics 2003 conference. UCREL Technical Papers Volume 16*. Department of Linguistics, Lancaster University.

Karlsson, F, Voutilainen, A, Heikkilä, J and Anttila, A (eds.) (1995) *Constraint Grammar: a language-independent system for parsing unrestricted text*. Berlin: Mouton de Gruyter.

Klein, S and Simmons, RF (1963) A computational approach to grammatical coding of English words. In: *Journal of the Association for Computing Machinery*, 10: 334-347.

Kupiec, J 1992 Robust part of speech tagging using a hidden Markov model. *Journal of Computer Speech and Language*, 6 (3): 225-242.

Leech, G (1987) General introduction. In: Garside, Leech and Sampson (1987).

Leech, G and Smith, N (1999) The use of tagging. In: van Halteren (1999a).

Leech, G and Wilson, A (1999) Standards for tagsets. In: van Halteren (1999a). (Edited version of *EAGLES Recommendations for the Morphosyntactic Annotation of Corpora* (1996): available on the internet at http://www.ilc.cnr.it/EAGLES96/annotate/annotate.html .)

Merialdo, B (1994) Tagging English text with a probabilistic model. In: *Computational Linguistics*, 20 (2): 155-171.

Schmidt, RL (1999) *Urdu: an essential grammar*. London: Routledge.

Voutilainen, A (1999a) Orientation. In: van Halteren (1999a).

Voutilainen, A (1999b) Hand-crafted rules. In: van Halteren (1999a).

Appendix: tags mentioned in the text

Tag	Description
CC	Coordinating conjunction
FX	Non-Perso-Arabic string
II	Unmarked postposition
JDNU	Cardinal number
JDNM1N	Masculine singular nominative ordinal number
JDNM1O	Masculine singular oblique ordinal number
JDNM2N	Masculine plural nominative ordinal number
JDNM2O	Masculine plural oblique ordinal number

JDNF1N	Feminine singular nominative ordinal number
JDNF1O	Feminine singular oblique ordinal number
JDNF2N	Feminine plural nominative ordinal number
JDNF2O	Feminine plural oblique ordinal number
JJM1N	Marked masculine singular nominative adjective
JJM1O	Marked masculine singular oblique adjective
JJM2N	Marked masculine plural nominative adjective
JJM2O	Marked masculine plural oblique adjective
JJF1N	Marked feminine singular nominative adjective
JJF1O	Marked feminine singular oblique adjective
JJF2N	Marked feminine plural nominative adjective
JJF2O	Marked feminine plural oblique adjective
JJU	Unmarked adjective
NNMM1N	Common marked masculine singular nominative noun
NNMM1O	Common marked masculine singular oblique noun
NNMM1V	Common marked masculine singular vocative noun
NNMM2N	Common marked masculine plural nominative noun
NNMM2O	Common marked masculine plural oblique noun
NNMM2V	Common marked masculine plural vocative noun
NNMF1N	Common marked feminine singular nominative noun
NNMF1O	Common marked feminine singular oblique noun
NNMF1V	Common marked feminine singular vocative noun
NNMF2N	Common marked feminine plural nominative noun
NNMF2O	Common marked feminine plural oblique noun
NNMF2V	Common marked feminine plural vocative noun
NNUM1N	Common unmarked masculine singular nominative noun
NNUM1O	Common unmarked masculine singular oblique noun
NNUM1V	Common unmarked masculine singular vocative noun
NNUM2N	Common unmarked masculine plural nominative noun
NNUM2O	Common unmarked masculine plural oblique noun
NNUM2V	Common unmarked masculine plural vocative noun
NNUF1N	Common unmarked feminine singular nominative noun
NNUF1O	Common unmarked feminine singular oblique noun
NNUF1V	Common unmarked feminine singular vocative noun
NNUF2N	Common unmarked feminine plural nominative noun
NNUF2O	Common unmarked feminine plural oblique noun
NNUF2V	Common unmarked feminine plural vocative noun
PPM1N	First person singular nominative personal pronoun (*mai~*)
PPM1O	First person singular oblique personal pronoun (*mujh*)
PY1N	Singular nominative proximal demonstrative pronoun (*yah*)
PNN	Nominative indefinite pronoun (*kōī, kuch, sab*)
RR	General adverb
RRJ	General adverb derived from adjective
VV0	Root form lexical verb
VVNM1N	Infinitive lexical verb, masculine singular nominative
VVNM1O	Infinitive lexical verb, masculine singular oblique
VVNM2	Infinitive lexical verb, masculine plural nominative
VVNF1	Infinitive lexical verb, feminine singular nominative
VVNF2	Infinitive lexical verb, feminine plural nominative
VVTM1N	Masculine singular (nominative) imperfective participle lexical verb
VVTM1O	Masculine singular oblique imperfective participle lexical verb
VVTM2N	Masculine plural (nominative) imperfective participle lexical verb
VVTM2O	Masculine plural oblique imperfective participle lexical verb
VVTF1N	Feminine singular (nominative) imperfective participle lexical verb
VVTF1O	Feminine singular oblique imperfective participle lexical verb
VVTF2N	Feminine plural (nominative) imperfective participle lexical verb
VVTF2O	Feminine plural oblique imperfective participle lexical verb
VVYM1N	Masculine singular (nominative) perfective participle lexical verb

VVYM1O	Masculine singular oblique perfective participle lexical verb
VVYM2N	Masculine plural (nominative) perfective participle lexical verb
VVYM2O	Masculine plural oblique perfective participle lexical verb
VVYF1N	Feminine singular (nominative) perfective participle lexical verb
VVYF1O	Feminine singular oblique perfective participle lexical verb
VVYF2N	Feminine plural (nominative) perfective participle lexical verb
VVYF2O	Feminine plural oblique perfective participle lexical verb
VVSM1	First person singular subjunctive lexical verb
VVSM2	First person plural subjunctive lexical verb
VVST1	Second person singular subjunctive lexical verb
VVST2	Second person plural subjunctive lexical verb
VVSV1	Third person singular subjunctive lexical verb
VVSV2	Third person plural subjunctive lexical verb
VVIT1	Second person singular imperative lexical verb
VVIT2	Second person singular imperative lexical verb
VVIA	Second person honorific imperative lexical verb