CHAPTER 5
PART OF SPEECH TAGGER FOR TAMIL

5.1 GENERAL

The knowledge of the language pair is proved to improve the translation performance. Adding pre-processing in SMT system convert the language pairs into more similar. Philip Koehn and Hoang (2007) [10] developed a Factored translation framework for statistical translation models to tightly integrate linguistic information. It is an extension of phrase-based statistical machine translation that allows the integration of additional morphological and lexical information, such as lemma, word class, gender, number, etc., at the word level on both source and the target languages. Preprocessing methods are used to convert Tamil language sentences into factored Tamil sentences. The preprocessing module for Tamil language sentence includes two stages, which are POS tagging and Morphological analysis. The first step in preprocessing Tamil language sentence is to retrieve the Part-of-Speech information of each and every word. This information is included in the factors of surface word. This chapter explains about the development of Tamil POS tagger system. In the next stage, Tamil morphological analysis is used to retrieve the lemma and morphological information. This information also included in factors of surface word. The next chapter (Chapter-6) explains the implementation details about the Tamil morphological analyzer.

5.1.1 Part of Speech Tagging

Part of Speech (POS) tagging is the process of labeling a Part-of-Speech or other lexical class marker to each and every word in a sentence. It is similar to the process of tokenization for computer languages. Hence POS tagging is considered to be an important process in speech recognition, natural language parsing, morphological parsing, information retrieval and machine translation. Generally, a word in a language contains both grammatical category and grammatical features. Tamil being a morphologically rich language inflects, with more grammatical features which makes the POS tagger system complex. Here a POS tagger system has been developed based only on grammatical categories. Additionally, a morphological analyzer has also been developed for handling grammatical features. Automatic Part-of-Speech tagger can
help in building automatic word-sense disambiguating algorithms. Parts of Speech are very often used for shallow parsing texts, or for finding noun and other phrases for information extraction applications. The corpora that have been marked for Part-of-Speech are very useful for linguistic research, for example, to find instances or frequencies of a particular word or sentence constructions in large corpora.

Apart from these, many Natural Language Processing (NLP) activities such as summarization, document classification and Natural Language Understanding (NLU) and Question Answering (QA) systems are dependent on Part-of-Speech Tagging. Words are divided into different classes called Parts of Speech (POS), word classes, morphological classes, or lexical tags. In traditional grammar, there are only a few parts of speech (noun, verb, adjective, adverb, etc.). Many of the recent models have much larger number of word classes (POS Tags). Part-of-Speech tagging (POS tagging or POST), also called grammatical tagging, is the process of marking up the words in a text as corresponding to a particular Part of Speech, based on both its definition, as well as its context.

Parts-of-speech can be divided into two broad super categories:

- CLOSED CLASS types
- OPEN CLASS types.

Closed classes are those that have relatively a fixed membership. For example, prepositions are a closed class because there is a fixed set of them in English; new prepositions are rarely coined. By contrast, nouns and verbs are open classes because new nouns and verbs are continually coined or borrowed from other languages. There are four major open classes that occur in the languages of the world; nouns, verbs, adjectives, and adverbs. It turns out that Tamil and English have all the four of these, although not every other language does.

Parts of Speech (POS) tagging means assigning grammatical classes i.e. suitable Parts of Speech tags to each word in a natural language sentence. Assigning a POS tag to each word of an un-annotated text by hand is a laborious and time consuming process. This has led to the development of various approaches to automate the POS tagging work. Automatic POS tagger take a sentence as input, assigns a POS tag to each and every word in the sentence, and produces the tagged text as output. Tags are
also applied to punctuation markers; thus tagging for natural language is the same process as tokenization for computer languages. The input to a tagging algorithm is a string of words and a specified tagset. The output is a single best tag for each word. For example in English,

\[
\begin{array}{c|c|c}
\text{Take} & \text{that} & \text{Book.} \\
\text{VB} & \text{DT} & \text{NN} \\
\end{array}
\] (Tagged using Penn Tree Bank Tagset)

Even in this simple sentence, automatically assigning a tag to each word is not trivial. For example, the word \emph{book} is ambiguous. That is, it has more than one possible usage and Part of Speech. It can be a verb (as in, book that bus or to book the suspect) or a noun (as in, hand me that book, or a book of matches). Similarly \emph{that} can be a determiner (as in, Does that flight serve dinner), or a complimentizer (as in, I thought that your flight was earlier).

As Tamil is a morphologically rich language, a word may have many grammatical categories which lead to ambiguity. For example, consider the following sentence and its corresponding POS tags:

\[
\begin{array}{c|c|c|c|c|c|c|c|c|c}
\text{Tamil Example:} & \text{ேகாவி} & \text{அரு} & \text{அதி} & \text{உயரமான} & \text{மணி} & \text{உள்ள} \\
\text{kOvilil} & \text{ARu} & \text{adi} & \text{uyaramAna} & \text{maNi} & \text{uLLathu} \\
\text{NN} & \text{CRD} & \text{NN} & \text{ADJ} & \text{NN} & \text{VF} \\
\end{array}
\] (Tagged using AMRITA Tagset)

Here

“\text{adi}” can be tagged as Noun (NN) or Verb Finite (VF),
“\text{ARu}” can be tagged as Noun (NN) or Cardinal (CRD)
“\text{maNi}” can be tagged as common noun(NN) or as proper noun (NNP).

Considering the syntax or the context in the sentence, the word “\text{adi}” should be tagged as noun (NN). The problem of automatic POS-tagging is to resolve these ambiguities in choosing the proper tag for the context. Part-of-Speech tagging is thus a disambiguation task. Another important point which was discussed and agreed upon
was that POS tagging is NOT a replacement for morph analyzer. A 'word' in a text carries the following linguistic knowledge

- Grammatical category and
- Grammatical features such as gender, number, person etc.

The POS tag should be based on the 'category' of the word and the features can be acquired from the morph analyzer.

5.1.2 Tamil POS Tagging

Words can be classified under various parts of speech classes based on the role they play in the sentence. Traditionally Tamil grammarian Tholkappiar has classified Tamil word categories into four major classes.

- பெயர் peyar (noun)
- வினை vinai (verb)
- இடை idai (part of speech which modifies the relationships between verbs and nouns)
- உரி� uri (word that further qualifies a noun or verb)

Examining the grammatical properties of words in modern Tamil, Thomas Lehman (1983) has proposed eight POS categories [134]. The following are the major POS classes in Tamil.

1. Nouns
2. Verbs
3. Adjectives
4. Adverbs
5. Determiners
6. Post Positions
7. Conjunctions
8. Quantifiers
Other POS categories for Tamil

Apart from nouns and verbs, the other POS categories that are “open class” are the adverbs and adjectives. Most adjectives and adverbs, in their root can be placed in the lexicon. But there are adjectives and adverbs that can be derived from noun and verb stems. Following are the Morphotactics of derived adjectives from noun root and verb stems.

Examples:

Noun_root + adjective_suffix
uyaram + Ana = uyaramAna <ADJ>
உயரம் + ஆன = உயரமான

verb_stem + relative_participle
cey + tha = ceytha <VNAJ>
ெசய் + த = ஒசய்த

Following is the Morphotactics of derived adverbs from noun roots and verb stem.

noun_root + adverb_suffix
uyaram +Aka = uyaramAka <ADV>
உயரம் + ஆக = உயரமாக

verb_stem + adverbial participle
cey + thu = ceythu <VNAV>
ெசய் + ஐ = ஒசய்ஐ

There are number of non-finite verb structure forms in Tamil. Apart from participles forms, Grammatically they are classified into structures such as infinitive, conditional, etc.,

Examples:

verb_stem + Infinitive marker
paRa + kka=paRakka <VINT> (to fly)
பற + க்க = பறக்க
Examples:

*verb_stem + Conditional suffix*

\[ vA + wth + Al = vawthAl \ <CVB> \ (if one comes) \]
\[ வா + ந் +ஆள் = வந்தாள் \]

There are other categories like conjunctions, complementizers etc. Some of these may be derived forms. But there aren’t many. So they can be listed in the lexicon. Other categories that need to be listed in the lexicon as roots are post positions which are “closed class”. This is because they can occur as words in isolation even though they are semantically bonded to the noun or verb preceding them.

5.2 COMPLEXITY IN TAMIL POS TAGGING

As Tamil is an agglutinative language, nouns get inflected for number and cases. Verbs get inflected for various inflections which include tense, person, number, gender suffixes. Verbs are adjectivalized and adverbialized. Also verbs and adjectives are nominalized by means of certain nominalizers. Adjectives and adverbs do not inflect. Many post-positions in Tamil [159] are from nominal and verbal sources. So, many times one has to depend on the syntactic function or context to decide upon whether one is a noun or adjective or adverb or postposition. This leads to the complexity of Tamil in POS tagging.

5.2.1 Root Ambiguity

The root word can be ambiguous. It can have more than one sense, sometimes roots belong to more than one POS category. Though the POS can be disambiguated using contextual information like co-occurring morphemes, it is not possible always. These issues should be taken care of when POS taggers are built for Tamil language. For example, the Tamil root words like *adi, padi, isai, mudi, kudi* can take both noun and verb category which leads to the root ambiguity problem in POS tagging.

5.2.2 Noun Complexity

Nouns are the words which denote a person, place, thing, time, etc. In Tamil language, nouns are inflected for the number and case in morphological level. However on phonological level, four types of suffixes can occur with noun stem.
Morphological level inflection

Noun (+ number) (+ case)

Example: ᾧκκαλι <NN> pUk-kaL-ai
Flower-plural-accusative case suffix

Noun (+ number) (+ oblique) (+ euphonic) (+ case)

Example: ᾧκκαλிங <NN> pUk-kaL-in-Al
Flower-plural-euphonic suffix-accusative case suffix

Nouns are needed to be annotated into common noun, compound noun, proper noun, compound proper noun, pronoun, cardinal and ordinal. Pronouns need to be further annotated for personal pronoun. There occurs confusion between common noun and compound noun and also between proper noun and compound proper noun. Common noun can also occur as compound noun, for example

UrAdci <NNC> thalaivar <NNC>

When UrAdci and thalaivar comes together it can be compound noun (<NNC>), but when UrAdci and thalaivar comes separately in a sentence it should be tagged as a common noun (<NN>). Such complexity also occurs with the proper noun <NNP> and compound proper noun (<NNPC>). Moreover there occurs confusion between noun and adverb, pronoun and emphasis in syntactic level.

5.2.3 Verb Complexity

The verbal forms are complex in Tamil. A finite verb shows the following morphological structure

Verb stem + Tense + Person-Number + Gender

Example: நடந்ேதன் wada +wth +En <VF>
‘I walked’

A number of non-finite forms are possible: adverbial forms, adjectival forms, infinitive forms, and conditional.

Verb stem + Adverbial participle

Example: cey + thu = ceythu <VNAV>
Verb stem + relative participle
Example: cey + tha = ceytha <VNAJ>
basu + th = basuth ‘who did’

Verb stem + infinitive suffix
Example: azu + a = aza <VINT>
asuy + s = asuy ‘to weep’

Verb stem + conditional suffix
Example: kEL+d + Al = kEdAl <CVB>
basuy + s + th = basuth th ‘if asked’

Distinction needs to be made between a main verb followed by another main verb and a main verb followed by an auxiliary verb. The main verb followed by an auxiliary verb need to be interpreted together, whereas the main verb followed by another main verb need to be interpreted separately. This lead to functional ambiguity as described below:

Functional ambiguity in adverbial <VNA> form

The morphological structure of adverbial verb is

Verb root + adverbial participle
Example: sey + thu = seythu <VNA>

‘having done’
vawthu <VNA> sAppidduvudd <VNA> pO <VF>

‘Having come and having eaten went’

Functional ambiguity in adjectival <VNAJ> form

The adjectival <VNAJ> forms differ by tense markings:
Verb stem+Tense+Adjectivalizer

Example:

vandta ‘x who came’

varukiRa ‘x who comes’

varum ‘x who will come’

Adjectival<VNAJ> form allows several interpretations as given in the following examples.

sAppida ilai ‘the leaf which is eaten by x’

‘the leaf on which x had his food and ate’

um-suffixed adjectival form clashes with other homophonous forms which leads ambiguity.

varum <VNAJ>paiyan ‘the boy who will come’

varum <VF> ‘it will come’

Functional ambiguity in infinitival <VINT> form

verb_stem + infinitive suffix

Example: azu + a = aza <VINT>

atha+y = athy

vara-v-iru ‘going to come’

vara-k-kuuTaatu ‘should not come’

vara-s-so ‘ask to come’

5.2.4 Adverb Complexity

A number of adjectival and adverbial forms of verbs are lexicalized as adjectives and adverbs respectively and clash with their respective sentential adjectival and adverbial forms semantically creating ambiguity in POS tagging. Adverbs too need to be distinguished based on their source category. Many adverbs are derived by suffixing aaka with nouns in Tamil. Functional clash can be seen between noun and adverb in aaka suffixed forms. This type of clash is seen among other Dravidian languages too.

‘அவள் அழகாக இந்திக்கிறாள்’
avaL azakAka irukkiRAL

‘she beauty_ADV be_PRE_she

‘she is beautiful’

5.2.5 Postposition Complexity

Postpositions are from various categories such as verbal, nominal and adverbial in Tamil. Many a time, the demarking line between verb/noun/adverb and postposition is slim leading to ambiguity. Some postpositions are simple and some are compound. Postpositions are conditioned by the nouns inflected for case they follow. Simply tagging one form as postposition will be misleading.

There are postpositions which come after noun and also after verbs which makes the postposition ambiguous (spatial vs. temporal).

pinnAl <PPO> ‘behind’ as in vlddukkup pinnAl ‘behind the house’
pinnAl ‘after’<ADV> avanukkup pinnAl vawthAn ‘he came after him’

5.3 PART OF SPEECH TAGSET DEVELOPMENT

For developing a POS tagged corpus, it is necessary to define a Tagset (POS Tagset) used in that corpus. Collection of all the possible tags is called tagset. Tagsets differ from language to language. After referring and considering the available tagsets for Tamil and other languages, a customized Tagset named AMRITA Tagset was developed. The guidelines from “AnnCorra, IIIT Hyderabad [160]” and EAGLES, (1996), were also considered while developing the AMRITA Tagset. Guidelines followed while developing the AMRITA Tagset are given below.

1. The tags should be simple.
2. Maintaining simplicity forEase of Learning and Consistency in annotation.
3. POS tagging is not a replacement for morph analyzer.
4. A 'word' in a text carries grammatical category and grammatical features such as gender, number, person etc. The POS tag should be based on the 'category' of the word and the features can be acquired from the morph analyzer.
Another point that was considered while deciding the tags was whether to come up with a totally new tag set or take any other standard tagger as a reference and make modifications in it according to the objective of the new tagger. It was felt that the later option is often better because the tag names which are assigned by an existing tagger may be familiar to the users and thus can be easier to adopt for a new language rather than a totally new one. It saves time in getting familiar to the new tags and then work on it.

5.3.1 Available POS Tagsets for Tamil

Tagset by AUKBC

AUKBC Research Centre at Chennai developed a tagset with the help of eminent linguists from Tamil University, Tanjore. This is an exhaustive tagset, which covers almost all possible grammatical and lexical constituents. It contains 68 tags [161].

Vasu Renganathan’s Tagset

TagTamil by Vasu Ranganathan is based on Lexical phonological approach. Tag Tamil does morphotactics of morphological processing of verbs by using index method. Tag Tamil does both tagging and generation [162].

Tagset by IIIT, Hyderabad

POS Tag Set for Indian Languages was developed by IIIT, Hyderabad. Their tags are decided on coarse linguistic information with an idea to expand it to finer knowledge if required. The annotation standards for POS tagging for Indian languages include 26 tags [163].

CIIL Tagset for Tamil

This tagset was developed by CIIL (Central Institute of Indian Languages) Mysore. It contains 71 tags for Tamil. As the tagset considers noun and verb inflection, the number of tags got increased. It has 30 noun forms including pronoun categories and 25 verb forms including participle forms [164].
Ganesan’s POS Tagset

Ganesan has prepared a POS tagger for Tamil. His tagger works well in CIIL corpus. Its efficiency in other corpora has to be tested. He has a rich tagset for Tamil. He tagged a portion of CIIL corpus by using a dictionary as well as a morphological analyzer. He corrected it manually and trained the rest of the corpus with it. The tags are added morpheme by morpheme [165].

Selvam’s POS Tagset

The tagset developed by Selvam considers morphological inflections on nouns for various cases such as accusative, dative, instrumental, sociative, locative, ablative, benefactive, genitive and vocative and clitics and morphological inflections on verbs for tense etc [166].

5.3.2 AMRITA POS Tagset

The main drawback in majority of tagsets used for Tamil is that they take into account the verb and noun inflections for tagging. Hence at the tagging time, one needs to split each and every inflected word into morphemes in the corpus. It is a tough and time consuming process. At POS level, one needs to determine only the word’s grammatical category, which can be done using a limited number of tagset. The inflectional forms can be taken care of morph analyzer. So there is no need of using a large number of tags. Moreover a large number of tags will lead to more complexity which in turn reduces the tagging accuracy. Considering the complexity of Tamil in POS tagging and referring to various tagsets, a customized tagset has been developed (AMRITA POS tagset). The customized POS tagset which has been used for the present research work contains 32 tags without considering the inflections. The 32 tags are listed in the Table 5.1.

In AMRITA POS tagset, compound tags for common noun (NNC) and proper noun (NNPC) were used. Tag VBG is used for verbal nouns and participle nouns. These 32 POS tags are used for POS tagger and chunker. For morphological analyzer, these 32 tags were further simplified and reduced to 10 Tags.
Corpus linguistics seeks to further the understanding of language through the analysis of large quantities of naturally occurring data. Text corpora are used in a number of different ways. Traditionally, corpora have been used for the study and analysis of language at different levels of linguistic description. Corpora have been constructed for the specific purpose of acquiring knowledge for information extraction systems, knowledge-based systems and e-business systems [167]. Corpora have been used for studying child language development. Speech corpora play a vital role in the specification, design and implementation of telephonic communication and for the broadcast media.

There is a long tradition of corpus linguistic studies in Europe. The need for corpus for a language is multifarious. Starting from the preparation of a dictionary or lexicon to machine translation, corpus has become an inevitable resource for technological development of languages. Corpus means a body of huge text incorporating various types of textual materials, including newspaper, weeklies, fictions, scientific writings, literary writings, and so on. Corpus represents all the styles of a language. Corpus must be very huge in size as it is going to be used for many

### Table 5.1 AMRITA POS Tagset

<table>
<thead>
<tr>
<th>S.N</th>
<th>TAG</th>
<th>DESCRIPTION</th>
<th>S.N</th>
<th>TAG</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;NN&gt;</td>
<td>NOUN</td>
<td>17</td>
<td>&lt;VINT&gt;</td>
<td>VERB INFINITE</td>
</tr>
<tr>
<td>2</td>
<td>&lt;NNC&gt;</td>
<td>COMPOUND NOUN</td>
<td>18</td>
<td>&lt;CN&gt;</td>
<td>CONJUNCTION</td>
</tr>
<tr>
<td>3</td>
<td>&lt;NNP&gt;</td>
<td>PROPER NOUN</td>
<td>19</td>
<td>&lt;LV&gt;</td>
<td>LUNNUMERAL VERB</td>
</tr>
<tr>
<td>4</td>
<td>&lt;NNPC&gt;</td>
<td>COMPOUND PROPER NOUN</td>
<td>20</td>
<td>&lt;QW&gt;</td>
<td>QUESTION WORDS</td>
</tr>
<tr>
<td>5</td>
<td>&lt;ORD&gt;</td>
<td>ORDINALS</td>
<td>21</td>
<td>&lt;COM&gt;</td>
<td>COMPLEMENTIZER</td>
</tr>
<tr>
<td>6</td>
<td>&lt;CRD&gt;</td>
<td>CARDINALS</td>
<td>22</td>
<td>&lt;NNO&gt;</td>
<td>QUANTITY WORDS</td>
</tr>
<tr>
<td>7</td>
<td>&lt;PRP&gt;</td>
<td>PERSONAL PRONOUN</td>
<td>23</td>
<td>&lt;QTF&gt;</td>
<td>QUANTIFIERS</td>
</tr>
<tr>
<td>8</td>
<td>&lt;PRIN&gt;</td>
<td>PRONOUN INTROGATIVE</td>
<td>24</td>
<td>&lt;PRO&gt;</td>
<td>POSTPOSITIONS</td>
</tr>
<tr>
<td>9</td>
<td>&lt;PRID&gt;</td>
<td>PRONOUN INDEFINITE</td>
<td>25</td>
<td>&lt;DET&gt;</td>
<td>DETERMINERS</td>
</tr>
<tr>
<td>10</td>
<td>&lt;ADJ&gt;</td>
<td>ADJECTIVE</td>
<td>26</td>
<td>&lt;INT&gt;</td>
<td>INTENSIFIER</td>
</tr>
<tr>
<td>11</td>
<td>&lt;ADV&gt;</td>
<td>ADVERB</td>
<td>27</td>
<td>&lt;ECH&gt;</td>
<td>ECHO WORDS</td>
</tr>
<tr>
<td>12</td>
<td>&lt;VNA&gt;</td>
<td>VERB NON FINITE ADJECTIVE</td>
<td>28</td>
<td>&lt;LMP&gt;</td>
<td>EMPHASIS</td>
</tr>
<tr>
<td>13</td>
<td>&lt;VNAV&gt;</td>
<td>VERB NON FINITE VERB</td>
<td>29</td>
<td>&lt;COMM&gt;</td>
<td>COMMA</td>
</tr>
<tr>
<td>14</td>
<td>&lt;VBG&gt;</td>
<td>VERBAL GERUND</td>
<td>30</td>
<td>&lt;DOT&gt;</td>
<td>DOT</td>
</tr>
<tr>
<td>15</td>
<td>&lt;VP&gt;</td>
<td>VERB FINITE</td>
<td>31</td>
<td>&lt;QMP&gt;</td>
<td>QUESTION MARK</td>
</tr>
<tr>
<td>16</td>
<td>&lt;VAX&gt;</td>
<td>VERB AUXILIARY</td>
<td>32</td>
<td>&lt;RDW&gt;</td>
<td>REDUPLICATION WORDS</td>
</tr>
</tbody>
</table>

### 5.4 DEVELOPMENT OF TAMIL POS CORPORA FOR PREPROCESSING
language applications such as preparation of lexicons of different sizes, purposes and types, NLP tools, machine translation programs and so on.

5.4.1 Untagged and Tagged Corpus

Untagged or un-annotated corpus provides limited information to the users. Corpus can be augmented with additional information by way of labeling the morpheme, word, phrase and sentence for their grammatical values. Such information helps the user to retrieve information selectively and easily. Figure 5.1 presents an example of untagged corpus.

The frequency of the lemma is useful in the analysis of the corpus. When the frequency of a particular word is compared to other context words, it is useful to find whether the word is common or rare. The frequencies are relatively reliable for the most common words in a corpus, but to analyze the senses and association patterns of words, a very large number of occurrences with a very large corpus containing many different texts, a wider range of topics should be represented, so that the frequencies of words are less influenced by individual texts. Frequency list based on an untagged corpus are limited in usefulness, because they do not provide grammatical uses which are common or rare. Tagged corpus is an important dataset for NLP applications. Figure 5.2 shows an example of tagged corpus.

Figure 5.1 Example of Untagged Corpus

Figure 5.2 Example of Tagged Corpus
5.4.2 Available Corpus for Tamil

Corpus can be distinguished as tagged corpus, parallel corpus and aligned corpus. The tagged corpus is that which is tagged for Part-of-Speech, morphology, lemma, phrases etc. A parallel corpus contains texts and translations in each of the languages involved in it. It allows wider scopes for double-checking of the translation equivalents. Aligned corpus is a kind of bilingual corpus where text samples of one language and their translations into another language are aligned, sentence by sentence, phrase by phrase, word by word, or even character by character.

CIIL Corpus for Tamil

As far as building corpus for the Indian languages is concerned, it was Central Institute of Indian Languages (CIIL) which took the initiative and started preparing corpus for some of the Indian languages (Tamil, Telugu, Kannada, and Malayalam). Department of Electronics (DOE) financed the corpus-building project. The target was to prepare corpus with ten million words for each language. But, due to financial crunch and time restriction, it ended up with just three million words for each language. Tamil corpus, with three million words, is built by CIIL in this way. It is a partially tagged corpus.

AUKBC-RC’s Improved Tagged Corpus for Tamil

AUKBC Research Centre which has taken up NLP oriented works for Tamil, has improved upon the CIIL Tamil Corpus and tagged it for their MT programs. It also developed a parallel corpus for English-Tamil to promote its goal of preparing an MT tool for English-Tamil translation. Parallel corpus is very useful for training the corpus and for building example based machine translation. Parallel corpus is a useful tool for MT programs.

5.4.3 POS Tagged Corpus Development

The tagged corpus is the immediate requirement for different analyses in the field of Natural Language Processing. Most of the language processing works are in need of such large database of texts, which provide a real, natural, native language of varying types. Annotation of corpora can be done at various levels through, Part of Speech,
phrase/clause level, dependency level, etc. Part of Speech tagging forms the basic step towards building an annotated corpus.

For creating Tamil Part of Speech Tagger, a grammatically tagged corpus is needed. So a tagged corpus size of 500k words was built. Sentences from Dinamani newspaper, yahoo Tamil news and Tamil short stories etc were collected and tagged.

POS corpus tagging is done in three stages:

1. Pre-editing
2. Manual Tagging
3. Bootstrapping

In pre-editing, untagged corpus is converted to a suitable format for SVMTool, in order to assign a Part of Speech tag to each word. Because of orthographic similarity, one word may have several possible POS tags. After an initial assignment of possible POS tags, words are manually tagged using AMRITA tagset. The tagged corpus is trained using SVMTlearn component. After training, the new untagged corpus is tagged using SVMTagger. The output of SVMTagger is again manually corrected and added into the tagged corpus to increase the corpus size. The corpora development process is elaborated below.

**Pre-editing**

Tamil text documents have been collected from Dinamani website, Yahoo Tamil, Tamil short stories etc (For example, Figure 5.3). The corpus has been cleaned using simple program i.e. to remove punctuations (except dots, commas and question marks). The corpus has been sententially aligned. The next step is to change the corpora into a column format because the SVMTool training data must be in column format, i.e. a token (word) per line corpus in a sentence by sentence fashion. The column separator is the blank space.

Figure 5.3 Untagged Corpus before Pre-editing
**Manual tagging**

After pre-editing, the untagged corpus is tokenized into column format (Figure 5.4). In second stage, the untagged corpus is manually POS tagged using AMRITA tagset. Initially 10,000 words were manually tagged. During manual POS tagging process, great difficulties were faced while assigning tags to corpora.

**Bootstrapping**

After completing the manual tagging, the tagged corpus is given to the learning component of training algorithm for generating the model. Using the model generated, decoder of the training algorithm tags the untagged sentences. The output of the component is a tagged corpus with some error. Then the tags are corrected manually. After correcting the tags, the tagged corpus was added into the training corpus for increasing the size of training corpus.

![Figure 5.4 Untagged Corpus after Pre-editing](image-url)
5.4.4 Applications of POS Tagged Corpus

The POS tagged corpus is used in the following tasks.

- Chunking
- Parsing
- Information extraction and retrieval
- Machine Translation
- Tree bank creation
- Document classification
- Question answering
- Automatic dialogue system
- Speech processing
- Summarization
- Statistical training of Language models
- Machine Translation using multilingual corpora
- Text checkers for evaluating spelling and grammar
- Computer Lexicography
- Educational application like Computer Assisted Language Learning

5.4.5 Details of POS Tagged Corpus Developed

The POS tagged corpus details are given in the corpus statistics and tag count table (Table 5.2 and 5.3).

<table>
<thead>
<tr>
<th>Table 5.2 Corpus Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>No of sentences</td>
</tr>
<tr>
<td>No of words</td>
</tr>
<tr>
<td>No of distinct words</td>
</tr>
</tbody>
</table>
Table 5.3 Tag Count

<table>
<thead>
<tr>
<th>S.No</th>
<th>Tags</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;ADJ&gt;</td>
<td>17400</td>
</tr>
<tr>
<td>2</td>
<td>&lt;ADV&gt;</td>
<td>22150</td>
</tr>
<tr>
<td>3</td>
<td>&lt;CNJ&gt;</td>
<td>6853</td>
</tr>
<tr>
<td>4</td>
<td>&lt;COM&gt;</td>
<td>8488</td>
</tr>
<tr>
<td>5</td>
<td>&lt;COMM&gt;</td>
<td>3955</td>
</tr>
<tr>
<td>6</td>
<td>&lt;CRD&gt;</td>
<td>12600</td>
</tr>
<tr>
<td>7</td>
<td>&lt;CVB&gt;</td>
<td>2315</td>
</tr>
<tr>
<td>8</td>
<td>&lt;DET&gt;</td>
<td>6368</td>
</tr>
<tr>
<td>9</td>
<td>&lt;DOT&gt;</td>
<td>43514</td>
</tr>
<tr>
<td>10</td>
<td>&lt;ECH&gt;</td>
<td>8</td>
</tr>
<tr>
<td>11</td>
<td>&lt;EMP&gt;</td>
<td>2777</td>
</tr>
<tr>
<td>12</td>
<td>&lt;INT&gt;</td>
<td>2289</td>
</tr>
<tr>
<td>13</td>
<td>&lt;NN&gt;</td>
<td>128349</td>
</tr>
<tr>
<td>14</td>
<td>&lt;NNC&gt;</td>
<td>74575</td>
</tr>
<tr>
<td>15</td>
<td>&lt;NNP&gt;</td>
<td>34594</td>
</tr>
<tr>
<td>16</td>
<td>&lt;NNPC&gt;</td>
<td>9042</td>
</tr>
<tr>
<td>17</td>
<td>&lt;NNQ&gt;</td>
<td>1911</td>
</tr>
<tr>
<td>18</td>
<td>&lt;ORD&gt;</td>
<td>1773</td>
</tr>
<tr>
<td>19</td>
<td>&lt;PPO&gt;</td>
<td>7467</td>
</tr>
<tr>
<td>20</td>
<td>&lt;PRID&gt;</td>
<td>399</td>
</tr>
<tr>
<td>21</td>
<td>&lt;PRIN&gt;</td>
<td>789</td>
</tr>
<tr>
<td>22</td>
<td>&lt;PRP&gt;</td>
<td>15559</td>
</tr>
<tr>
<td>23</td>
<td>&lt;QM&gt;</td>
<td>1615</td>
</tr>
<tr>
<td>24</td>
<td>&lt;QTF&gt;</td>
<td>922</td>
</tr>
<tr>
<td>25</td>
<td>&lt;QW&gt;</td>
<td>2389</td>
</tr>
<tr>
<td>26</td>
<td>&lt;RDW&gt;</td>
<td>273</td>
</tr>
<tr>
<td>27</td>
<td>&lt;VAX&gt;</td>
<td>7793</td>
</tr>
<tr>
<td>28</td>
<td>&lt;VBG&gt;</td>
<td>9294</td>
</tr>
<tr>
<td>29</td>
<td>&lt;VF&gt;</td>
<td>34888</td>
</tr>
<tr>
<td>30</td>
<td>&lt;VINT&gt;</td>
<td>11604</td>
</tr>
<tr>
<td>31</td>
<td>&lt;VNAJ&gt;</td>
<td>17843</td>
</tr>
<tr>
<td>32</td>
<td>&lt;VNAV&gt;</td>
<td>20671</td>
</tr>
</tbody>
</table>
5.5 DEVELOPMENT OF POS TAGGER USING SVMTOOL

5.5.1 SVMTool

This section presents the SVMTool, a simple, flexible, and effective generator of sequential taggers based on Support Vector Machines (SVM) and explains how it is applied to the problem of Part-of-Speech tagging. This SVM-based tagger is robust and flexible for feature modeling (including lexicalization), trains efficiently with almost no parameters to tune, and is able to tag thousands of words per second, which makes it really practical for real NLP applications. Regarding accuracy, the SVM-based tagger significantly outperforms the TnT tagger [39] exactly under the same conditions, and achieves a very competitive accuracy of 94.6% for Tamil.

Generally, tagging is required to be as accurate as possible, and as efficient as possible. But, certainly, there is a conflict between these two desirable properties. This is so because obtaining a higher accuracy relies on processing more and more information. However, sometimes, depending on the kind of application, a loss in efficiency may be acceptable in order to obtain more precise results. Or the other way around, a slight loss in accuracy may be tolerated in favor of tagging speed.

Moreover, some languages like Tamil have a richer morphology than others. This leads the tagger to have a large set of feature patterns. Also, the tagset size and ambiguity rate may vary from language to language and from problem to problem. Besides, if few data are available for training, the proportion of unknown words may be huge. Sometimes, morphological analyzers could be utilized to reduce the degree of ambiguity when facing unknown words. Thus, a sequential tagger should be flexible with respect to the amount of information utilized and context shape.

Another very interesting property for sequential taggers is their portability. Multilingual information is a key ingredient in NLP tasks such as Machine Translation, Information Retrieval, Information Extraction, Question Answering and Word Sense Disambiguation. Therefore, having a tagger that works equally well for several languages is crucial for the system robustness. For some languages, lexical resources are hard to obtain. Therefore, ideally, a tagger should be capable for learning with fewer annotated data. The SVMTool is intended to comply with all the requirements of modern NLP technology, by combining simplicity, flexibility, robustness, portability.
and efficiency with state-of-the-art accuracy. This is achieved by working in the Support Vector Machines (SVM) learning framework, and by offering NLP researchers a highly customizable sequential tagger generator. The SVMTool which is a language independent sequential tagger is applied to Tamil POS tagging.

### 5.5.2 Features of SVMTool

The following are the features of the SVMTool [10].

**Simplicity:** The SVMTool is easy to configure and train. The learning is controlled by means of a very simple configuration file. There are very few parameters to tune. And the tagger itself is very easy to use, accepting standard input and output pipelining. Embedded usage is also supplied by means of the SVMTool API.

**Flexibility:** The size and shape of the feature context can be adjusted. Also, rich features can be defined, including word and POS (tag) n-grams as well as ambiguity classes and “may be’s”, apart from lexicalized features for unknown words and sentence general information. The behavior at tagging time is also very flexible, allowing different strategies.

**Robustness:** The over fitting problem is well addressed by tuning the C parameter in the soft margin version of the SVM learning algorithm. Also, a sentence-level analysis may be performed in order to maximize the sentence score. And, for unknown words not to punish so severely on the system effectiveness, several strategies have been implemented and tested.

**Portability:** The SVMTool is language independent. It has been successfully applied to English and Spanish without a priori knowledge other than a supervised corpus. Moreover, thinking of languages for which labeled data is a scarce resource, the SVMTool also may learn from unsupervised data based on the role of non-ambiguous words with the only additional help of a morpho-syntactic dictionary.

**Accuracy:** Compared to state-of-the-art POS taggers reported up to date, it exhibits a very competitive accuracy. Clearly, rich sets of features allow modeling very precisely most of the information involved. Also the learning paradigm, SVM, is highly suitable for working accurately and efficiently with high dimensionality feature spaces.
**Efficiency:** Performance at tagging time depends on the feature set size and the tagging scheme selected. For the default (one-pass left-to-right greedy) tagging scheme, it exhibits a tagging speed of 1,500 words/second whereas the C++ version achieves a tagging speed of over 10,000 words/second. This has been achieved by working in the primal formulation of SVM. The use of linear kernels causes the tagger to perform more efficiently both at tagging and learning time, but forces the user to define a richer feature space.

### 5.5.3 Components of SVMTool

The SVMTool [12] software package consists of three main components, namely the model learner (SVMTlearn), the tagger (SVMTagger) and the evaluator (SVMEval). Previous to the tagging, SVM models (weight vectors and biases) are learned from a training corpus using the SVMTlearn component. Different models are learned for different strategies. Then, at tagging time, using the SVMTagger component, one may choose the tagging strategy that is most suitable for the purpose of tagging. Finally, given a correctly annotated corpus and the corresponding SVMTool predicted annotation, the SVMEval component displays tagging results.

#### 5.5.3.1 SVMTlearn

Given a set of examples (either annotated or unannotated for training), the SVMTlearn trains a set of SVM classifiers. So as to do that, it makes use of SVM-light, an implementation of Vapnik’s SVMs in C, developed by Thorsten Joachim’s (2002). The SVMlight software implementation of Vapnik’s Support Vector Machine by Thorsten Joachim’s has been used to train the models.

**Training Data Format**

Training data must be in column format, i.e. a token per line corpus in a sentence by sentence format. The column separator is the blank space. The word is to be the first column of the line. The tag to predict takes the second column in the output. The rest of the line may contain additional information. Example is given below in Figure 5.5. No special ‘<EOS>’ mark is employed for sentence separation. Sentence punctuation is used instead, i.e. [.!?] symbols are taken as unambiguous sentence separators. In this system these symbols [.?] are used as sentence separators.
Figure 5.5 Training Data Format

**Known words features**

\[ C(0;-1) \ C(0;0) \ C(0;1) \ C(0;-2,-1) \ C(0;-1,0) \ C(0;0,1) \ C(0;-1,1) \]
\[ C(0;1,2) \ C(0;-2,-1,0) \ C(0;-1,0,1) \ C(0;0,1,2) \ C(1;-1) \ C(1;0,1) \ C(1;-1,0) \]
\[ C(1;0) \ k(0) \ k(1) \ k(2) \ m(0) \ m(1) \ m(2) \]

**Unknown words features**

\[ C(0;-1) \ C(0;0) \ C(0;1) \ C(0;-2,-1) \ C(0;-1,0) \ C(0;0,1) \ C(0;-1,1) \]
\[ C(0;1,2) \ C(0;-2,-1,0) \ C(0;-1,0,1) \ C(0;0,1,2) \ C(1;-1) \ C(1;0,1) \ C(1;-1,0) \]
\[ C(1;0) \ k(0) \ k(1) \ k(2) \ m(0) \ m(1) \ m(2) \ a(2) \ a(3) \ a(4) \ a(5) \ a(6) \ a(7) \ a(8) \]
\[ a(9) \ a(10) \ a(11) \ a(12) \ a(13) \ a(14) \ a(15) \ z(2) \ z(3) \ z(4) \ z(5) \ z(6) \ z(7) \]
\[ z(8) \ z(9) \ z(10) \ z(11) \ z(12) \ z(13) \ z(14) \ z(15) \ ca(1) \ cz(1) \ L \ SN \ CF \ CN \]
Models

Five different kinds of models have been implemented in this Tool. Models 0, 1, and 2 differ only in the features they consider. Model 3 and Model 4 are just like Model 0 with respect to feature extraction but examples are selected in a different manner. Model 3 is for unsupervised learning. Hence, given an unlabeled corpus and a dictionary, at learning time it can only count on knowing the ambiguity class, and the POS information only for unambiguous words. Model 4 achieves robustness by simulating unknown words in the learning context at training time.

Model 0: This is the default model. The unseen context remains ambiguous. It was thought of having in mind the one-pass on-line tagging scheme, i.e. the tagger goes either left-to-right or right-to-left making decisions. So, past decisions feed future ones in the form of POS features. At tagging time, only the parts-of-speech of already disambiguated tokens are considered. For the unseen context, ambiguity classes are considered instead (Table 5.4).

Model 1: This model considers the unseen context already disambiguated in a previous step. So it is thought for working at a second pass, revisiting and correcting already tagged text (Table 5.5).

Model 2: This model does not consider pos features at all for the unseen context. It is designed to work at a first pass, requiring Model 1 to review the tagging results at a second pass (Table 5.6).

Model 3: The training is based on the role of unambiguous words. Linear classifiers are trained with examples of unambiguous words extracted from an unannotated corpus. So, fewer POS information is available. The only additional information required is a morpho-syntactic dictionary.

Model 4: The errors caused by unknown words at tagging time punish the system severely. So as to reduce this problem, during learning, some words are artificially marked as unknown in order to learn a more realistic model. The process is very simple. The corpus is divided in a number of folders. Before starting to extract samples from each of the folders, a dictionary is generated out from the rest of folders. So, the words appearing in a folder but not in the rest are unknown words to the learner.
### Table 5.4 Example of Suitable POS Features for Model 0

<table>
<thead>
<tr>
<th>Ambiguity classes</th>
<th>$a_0, a_1, a_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>May_be’s</td>
<td>$m_0, m_1, m_2$</td>
</tr>
<tr>
<td>POS Features</td>
<td>$p_{-2}, p_{-1}, p_{+1}, p_{+2}$</td>
</tr>
<tr>
<td>POS Bigrams</td>
<td>$(p_{-2}, p_{-1})</td>
</tr>
<tr>
<td>POS Trigrams</td>
<td>$(p_{-2}, p_{-1}, a_0)</td>
</tr>
<tr>
<td>Single characters</td>
<td>ca(1), cz(1)</td>
</tr>
<tr>
<td>Prefixes</td>
<td>a(2), a(3), a(4)</td>
</tr>
<tr>
<td>Suffixes</td>
<td>z(2), z(3), z(4)</td>
</tr>
<tr>
<td>Lexicalized features</td>
<td>SA, CAA, AA, SN, CP, CN, CC, MW,L</td>
</tr>
<tr>
<td>Sentence_info</td>
<td><em>punctuation</em> (’’, ’?’, ’!’)</td>
</tr>
</tbody>
</table>

### Table 5.5 Example of Suitable POS Features for Model 1

<table>
<thead>
<tr>
<th>Ambiguity classes</th>
<th>$a_0, a_1, a_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>May_be’s</td>
<td>$m_0, m_1, m_2$</td>
</tr>
<tr>
<td>POS Features</td>
<td>$p_{-2}, p_{-1}, p_{+1}, p_{+2}$</td>
</tr>
<tr>
<td>POS Bigrams</td>
<td>$(p_{-2}, p_{-1})</td>
</tr>
<tr>
<td>POS Trigrams</td>
<td>$(p_{-2}, p_{-1}, a_0)</td>
</tr>
<tr>
<td>Single characters</td>
<td>ca(1), cz(1)</td>
</tr>
<tr>
<td>Prefixes</td>
<td>a(2), a(3), a(4)</td>
</tr>
<tr>
<td>Suffixes</td>
<td>z(2), z(3), z(4)</td>
</tr>
<tr>
<td>Lexicalized features</td>
<td>SA, CAA, AA, SN, CP, CN, CC, MW,L</td>
</tr>
<tr>
<td>Sentence_info</td>
<td><em>punctuation</em> (’’, ’?’, ’!’)</td>
</tr>
</tbody>
</table>
Table 5.6  Example of Suitable POS Features for Model 2

<table>
<thead>
<tr>
<th>Ambiguity classes</th>
<th>$a_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>May_be’s</td>
<td>$m_0$</td>
</tr>
<tr>
<td>POS Features</td>
<td>$p_{-2},p_{-1}$</td>
</tr>
<tr>
<td>POS Bigrams</td>
<td>$(p_{-2},p_{-1})$</td>
</tr>
<tr>
<td>POS Trigrams</td>
<td>$(p_{-2},p_{-1},a_0)$</td>
</tr>
<tr>
<td>Single characters</td>
<td>ca(1), cz(1)</td>
</tr>
<tr>
<td>Prefixes</td>
<td>a(2), a(3), a(4)</td>
</tr>
<tr>
<td>Suffixes</td>
<td>z(2), z(3), z(4)</td>
</tr>
<tr>
<td>Lexicalized features</td>
<td>SA, CAA, AA, SN, CP, CN, CC, MW,L</td>
</tr>
<tr>
<td>Sentence_info</td>
<td>punctuation (',','?','!')</td>
</tr>
</tbody>
</table>

**SVMTlearn for Tamil POS Tagging**

The SVMTlearn is the primary component in SVMTool. It is used for training the tagged corpus using SVMlight. This component works in Linux Operating System. POS Tagged corpus is required for training. However, if there is enough data, it is a good practice to split it into three working sets (i.e. training, validation and test). That will allow the system to train, tune and evaluate it. With less data also one can still train, tune and test the system through cross-validation. But the accuracy and efficiency will suffer.

In this component, input is POS tagged training corpus. The training corpus is given to the SVMTlearn component that trains the model using the features given in the configuration file. The features are defined in the configuration file, based on Tamil language. The outputs of the SVMTlearn are dictionary file, merged files for unknown and known words (for all models). Each merged file contains all the features of known words and unknown words (Figure 5.6).
Figure 5.6 Implementation of SVMTlearn

Example for training output of SVMTlearn

-------------------------------
SVMTool v1.3
(C) 2006 TALP RESEARCH CENTER.
Written by Jesus Gimenez and Lluis Marquez.
-------------------------------

TRAINING SET = /media/disk-1/SVM/SVMTool-1.3/bin/TAMIL_CORPUS.TRAIN
-------------------------------

DICTIONARY <TAMIL_CORPUS.DICT> [31605 words]

*****************************************************************************
*****************************************************************************

BUILDING MODELS... [MODE = 0 :: DIRECTN = LR]
*****************************************************************************
*****************************************************************************
C-PARAMETER TUNING by 10-fold CROSS-VALIDATION
on </media/disk-1/SVM/SVMTool-1.3/bin/TAMIL_CORPUS.TRAIN>
on <MODE 0> <DIRECTION LR> [KNOWN]
C-RANGE = [0.01..1] :: [log] :: #LEVELS = 3 :: SEGMENTATION RATIO = 10

LEVEL = 0 :: C-RANGE = [0.01..1] :: FACTOR = [* 10 ]

----------

TEST ACCURACY: 90.6093%
KNOWN[92.886% ] AMBIG.KNOWN [ 83.3052% ] UNKNOWN [ 78.5781% ]
TEST ACCURACY: 90.392%
KNOWN[92.6809% ] AMBIG.KNOWN [ 82.838% ] UNKNOWN [ 78.0815% ]
TEST ACCURACY: 90.1015%
KNOWN [ 92.6128% ] AMBIG.KNOWN [ 83.4766% ] UNKNOWN [ 77.5075% ]
TEST ACCURACY: 89.7127%
KNOWN [ 92.0721% ] AMBIG.KNOWN [ 81.8731% ] UNKNOWN [ 77.5281% ]
TEST ACCURACY: 90.7699%
KNOWN [ 92.7304% ] AMBIG.KNOWN [ 83.4785% ] UNKNOWN [ 80.3874% ]
TEST ACCURACY: 89.8988%
KNOWN [ 92.3462% ] AMBIG.KNOWN [ 81.5675% ] UNKNOWN [ 77.286% ]
TEST ACCURACY: 90.8836%
KNOWN [ 92.9671% ] AMBIG.KNOWN [ 83.6309% ] UNKNOWN [ 79.5591% ]
TEST ACCURACY: 89.9724%
KNOWN [ 92.4002% ] AMBIG.KNOWN [ 82.1854% ] UNKNOWN [ 77.664% ]
TEST ACCURACY: 90.2643%
KNOWN [ 92.5675% ] AMBIG.KNOWN [ 83.0289% ] UNKNOWN [ 78.0907% ]
TEST ACCURACY: 90.7494%
KNOWN [ 92.7798% ] AMBIG.KNOWN [ 82.7494% ] UNKNOWN [ 79.8929% ]
OVERALL ACCURACY [Ck = 0.01 :: Cu = 0.07975] : 90.33539%
KNOWN [ 92.6043% ] AMBIG.KNOWN [ 82.81335% ] UNKNOWN [ 78.45753% ]
MAX ACCURACY -> 90.33539 :: C-value = 0.01 :: depth = 0 :: iter = 1

----
*********************************************************************** level - 0 : ITERATION 1 - C = 0.1 - [M0 :: LR]
***********************************************************************

TEST ACCURACY: 91.7702%
KNOWN [ 94.2402% ] AMBIG.KNOWN [ 87.5492% ] UNKNOWN [ 78.7175% ]
TEST ACCURACY: 91.8881%
KNOWN [ 94.4737% ] AMBIG.KNOWN [ 88.4324% ] UNKNOWN [ 77.9821% ]
TEST ACCURACY: 91.3219%
KNOWN [ 94.0596% ] AMBIG.KNOWN [ 88.0441% ] UNKNOWN [ 77.5928% ]
TEST ACCURACY: 91.0615%
KNOWN [ 93.6037% ] AMBIG.KNOWN [ 86.6795% ] UNKNOWN [ 77.9326% ]
TEST ACCURACY: 92.0852%
KNOWN [ 94.2575% ] AMBIG.KNOWN [ 88.3275% ] UNKNOWN [ 80.5811% ]
TEST ACCURACY: 91.3927%
KNOWN [94.1299% ] AMBIG.KNOWN [ 87.4226% ] UNKNOWN [ 77.286% ]
TEST ACCURACY: 91.9891%
KNOWN[94.2944% ] AMBIG.KNOWN [ 88.0182% ] UNKNOWN [ 79.4589% ]
TEST ACCURACY: 91.3063%
KNOWN[93.9605% ] AMBIG.KNOWN [ 87.1258% ] UNKNOWN [ 77.8502% ]
TEST ACCURACY: 91.3654%
KNOWN[93.8499%] AMBIG.KNOWN [87.2127%] UNKNOWN [78.2339%]
TEST ACCURACY: 91.8693%
KNOWN [94.1%] AMBIG.KNOWN [87.0546%] UNKNOWN [79.9416%]
OVERALL ACCURACY [Ck = 0.1 :: Cu = 0.07975] : 91.60497%
KNOWN [94.09694%] AMBIG.KNOWN [87.58666%] UNKNOWN [78.55767%]
MAX ACCURACY -> 91.60497 :: C-value = 0.1 :: depth = 0 :: iter = 2

5.5.3.2 SVMTagger

Given a text corpus (one token per line) and the path to a previously learned SVM model (including the automatically generated dictionary), it performs the POS tagging of a sequence of words. The tagging goes on-line, based on a sliding window which gives a view of the feature context to be considered at every decision.

In any case, there are two important concepts to be considered:

- Example generation
- Feature extraction

**Example generation:** This step is to define what an example is, according to the concept in which the machine is to be learned. For instance, in POS tagging, the machine has to correctly classify the words according to their POS. Thus, every POS tag is the class of a word that generates a positive example for its class, and a negative example for the rest of the classes. Therefore, every sentence may generate a large number of examples.

**Feature Extraction:** The set of features based on the algorithm to be used have to be defined. For instance, the POS tags should be guessed according to the preceding and following words. Thus, every example is represented by a set of active features. These representations will be the input for the SVM classifiers. If the working of SVMTool has to be learned, it is necessary to run the SVMTlearn (Perl version). By setting the REMOVE_FILES (in the configuration file) option to 0, it will not remove the intermediate files; if option 1 is given, it will remove all the intermediate files.

Feature extraction is performed by the sliding window object. A sliding window works on a very local context (as defined in the CONFIG file), usually a 5 words context [-2, -1, 0, +1, +2], being the current word under analysis at the core position.
Taking this context into account, a number of features may be extracted. The feature set depends on, how the tagger is going to proceed later (i.e., the context and information that’s going to be available at tagging time). Generally, all the words are known before tagging, but POS tag is available only for some words (those already tagged).

In tagging stage, if the input word is known and ambiguous, the word is tagged (i.e., classified), and the predicted tag feeds forward next decisions. This will be done in the "sub classify_sample_merged ()" subroutine in the SVMTAGGER file. In order to speed up SVM classification, merge mapping and SVM weights and biases, into a single file. Therefore, when a new example is to be tagged, the tagger just accesses the merged model and for every active feature, retrieves the associated weight. Then, for every possible tag, the bias will also be retrieved. Finally, SVM classification rule (i.e., scalar product + bias) is applied.

Example:

kUddukkudiwIr thiddam wiRaivERRappadum enRAr muthalvar

Tags:

<NNC>       <NNC>  <VNAJ>/<VF>     <VF>   <NN>  
(Ambiguity)

For tagging this sentence, first take the active features like w-1, w+1, i.e., to predict POS-tags based only on the preceding and following words. Here the ambiguity word is “wiRaivERRappadum”. The correct tag of this ambiguous word is <VF>.

“w0, the current word is wiRaivERRappadum, the active features are "w-1 is thiddam " and "w+1 is enRAr ".

while applying the SVM classification rule for a given POS Tagger, it is necessary to go to the merged model and retrieve the weight for these features, and the bias (first line after the header, beginning with "BIASES "), corresponding to the given POS. For instance, suppose this ".MRG" file:
BIASES  <ADJ>: 0.37059487  <ADV>: -0.19514606  <CNJ>: 0.43007979
<COM>: -0.037037037  <CRD>: 0.55448766  <CVB>: -0.19911161  <DET>: -1.1815452
<EMP>: -0.86491783  <INT>: 0.61775334  <NN>: -0.21980137  <NNC>: 1.3656117
<NNP>: 0.072242349  <NNFC>: 0.7906585  <NNQ>: 0.44012828  <ORD>: 0.30304924
<PPO>: -0.2182171  <PRI>: 0.89491131  <PRID>: -0.15550162  <PRIN>: 0.56913633
<PRF>: 0.35316978  <QW>: 0.039121434  <RDW>: 0.84771943  <VAX>: 0.041690388
<VBG>: 0.23199934  <VF>: 0.33486366  <VINT>: 0.0048185684  <VNAJ>: 0.42063524
<VNAV>: 0.18009116

C0~1:thiddam  <CRD>: 0.00579042912371902  <NN>: 0.532690716551973
<NNC>: -0.508699048073652  <ORD>: -0.000698015879911668
<VBG>: 0.142313085089229  <VF>: 0.296699729267891  <VNAJ>: -0.32
C0~1:enRAr  <VAX>: 0.132726597682121  <VF>: 0.66667135122578
<VNAJ>: -0.67633251479603

The SVM score for “wiRaivERRappadum” being <VNAJ> is:

\[
\text{Weight ("w-1: thiddam", "VNAJ") + weigh("w+1: enRAr", "VNAJ") - bias("VNAJ") = (-0.32) + (-0.67633251479603) - (0.42063524)} = -1.416967781749603
\]

The SVM score for “wiRaivERRappadum” being <VF> is:

\[
\text{Weight ("w-1: thiddam", "VF") + weight ("w+1: enRAr", "VF") - bias ("VF") = (0.296699729267891) + (0.66667135122578) - (0.33486366)} = 0.6285047420493671
\]

Here SVM score for <VF> is more compared to <VNAJ>, So, the tag VF is assigned to the word ‘wiRaivERRappadum’.

Calculated part-of-speech tags feed directly forward next tagging decisions as context features. The SVMTagger component works on standard input/output. It processes a token per line corpus in a sentence by sentence fashion. The token is expected to be the first column of the line. The predicted tag will take the second column in the output. The rest of the line remains unchanged. Lines beginning with ‘##’ are ignored by the tagger. Figure 5.7 is an example of input file. SVMTagger will consider only the first column of the input file. Figure 5.8 shows an example of output file.
In SVMTagger component, the important options are strategies and backup lexicon. Here, it is important to choose the tagging strategy that is going to be used. This may depend, for instance, on efficiency requirements. If the tagging must be as fast as possible, then one should forget about strategies 1, 5, and 6, because strategy 1 goes in two passes and strategies 5 and 6 perform a sentence-level tagging. Strategy 3 is only for unsupervised learning (no hand-annotated data is needed). To choose among strategies 0, 2 and 4, the best solution is to try them all. If unknown words are known to the tagger at tagging time, strategies 2 and 4 are more robust than strategy 0. If any speed requirement or information about future data is not needed, the tagging strategies 4 and 6 systematically show best results.
Here the format of backup lexicon file is same as the dictionary format. So a PERL program can be used for converting a tagged corpus into a dictionary format. Tagging will be complex for open tag categories. The main drawback in POS tagging is tagging the proper nouns. For English, they use capitalization for tagging the proper noun words. But in Tamil, it is not possible; therefore a large backup lexicon with proper nouns is provided to the system. A large dataset for proper noun (Indian place and person names) was collected and given as the input to the morphological generator (using PERL program). Morph generator generates nearly twelve inflections for every proper noun. This new dataset is converted into SVMTool dictionary format and given to SVMTagger as a back up lexicon. Figure 5.9 shows the steps in implementation of SVMTagger for Tamil. The input to the system is an untagged cleaned Tamil corpus and output is tagged or annotated corpus. Supporting files are training corpus, dictionary file, merged models for unknown and known words and backup lexicon.

Figure 5.9 Implementation of SVMTagger
5.5.3.3 SVMTeval

Given a SVMTool predicted tagging output and the corresponding gold-standard, SVMTeval evaluates the performance in terms of accuracy. It is a very useful component for the tuning of the system parameters, such as the C parameter, the feature patterns and filtering, the model compression etc. Based on a given morphological dictionary (e.g., the automatically generated at training time), results may be presented also for different sets of words (known words vs. unknown words, ambiguous words vs. unambiguous words). A different view of these same results can be seen from the class of ambiguity perspective too, i.e., words sharing the same kind of ambiguity may be considered together. Also, words sharing the same degree of disambiguation complexity, determined by the size of their ambiguity classes, can be grouped.

Usage: SVMTeval [mode] <model> <gold> <pred>

- mode: 0 - complete report (everything)
  1 - overall accuracy only [default]
  2 - accuracy of known vs. unknown words
  3 - accuracy per level of ambiguity
  4 - accuracy per kind of ambiguity
  5 - accuracy per class
- model: model name
- gold: correct tagging file
- pred: predicted tagging file

Example: SVMTeval TAMIL_CORPUS_4L TAMIL.GOLD TAMIL.OUT

SVMTeval for Tamil

SVMTeval is the last component of SVMTool. In this, the component is used to evaluate the outputs based on different modes. The main input of this component is a correctly tagged corpus, also called gold standard (Figure 5.10).
**SVMTeval report**

**Brief report**

By default, a brief report mainly returning the overall accuracy is elaborated. It also provides information about the number of tokens processed, and how much were known/unknown and ambiguous/unambiguous according to the model dictionary.

Results are always compared to the most-frequent-tag (MFT) baseline.

```
*-----------------------------SVMTevalreport-----------------------------
******
* model               = [E:\SVMTool-1.3\bin\CORPUS]
* testset (gold)      = [E:\SVMTool-1.3\bin\files\test.gold]
* testset (predicted) = [E:\SVMTool-1.3\bin\files\test.out]
* 
================================================================
========
EVALUATING <E:\SVMTool-1.3\bin\files\test.out> vs. <E:\SVMTool-1.3\bin\files\test.gold> on model <E:\SVMTool-1.3\bin\CORPUS>..
*-----------------------------TAGGINGSUMMARY-----------------------------
#TOKENS             = 1063
```
AVERAGE_AMBIGUITY = 6.4901 tags per token
*  ---------------------------------------------------------------

#KNOWN            = 80.3387% -->              854 / 1063
#UNKNOWN          = 19.6613% -->              209 / 1063
#AMBIGUOUS        = 21.7310% -->              231 / 1063
#MFT baseline     = 71.2135% -->              757 / 1063

*=============OVERALLACCURACY=========================================

HITS          TRIALS          ACCURACY               MFT
*  ---------------------------------------------------------------

                      1002             1063          94.2615%          71.2135%

*  ================================================================

Known vs. unknown tokens

Accuracy for four different sets of words is returned. The first set is that of all known tokens, tokens which were seen during the training. The second and third sets contain respectively all ambiguous and all unambiguous tokens among these known tokens. Finally, there is the set of unknown tokens, which were not seen during the training.

*=========================SVMTevalreport

* model               = [E:\SVMTool-1.3\bin\CORPUS]
* testset (gold)      = [E:\SVMTool-1.3\bin\files\test.gold]
* testset (predicted) = [E:\SVMTool-1.3\bin\files\test.out]
*  ---------------------------------------------------------------

==
EVALUATING <E:\SVMTool-1.3\bin\files\test1.out> vs. <E:\SVMTool-1.3\bin\files\test.gold> on model <E:\SVMTool-1.3\bin\CORPUS>...
*=================TAGGINGSUMMARY======================================

#TOKENS            = 1063
AVERAGE_AMBIGUITY  = 6.4901 tags per token
*  ---------------------------------------------------------------
  #KNOWN  = 80.3387% -->  854 / 1063
  #UNKNOWN = 19.6613% -->  209 / 1063
  #AMBIGUOUS = 21.7310% -->  231 / 1063
  #MFT baseline = 71.2135% -->  757 / 1063

*====================================================================
<table>
<thead>
<tr>
<th>HITS</th>
<th>TRIALS</th>
<th>ACCURACY</th>
</tr>
</thead>
<tbody>
<tr>
<td>816</td>
<td>854</td>
<td>95.5504%</td>
</tr>
</tbody>
</table>

*========known===============================================

*---------known unambiguous tokens -------------------------

| 604  | 623    | 96.9502% |

*---------known ambiguous tokens --------------------------

| 212  | 231    | 91.7749% |

*====================================================================

*=======unknown===============================================

*---------unknown -------------------------------------------

| 186  | 209    | 88.9952% |

*====================================================================

*====================================================================
<table>
<thead>
<tr>
<th>HITS</th>
<th>TRIALS</th>
<th>ACCURACY</th>
<th>MFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1002</td>
<td>1063</td>
<td>94.2615%</td>
<td>71.2135%</td>
</tr>
</tbody>
</table>

*====================================================================

Level of ambiguity

This view of the results groups together all words having the same degree of POS–ambiguity.
*=============================SVMTevalreport
======================================================================
==
EVALUATING <E:\SVMTool-1.3\bin\files\test1.out> vs. <E:\SVMTool-1.3\bin\files\test.gold> on model <E:\SVMTool-1.3\bin\CORPUS>...

*=================TAGGINGSUMMARY======================================

#TOKENS           = 1063
AVERAGE_AMBIGUITY = 6.4901 tags per token

#KNOWN            = 80.3387% -->              854 / 1063
#UNKNOWN          = 19.6613% -->              209 / 1063
#AMBIGUOUS        = 21.7310% -->              231 / 1063
#MFT baseline     = 71.2135% -->              757 / 1063

*=================ACCURACY PER LEVEL OF AMBIGUITY

#CLASSES = 5

LEVEL             HITS           TRIALS    ACCURACY         MFT
---------------------------------------------------------------------
--------------------
1              605              624    96.9551%    96.6346%
2              204              220    92.7273%    66.8182%
3                7                9    77.7778%    66.6667%
4                2                3    66.6667%    33.3333%
28              184              207    88.8889%     0.0000%

*=================OVERALLACCURACY=====================================

HITS           TRIALS          ACCURACY               MFT
---------------------------------------------------------------------
--------------------
1002             1063          94.2615%          71.2135%
### Kind of ambiguity

This view is much finer. Every class of ambiguity is studied separately.

*======================================================================
* model               = [E:\SVMTool-1.3\bin\CORPUS]
* testset (gold)      = [E:\SVMTool-1.3\bin\files\test.gold]
* testset (predicted) = [E:\SVMTool-1.3\bin\files\test.out]
*======================================================================

==
EVALUATING <E:\SVMTool-1.3\bin\files\test.out> vs. <E:\SVMTool-1.3\bin\files\test.gold> on model <E:\SVMTool-1.3\bin\CORPUS>...
*================TAGGINGSUMMARY======================================
#TOKENS           = 1063
AVERAGE_AMBIGUITY = 6.4901 tags per token
======================================================================
#KNOWN            = 80.3387% -->              854 / 1063
#UNKNOWN          = 19.6613% -->              209 / 1063
#AMBIGUOUS        = 21.7310% -->              231 / 1063
#MFT baseline     = 71.2135% -->              757 / 1063
*=================ACCURACY PER CLASS OF AMBIGUITY
======================================================================
#CLASSES = 55

<table>
<thead>
<tr>
<th>CLASS</th>
<th>HITS</th>
<th>TRIALS</th>
<th>ACCURACY</th>
<th>MFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;ADJ&gt;</td>
<td>28</td>
<td>28</td>
<td>100.0000%</td>
<td>100.0000%</td>
</tr>
<tr>
<td>&lt;ADV&gt;_&lt;CNJ&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;COM&gt;_&lt;CRD&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;CRD&gt;_&lt;CVB&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;DET&gt;_&lt;ECH&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;INT&gt;_&lt;NN&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;NNC&gt;_&lt;NNP&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| _<NNPC>_<NNQ>_<ORD>_<PPO>_<PRID>_<PRIN>_<PRP>_<QTF>_<QW>_<RDW>_<VAX>_<}
<table>
<thead>
<tr>
<th>Tag</th>
<th>Count1</th>
<th>Count2</th>
<th>Percent1</th>
<th>Percent2</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;ADJ&gt;_&lt;NN&gt;</td>
<td>1</td>
<td>1</td>
<td>100.000%</td>
<td>100.000%</td>
</tr>
<tr>
<td>&lt;ADJ&gt;_&lt;VNAJ&gt;</td>
<td>1</td>
<td>1</td>
<td>100.000%</td>
<td>100.000%</td>
</tr>
<tr>
<td>&lt;ADV&gt;</td>
<td>31</td>
<td>32</td>
<td>96.875%</td>
<td>96.875%</td>
</tr>
<tr>
<td>&lt;ADV&gt;_&lt;NN&gt;</td>
<td>1</td>
<td>2</td>
<td>50.000%</td>
<td>50.000%</td>
</tr>
<tr>
<td>&lt;ADV&gt;_&lt;RDW&gt;</td>
<td>2</td>
<td>2</td>
<td>100.000%</td>
<td>100.000%</td>
</tr>
<tr>
<td>&lt;CNJ&gt;</td>
<td>20</td>
<td>20</td>
<td>100.000%</td>
<td>100.000%</td>
</tr>
<tr>
<td>&lt;CNJ&gt;_&lt;PPO&gt;</td>
<td>1</td>
<td>1</td>
<td>100.000%</td>
<td>100.000%</td>
</tr>
<tr>
<td>&lt;COM&gt;</td>
<td>17</td>
<td>17</td>
<td>100.000%</td>
<td>100.000%</td>
</tr>
<tr>
<td>&lt;COMM&gt;</td>
<td>49</td>
<td>49</td>
<td>100.000%</td>
<td>100.000%</td>
</tr>
<tr>
<td>&lt;CRD&gt;</td>
<td>23</td>
<td>23</td>
<td>100.000%</td>
<td>95.6522%</td>
</tr>
<tr>
<td>&lt;CRD&gt;_&lt;DET&gt;</td>
<td>22</td>
<td>22</td>
<td>100.000%</td>
<td>100.000%</td>
</tr>
<tr>
<td>&lt;CRD&gt;_&lt;NN&gt;</td>
<td>2</td>
<td>2</td>
<td>100.000%</td>
<td>50.000%</td>
</tr>
<tr>
<td>&lt;CRD&gt;_&lt;ORD&gt;</td>
<td>2</td>
<td>2</td>
<td>100.000%</td>
<td>0.0000%</td>
</tr>
<tr>
<td>&lt;CNJ&gt;</td>
<td>20</td>
<td>20</td>
<td>100.000%</td>
<td>100.000%</td>
</tr>
<tr>
<td>&lt;CNJ&gt;_&lt;PPO&gt;</td>
<td>1</td>
<td>2</td>
<td>100.000%</td>
<td>50.000%</td>
</tr>
<tr>
<td>&lt;CNJ&gt;_&lt;PPO&gt;</td>
<td>1</td>
<td>1</td>
<td>100.000%</td>
<td>100.000%</td>
</tr>
<tr>
<td>&lt;NN&gt;</td>
<td>81</td>
<td>91</td>
<td>89.0110%</td>
<td>89.0110%</td>
</tr>
<tr>
<td>&lt;NN&gt;_&lt;NNC&gt;</td>
<td>148</td>
<td>161</td>
<td>91.9255%</td>
<td>58.3851%</td>
</tr>
<tr>
<td>&lt;NN&gt;<em>&lt;NNC&gt;</em>&lt;NNPC&gt;</td>
<td>1</td>
<td>1</td>
<td>100.000%</td>
<td>100.000%</td>
</tr>
<tr>
<td>&lt;NN&gt;<em>&lt;NNC&gt;</em>&lt;NNPC&gt;_&lt;NNPC&gt;</td>
<td>1</td>
<td>1</td>
<td>100.000%</td>
<td>100.000%</td>
</tr>
<tr>
<td>&lt;NN&gt;_&lt;PRD&gt;</td>
<td>1</td>
<td>1</td>
<td>100.000%</td>
<td>100.000%</td>
</tr>
<tr>
<td>&lt;NN&gt;<em>&lt;PRD&gt;</em>&lt;PRIN&gt;_&lt;VNAJ&gt;</td>
<td>0</td>
<td>1</td>
<td>0.0000%</td>
<td>0.0000%</td>
</tr>
<tr>
<td>&lt;NN&gt;_&lt;VBG&gt;</td>
<td>1</td>
<td>1</td>
<td>100.000%</td>
<td>100.000%</td>
</tr>
<tr>
<td>&lt;NN&gt;_&lt;VF&gt;</td>
<td>0</td>
<td>1</td>
<td>0.0000%</td>
<td>0.0000%</td>
</tr>
<tr>
<td>&lt;NN&gt;_&lt;NNC&gt;</td>
<td>148</td>
<td>161</td>
<td>91.9255%</td>
<td>58.3851%</td>
</tr>
<tr>
<td>&lt;NN&gt;_&lt;NNN&gt;</td>
<td>4</td>
<td>5</td>
<td>80.000%</td>
<td>80.000%</td>
</tr>
<tr>
<td>&lt;NN&gt;_&lt;VAX&gt;</td>
<td>1</td>
<td>1</td>
<td>100.000%</td>
<td>100.000%</td>
</tr>
<tr>
<td>&lt;NN&gt;<em>&lt;VF&gt;</em>&lt;VNAJ&gt;</td>
<td>1</td>
<td>1</td>
<td>100.000%</td>
<td>0.0000%</td>
</tr>
<tr>
<td>&lt;NNP&gt;</td>
<td>34</td>
<td>37</td>
<td>91.8919%</td>
<td>91.8919%</td>
</tr>
<tr>
<td>&lt;NNP&gt;_&lt;NNPC&gt;</td>
<td>0</td>
<td>1</td>
<td>0.0000%</td>
<td>0.0000%</td>
</tr>
<tr>
<td>&lt;NNPC&gt;</td>
<td>0</td>
<td>1</td>
<td>0.0000%</td>
<td>0.0000%</td>
</tr>
<tr>
<td>&lt;NNQ&gt;</td>
<td>4</td>
<td>4</td>
<td>100.000%</td>
<td>100.000%</td>
</tr>
<tr>
<td>&lt;ORD&gt;</td>
<td>3</td>
<td>3</td>
<td>100.000%</td>
<td>66.6667%</td>
</tr>
<tr>
<td>&lt;PPO&gt;</td>
<td>6</td>
<td>6</td>
<td>100.000%</td>
<td>100.000%</td>
</tr>
<tr>
<td>&lt;PPO&gt;_&lt;VNAV&gt;</td>
<td>1</td>
<td>1</td>
<td>100.000%</td>
<td>100.000%</td>
</tr>
<tr>
<td>&lt;PRID&gt;</td>
<td>2</td>
<td>2</td>
<td>100.000%</td>
<td>100.000%</td>
</tr>
</tbody>
</table>
Every class is studied individually.

*===================================SVMevalreport====================================

EVALUATING <E:\SVMTool-1.3\bin\files\test.out> vs. <E:\SVMTool-1.3\bin\files\test.gold> on model <E:\SVMTool-1.3\bin\CORPUS>...

*===================================TAGGINGSUMMARY==================================

158
#TOKENS           = 1063
AVERAGE_AMBIGUITY = 6.4901 tags per token
* ---------------------------
#KNOWN            = 80.3387% -->              854 / 1063
#UNKNOWN          = 19.6613% -->              209 / 1063
#AMBIGUOUS        = 21.7310% -->              231 / 1063
#MFT baseline     = 71.2135% -->              757 / 1063
*----------------------- ACCURACY PER PART-OF-SPEECH
*---------------------------------------------------------------
POS           HITS           TRIALS    ACCURACY         MFT
* -----------------------------------------------
<ADJ>           30               31    96.7742%    90.3226%
<ADV>           47               48    97.9167%    70.8333%
<CNJ>           21               21   100.0000%    95.2381%
<COM>            17               17   100.0000%   100.0000%
<COMM>           49               49   100.0000%   100.0000%
<CRD>            26               26   100.0000%    84.6154%
<CVB>             7                8    87.5000%    75.0000%
<DET>            36               36   100.0000%   100.0000%
<DOT>            77               77   100.0000%   100.0000%
<EMP>             1                1   100.0000%   100.0000%
<INT>             6                7    85.7143%    85.7143%
<NN>            243              259    93.8224%    57.9151%
<NNC>           145              162    89.5062%    46.2963%
<NNP>            43               44    97.7273%    86.3636%
<NNPC>            0               16     0.0000%     0.0000%
<NNQ>             4                4   100.0000%   100.0000%
<ORD>             2                2   100.0000%   100.0000%
<PPO>             9                9   100.0000%   100.0000%
<PRID>            2                3    66.6667%    66.6667%
<PRIN>            2                2   100.0000%   100.0000%
<PRP>            34               34   100.0000%   97.0588%
<QM>             4                4   100.0000%   100.0000%
<QTF>             5                5   100.0000%   100.0000%
<QW>             6                6   100.0000%   66.6667%
RESULTS AND COMPARISON WITH OTHER TOOLS

Apart from SVMTool, three other taggers namely TnT [39], MBT [60] and WEKA [168] were trained with the same corpus. The accuracy result of SVMTool is compared with the above tools for the same testing corpus. Following is brief description of the above mentioned taggers.

*TnT* (Trigrams'n'Tags) is a very efficient statistical part-of-speech tagger that is trainable on different languages and virtually for any tagset. The tagger is an implementation of the Viterbi algorithm for second orders Markov Models. The component for parameter generation trains on tagged corpora. The system incorporates several methods of smoothing and of handling unknown words [39].

*MBT* (Memory Based Tagger) is an approach to POS tagging based on Memory-based learning. It is an extension of the classical k-Nearest Neighbor (k-NN) approach to statistical pattern classification. Here, all the instances are fully stored in memory and classification involves a pass along all stored instances. The approach is based on the assumption that reasoning is based on direct reuse of stored experiences rather than on the application of knowledge (such as rules or decision trees) abstracted from experience. Hence the tagging accuracy for unknown words is low [60].
**WEKA** is a collection of machine learning algorithms for solving real-world data mining problems. The J48 classifier was used for implementation of Tamil POS tagging. All the three tools were trained using the same corpus as used in SVMTool. The same data format was followed in all the cases [168].

The experiments were conducted with our tagged corpus. The corpus was divided into training set and test set. For SVMTool, 94.6% overall accuracy is obtained, which is much higher than that of the other taggers. POS Tagging using MBT gave a very low accuracy (65.65%) for unknown words since the algorithm is based on direct reuse of stored experiences. Ambiguous words were handled poorly by TnT, whereas WEKA gave a high accuracy of 90.11% for ambiguous words (Table 5.7). Though SVMTool gave very high accuracy for all cases, the training time was significantly higher when compared to other tools. The unknown word accuracy of the SVMTool is 86.25%. The accuracy goes down in case of some specific tags. Accuracy results of SVMTool compared to the various tools for the same corpus is given in Table 5.7.

<table>
<thead>
<tr>
<th></th>
<th>WEKA</th>
<th>MBT</th>
<th>TNT</th>
<th>SVMTool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Known</td>
<td>………</td>
<td>88.45%</td>
<td>92.54%</td>
<td>96.74%</td>
</tr>
<tr>
<td>Ambiguous</td>
<td>90.11%</td>
<td>80.23%</td>
<td>78.72%</td>
<td>94.57%</td>
</tr>
<tr>
<td>Unknown</td>
<td>………</td>
<td>65.65%</td>
<td>74.18%</td>
<td>86.25%</td>
</tr>
<tr>
<td>Overall Accuracy</td>
<td>………</td>
<td>78.48%</td>
<td>89.56%</td>
<td>94.6%</td>
</tr>
</tbody>
</table>

### 5.7 ERROR ANALYSIS

The detailed error analysis is conducted to identify the miscalculation of tags. The untagged sentences size of about 1200 sentences (10 k words) is taken for testing the system. For analyzing the error, 8 frequently error occurred tags are considered. The tags and their trials and errors are shown in Table 5.8. For instance, errors represents the tagger is failed to identify the CRD tag at 30 occurrences.

Table 5.9 shows the confusion matrix for 8 POS tags. This matrix shows the performance of the tagger.
Table 5.8 Trials and Error

<table>
<thead>
<tr>
<th>Tags</th>
<th>Trails</th>
<th>Hits</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRD</td>
<td>642</td>
<td>612</td>
<td>30</td>
</tr>
<tr>
<td>NN</td>
<td>4200</td>
<td>3989</td>
<td>211</td>
</tr>
<tr>
<td>NNC</td>
<td>2317</td>
<td>2264</td>
<td>53</td>
</tr>
<tr>
<td>NNP</td>
<td>1768</td>
<td>1721</td>
<td>47</td>
</tr>
<tr>
<td>NNPC</td>
<td>47</td>
<td>33</td>
<td>14</td>
</tr>
<tr>
<td>ORD</td>
<td>32</td>
<td>22</td>
<td>10</td>
</tr>
<tr>
<td>VBG</td>
<td>274</td>
<td>258</td>
<td>16</td>
</tr>
<tr>
<td>VNAJ</td>
<td>682</td>
<td>662</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 5.9 Confusion Matrix

<table>
<thead>
<tr>
<th></th>
<th>CRD</th>
<th>NN</th>
<th>NNC</th>
<th>NNP</th>
<th>NNPC</th>
<th>ORD</th>
<th>VBG</th>
<th>VNAJ</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRD</td>
<td>0.953</td>
<td>0.019</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.028</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NN</td>
<td>0</td>
<td>0.95</td>
<td>0.016</td>
<td>0.003</td>
<td>0.001</td>
<td>0.001</td>
<td>0.01</td>
<td>0.003</td>
<td>0.016</td>
</tr>
<tr>
<td>NNC</td>
<td>0.001</td>
<td>0.018</td>
<td>0.977</td>
<td>0.002</td>
<td>0.001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.001</td>
</tr>
<tr>
<td>NNP</td>
<td>0.001</td>
<td>0.013</td>
<td>0.001</td>
<td>0.973</td>
<td>0.007</td>
<td>0</td>
<td>0</td>
<td>0.001</td>
<td>0.004</td>
</tr>
<tr>
<td>NNPC</td>
<td>0</td>
<td>0.106</td>
<td>0.106</td>
<td>0.085</td>
<td>0.702</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ORD</td>
<td>0.125</td>
<td>0.063</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.688</td>
<td>0</td>
<td>0</td>
<td>0.125</td>
</tr>
<tr>
<td>VBG</td>
<td>0</td>
<td>0.04</td>
<td>0</td>
<td>0.004</td>
<td>0</td>
<td>0</td>
<td>0.942</td>
<td>0</td>
<td>0.015</td>
</tr>
<tr>
<td>VNAJ</td>
<td>0.001</td>
<td>0.004</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.007</td>
<td>0.971</td>
<td>0.016</td>
</tr>
</tbody>
</table>

5.8 SUMMARY

This chapter gave the detail about the development of POS tagger and tagged corpora. Part of Speech tagging plays an important role in various speech and language processing applications. Currently, many statistical tools are available to do Part of Speech tagging. The SVMTool has been already successfully applied to English and Spanish POS Tagging, exhibiting state–of–the–art performance (97.16% and 96.89%, respectively). In both cases, results clearly outperform the HMM–based TnT part–of–speech tagger. For Tamil, an accuracy of 94.6% has been obtained. Any language can be trained easily using the existing statistical tagger tools. POS tagging can be extended by applying this to other languages. The obstacle for the POS tagging for Indian languages is there is no annotated (tagged) corpus. 45k sentences (5 lakh words) POS annotated sentences are developed for train the POS Tagger.