HMM based POS Tagger for a Relatively Free Word Order Language

Arulmozhi Palanisamy and Sobha Lalitha Devi

AU-KBC Research Centre, MIT Campus,
Chromepet, Chennai - 600044, India.
{p.arulmozhi,sobha}@au-kbc.org

Abstract. We present an implementation of a part-of-speech tagger based on hidden markov model for Tamil, a relatively free word order, morphologically productive and agglutinative language. In HMM we assume that probability of an item in a sequence depends on its immediate predecessor. That is the tag for the current word depends up on the previous word and its tag. Here in the state sequence the tags are considered as states and the transition from one state to another state has a transition probability. The emission probability is the probability of observing a symbol in a particular state. In achieving this, we use viterbi algorithm. The basic tag set including the inflection is 53. Tamil being an agglutinative language, each word has different combinations of tags. Compound words are also used very often. So, the tagset increases to 350, as the combinations become high. The training corpus is tagged with the combination of basic tags and tags for inflection of the word. The evaluation gives encouraging result.

1 Introduction

Part of speech (POS) tagging is the task of labeling each word in a sentence with its appropriate syntactic category called part of speech. It is an essential task for all the language processing activities. There are two factors determining the syntactic category of a word (a) the words lexical probability (e.g. without context, *bank* is more probably a noun than a verb) (b) the words contextual probability (e.g. after a noun or a pronoun, bank is more a verb than a noun, as in "I bank with HSBC "). Hence it disambiguates the part of speech of a word when it occurs in different contexts. For any POS work the tagset of the language has to be developed. It should contain the major tags and the morpho-syntactic features called the sub tags.

In this paper we present a POS tagger developed for Tamil using HMM and Viterbi transition. Tamil is a South Dravidian language, which is relatively free word order, morphologically productive and agglutinative in nature. It is an old language and most of the words are built up from roots by following certain patterns and adding suffixes. Due to the agglutination many combination of words and suffixes occur to form a single word whose POS is a combination of all the suffixed words or suffixes. Hence the total number of tagset increases as the combination increases. In this work we confine to a fixed set of tagset, which gives the most commonly needed tags for any

© *A. Gelbukh (Editor)*
Advances in Natural Language Processing
Research in Computing Science 18, 2006, pp. 37-48

computational purposes. The paper's schema is as follows. The second section discusses about the different techniques used for POS tagging. Third section explains the tagset and the architecture of the system developed and the fourth section discusses the implementation details. The results are discussed in the next section (5) and the sixth section concludes the paper.

2 Tagging Techniques

Many techniques have been used for developing POS taggers. The different techniques are (1) Rule-based (2) Transformation Based and (3) statistical based. Each technique has its own merits and demerits.

The first technique to be developed was the rule-based tagger. These taggers used context sensitive rules in getting the correct tag. Among the rule-based approaches the one that achieved the best accuracy is 97.5%[1]. These taggers are based on a defined set of hand written rules. Most of the existing Rule Based POS taggers are based on two-stage architecture. The first stage assigns a list of probable tags (or the basic tag) for a particular word. The second stage uses the list of disambiguation rules, to reduce the list (or change a wrong tag) to a single right tag. The Brill's tagger initially tags a sentence by assigning each word its most likely tag, and then that is tagged according to context information at the second stage [2]. Here all the rules are pre-defined. The rules can be broadly classified in to two (1) Lexical Rules and (2) Context Sensitive Rules. The lexical rules act at the word level. The context sensitive rules act at the sentence level.

The rule-based taggers are developed for European languages. In Indian languages a rule-based POS tagger for Tamil was developed and tested [3]. It uses a suffix stripper to get the root word and no dictionary is used. The tagger tags the major tags (N, V, etc) of the root word. For Example, The word patikkira:n "reading+m+sng+3p" can be split in to pati "Read" V + kkir Pres + a:n Sng.M.3P. The tag for the root word pati cannot be identified without a dictionary. But from the suffixes it takes it could be identified whether it is a noun or a verb.. Here in this example, using the tense marker and GNP marker, the root word can be assumed as a verb. This is done in the word level. If there is any ambiguity, the word is left unknown. And it is resolved in the next step – context level. There are nearly 90 lexical rules and nearly 7 context sensitive rules, which are hand crafted. The precision of the system is 92%. This system gives only the major tags and the sub tags are overlooked while evaluation. Tagging becomes difficult when there is no inflection in the word. When the sub tags are considered the performance is reduced.

Transformation Based Learning method is an approach in which the rule-based and the stochastic methods are combined and used. Like the rule-based taggers, this is based on rules, which say what tag to be assigned to what words. And like the stochastic taggers, this is a (supervised) machine learning technique, in which the rules are automatically derived from the (pre tagged) training data. And those rules are applied to the test corpus. There are various algorithms, which are used for learning purpose. The most commonly used Brill's tagger [4, 5] (for English) is a TBL based tagger. Brill's algorithm has three major stages. In first stage, it labels

every word with the most likely tag. In second stage, it examines all the possible transformations and selects the one, which gives the most improved tagging. The stage two requires that the TBL algorithm knows the correct tag for each word. That is the TBL is a supervised learning algorithm [1]. The output of a TBL process is a list of transformations. The transformation consists of two parts: A triggering environment and a rewrite rule. Then the third stage - it re-tags the data according to the rule. These three steps are repeated till some threshold condition reaches. The accuracy of the tagger depends upon the training corpus and the set of rules. Apart from Brill's tagger, the fn-TBL [6] toolkit and mu-tbl [7] tool uses TBL for tagging English sentences. These are trained for English and also for other European languages.

The POS taggers are developed using statistical method. This is based on probability measures. Generally this needs a tagged corpus for training. The training corpus is a tagged corpus, which is assumed to be 100% correct. The probabilities are calculated using unigram, bigram, trigram, and n-gram methods [8]. Unigram Probability is the probability of a word to occur in a corpus. The Bigram model calculates the probability of a word, given the previous word. By definition, the bigram model approximates the probability of a word given all the previous words, by the conditional probability of the preceding word. Here an assumption is made that the probability of a word depends only on the previous word. This is called Markov assumption. Bigram model is called the first order Markov model, trigram is called the second order Markov model and an N-gram model is called the N-1th Markov model. In developing a POS tagger, the tags are called states in the Markov model and the the words are called the symbols. When N=2, this is generally called a Hidden Markov Model as the information about the previous states are hidden. Viterbi algorithm is a dynamic programming algorithm for finding the most likely sequence of hidden states, known as the Viterbi path.

For many morphologically rich languages like Japanese and Arabic, viterbi algorithm is used for tagging and disambiguation [9, 10]. The Arabic POS tagger achieved an accuracy of around 90% when disambiguating ambiguous words. But unknown words are tagged as nouns because the tagged training corpus had 67% of nouns [10]. Also for languages like Chinese and Japanese the word segmentation itself becomes a problem because of the nature of the language. Still Chinese part of-speech tagging using a first-order fully connected hidden Markov model gives promising results [11].

Regardless of whether one is using HMMs, maximum entropy conditional sequence models, or other techniques like decision trees, most systems work in one direction through the sequence (normally left to right, but occasionally right to left). Most of the well-known and most successful approaches are unidirectional [12].

2.1 Tamil Tagset

As Tamil is an agglutinative language, each word has different combinations of tags. Here we have 17 basic tags and 31 sub tags. Compound words are very common in this language. So the word will have the combination of basic tags and tags for inflection. Thus we have 350 unique tagset for the training data. The nouns take the

case markers and plural markers and verbs take perfect, imperative, person number, gender and tense markings. Other than this, pronouns have person number and gender and also the clitics which are suffixed to the root word to form adverbs, conjunctions etc. The dates, numbers and punctuations are also tagged separately. The tags we use are not monadic tags but structured tags such as <N+Pl+ACC>. This representation is required for further syntactic analysis because different pieces of information in the tag serve different roles in the syntactic representation. The representation is given below:

avan	*kataikku*	*cenRu*	*maruntukaLai*	*va:nkina:n*
He	shop (DAT)	go (VBP)	medicine (PL+ACC)	buy (PST+3SM)

avan/PR+3SM *kataikku*/N+DAT cenRu/V+VBP *maruntukaLai*/N+PL+ACC va:nkina:n/ V+PST+3SM

(He went to the shop and bought medicines.)

The main tags and the sub tags, which can occur with the main tag, are given in Tables 1 and 2.

Table 1. Main tags.

	Main Tag	Representation
1	Noun	N
2	Verb	V
3	Adjective	ADJ
4	Adverb	ADV
5	Verbal Participle	VBP
6	Relative Participle	AJP
7	Quantifier	Q
8	Conjunction	CNJ
9	Punctuation	PNC
10	Indeclinable	IND
11	Interrogative	INT
12	Ordinal	ORD
13	Pronoun	PR
14	Verbal Noun	VBN
15	Determiner	DET
16	Symbol	SYM
17	Unknown	UNK

2.2 Agglutinating Nature of the Language

In this section, we give a detailed description of the agglutinating nature of Tamil, and how difficult is to analyze the inflected/compound words. The grammatical properties of Tamil comprise both morphological and syntactic categories. Generally the main parts of speech in modern Tamil can be distinguished into three morphological word classes as Nouns, Verbs and Uninflected words.

Table 2. Sub tags

	Sub tag	Representation		Sub tag	Representation
1	2nd person, Plural, Neuter gender	2PN	12	Clitic	CLT
			13	Compound	COMP
2	2nd person, Singular, Honorific	2SH	14	Conditional	CND
			15	Dative	DAT
3	2nd person, Singular, Masculine	2SM	16	Emphatic	EMP
			17	Future Tense	FUT
4	2nd person, Singular, Neuter gender	2SN	18	Genitive	GEN
			19	Hortative	HRT
5	3rd person, Plural, Neuter gender	3PN	20	Imperative	IMP
			21	Inclusive	INC
6	3rd person, Singular, Feminine	3SF	22	Infinitive	INF
			23	Instrumental	INS
7	3rd person, Singular, Honorific	3SH	24	Locative	LOC
			25	Negative	NEG
8	3rd person, Singular, Masculine	3SM	26	Past Tense	PST
			27	Plural	PL
9	3rd person, Singular, Neuter Gender	3SN	28	Post Position	PSP
			29	Present Tense	PRE
10	Ablative	ABL	30	Representative	REP
11	Accusative	ACC	31	Sociative	SOC

Words in Tamil can be segmentized in to a sequence of parts called morphemes. Thus Tamil morphology can be characterized as agglutinating or concatenative [13]. This can be represented as follows.

$$\text{Stem } (+ \text{ affix})^n$$

Where the superscript n means one or more occurrences of suffixes. When morphs are serialized as suffixes at the right end of a word stem, inflected or derived words are formed. For example, an inflected noun consists of a noun stem followed by a plural suffix and/or a case suffix:

"*vItukaLil*" = *vItu* + *kaL* + *il* = house + PL + LOC = " in the houses ".

An inflected verb consists of a verb stem followed by a tense and PNG marker.

"*vantAn*" = *va* + *nt* + *An* = come + PST + 3SM = " he came ".

Apart from the inflection of stems, there can be oblique stems occurring with case suffixes, postpositions etc. when words are concatenating, to form a single compound word, almost all the combinations are valid. The most common rules for concatenation are, Noun stem + head noun, oblique stem + case suffix, Noun + Verb, Verb + Verb, Determiner + Noun, Adverb + Verb, Derived noun, Derived verb, {Noun, Verb, Adjective, Adverb} + Postposition, verbal participle+ verb, Adjectival participle + Postposition, Adverb + Verb + Postposition etc. Few examples are given below:

Rule 1: **Noun stem + head noun**
talaimuti = *talai* + *muti* = head hair = "hair of the head"

Rule 2: **oblique stem + case suffix**
enakku = *nAn (obl)* + *ku* = I (obl) + dat = " to me"

Rule 3: **Noun + Verb**
kaivaikAte = *kai* + *vai*+*Ate* = hand + put + (neg) = "do not put (your) hand"

Rule 4: **Verb + Verb**
cAppitamutiyum = *cAppita* + *mutiyum* = eat + can do = "can eat"

Depending on the position of suffixes, when it is added to words, same suffix may give different meaning at different places. For example, the suffix "um" when added to a noun, it is an inclusive marker. When it is added to verb, that indicates third person singular neuter gender. Another example would be postpositions. They can stand-alone, add as a suffix of a word or it can occur between two words as a glue and make a compound word. Few examples are given below

"vantapintAnenRu" = vanta + pin + tAn + enRu = "only after (somebody) came" = AJP + PSP + EMP + IND

"kARRuvazhicceyti" = kARRu + vazhi + ceyti = "news by word of mouth" = N + PSP + N

"enakkumun" = enakku + mun = "before me" = N(DAT) + PSP

"polave" = pola + e = "like that" = PSP + EMP

Thus, the concatenative nature of the language makes the POS tagging task more difficult. There are number of words which have the same spelling and act as different part of speech in a sentence.

Example:

iru – acts as a quantifier (two) and as a verb (be there)
pitikkum – acts as an RP and as a verb.
moodi – acts as a noun and verb.

Tamil is a relatively free word order language where the constituents in a sentence could be rearranged in all possible combinations without altering the meaning. The following example shows this nature of Tamil.

Example:

"He hit the ball with a bat" can be written in different ways,

a. *avan pantai mattaiAl atittAn*
b. *pantai avan mattaiAl atittAn*
c. *avan mattaiAl pantai atittAn*
d. *mattaiAl avan pantai atittAn*
e. * *avan atittAn pantai mattaiAl*

The sentence "e" is not a grammatical sentence in Tamil, since it is not verb ending. Handling of the tagging process for Tamil is explained in the next section.

3 The POS Tagger

The system has two major modules, where the first module is for building the language model and the second for finding out the viterbi tag sequence for a given sentence. The training corpus is assumed to be 100% correct. The training corpus consists of 25000 tagged words, which are tagged by a rule-based tagger and then manually checked and corrected.

The training corpus is given to the training module and the initial matrices for the viterbi algorithm are built. There are 3 matrices built by this module – initial state probability matrix (π), transition probability matrix (A) and emission probability matrix (B). Then a language model (λ) is built using the formula,

$$\lambda = (A, B, \pi)$$

The second module is the actual viterbi algorithm, which finds out the most likely tag sequence for a given sentence. The input sentence is given to this module and the output is the tagged sentence. How viterbi algorithm works for tagging a sentence is explained as follows.

Probability of item in sequence depends on its immediate predecessor. That is the tag for the current word depends up on the previous word and its tag. Here in the state sequence the tags are considered as states and the transition from one state to another state has a transition probability. The emission probability is the probability of observing a symbol in a particular state. Here in the POS tagger application words are considered as symbols and the tags are considered as states.

Training of the system using tagged Tamil corpus and using it for tagging a new unseen sentence is explained in section 4. As Tamil is an agglutinative language, each word may have different combinations of tags. Compound words are also used very often. The training corpus is tagged with the basic tag and the tags for the inflection of the word. Implementation of the modules and tagging a new corpus using the system are explained in detail in the following sub sections.

3.1 System Implementation

The system is implemented using the concept of HMM and Viterbi transitions. Hidden Markov Model (HMM) is a statistical method, which can be used for Natural Language Processing Tasks like Parts of Speech tagger, Information Extraction etc. Using HMM an attempt is made to build a Language model, such that the system (or the computer) can be trained to work similar to humans for processing natural languages like English, Tamil etc.

There are some basic assumptions made when HMM is modelled for natural language texts. First assumption is that Language is a Markov Process with unknown parameters or hidden states and we have to determine the hidden parameters with the use of observable output parameters. In a Markov model there are no hidden parameters all the parameters are observable.

There are 3 canonical problems to solve with HMMs:

1) Given the model parameters, compute the probability of a particular output sequence. Solved by the *forward algorithm.*
2) Given the model parameters, find the most likely sequence of (hidden) states which could have generated a given output sequence. Solved by the *Viterbi algorithm.*
3) Given an output sequence, find the most likely set of state transition and output probabilities. Solved by the *Baum-Welch algorithm.*

In constructing HMM for language model, we have to determine what are the states or parameters in the model and what they correspond to and how many total numbers of states are possible in the model. Prior to building of HMM, a state table has to be constructed. Once the state table is prepared and annotated training text is available we can build HMM model.

The HMM model (λ) is given by

$$\lambda = (A, B, \pi) \tag{1}$$

Here A, B, and π are the probability measures used for modeling. Let us say there are N, distinct states and states denoted as $S_1, S_2, \ldots, S_N$. And M number of distinct output symbols per state and denoted by $V_1, V_2 \ldots V_M$. If we denote a state at time t as q_t then A is the state transition probability distribution.

$$a_{ij} = P\,[q_{t+1} \mid q_t] \tag{2}$$

B is the observation symbol probability distribution in state j, $B=\{b_j(k)\}$ where

$$b_j(k) = P\,[\,V_k \text{ at } t \mid qt = Sj] \qquad \begin{matrix} 1 < j < N \\ 1 < k < M \end{matrix} \tag{3}$$

π is the initial state distribution where

$$\pi_i = P\,[\,q_1 = S_i\,] \qquad 1 < i < N \tag{4}$$

After calculating the initial state distribution, Viterbi algorithm can be implemented for finding out the hidden sequence of states for a particular input sentence. In this algorithm the following assumptions are made. First, both the observed events and hidden events must be in a sequence. This sequence often corresponds to time. Second, these two sequences need to be aligned, and an observed event needs to correspond to exactly one hidden event. Third, computing the most likely hidden sequence up to a certain point t must depend only on the observed event at point t, and the most likely sequence at point t − 1. These assumptions are all satisfied in a first-order hidden Markov model. What the algorithm does only is it tries to give the best possible solution. The difficulty lies in setting the optimal criteria for choosing the path. There are several possible optimal criteria's, which can be set. In Viterbi algorithm a single best state sequence is given as output. Here we find the path that maximizes the observations given the HMM model.

The formal algorithm consists of 5 steps. First step initializes the first column using the initial state vector and the first observation. Next is the recursion step, which gives the i-th element of the t-th column, which is the probability of the *most likely* way of being in that position, given events at time t–1. The back pointer indicates where one is most likely to have come from at time t–1 if currently in state i at time t. next step is

termination. It indicates what the most likely state is at time T, given the preceding T-1 states and the observations. Fifth step traces the back pointers through the lattice, initializing from the most likely final state. Using backtracking, we find the most likely tag sequence determined by the Viterbi algorithm. How a new sentence assigned the most likely tag sequence is explained below.

Here we assume each tag of POS as a state. The words in the text are assumed to be as observable symbols or tokens. If there is large enough text corpus, which contains of all possible sentence structures of the language then using HMM we can build a very robust language model for this task.

The initial state probability distribution (π) is constructed by finding the ratio of the count of each states occurring in the first position of the sentence to the total number of sentences. For example if state S1 has occurred n times as the first state of sentence in the corpus and if there are X number of sentences in the corpus then initial state probability for state S1 is given as (n/X). We similarly find for all states in the model and an initial state probability distribution is built.

The transition state probability distribution is nothing but the states bigram frequency. Or we can say it is the probability of state Si to transit to state Sj.

The observation symbol probability distribution or simply called emission probability is the probability of an output symbol Vk to occur in given particular state Si. That is probability for a word to have a particular tag.

Once these three (A, B, π) model parameters are found we use Viterbi transitions to find the best path for a given sequence of output symbols (i.e., a test sentence).

The training corpus is tagged by a rule-based system [3] and manually corrected. The tag for each word is a combination of basic tags and the tag for inflection.

Example 1:

the word *payanpatukinRana* - "they are useful" is tagged as "V+PRE+3PN".
That is, this word is formed with, "*payanpatu+kinRu+ana*"
payanpatu "are useful" – Verb (V)
kinRu – Present tense (PRE)
ana – 3rd person, Plural, Neuter Gender (3PN)
The final output of the tagger is V+PRE+3PN

Example 2:

The word *immaruntukaL* "the medicines" - is tagged as "N+COMP"
This compound word is formed with, "i+maruntu+kaL"
i (inta) "this" – Determiner (DET)
maruntu "medicine" – Noun (N)
kaL – Plural (PL)

When the combination of the basic tag increases, the tagset for viterbi increases. This leads to sparse data problem [14]. To avoid this problem, we reduce certain combination of tags to a single tag "COMP".

Using the training corpus, the initial state matrix, transition and emission probability matrices are calculated and the language model is built. There are many paths in an HMM with S states. We try to find the path that maximizes the observations given the HMM model [15].

As the corpus is initially tagged using a rule-based system and corrected, it takes care of the inflection to some level reducing the manual work, though compounding is not dealt in detail.

4 Results and discussion

The system is tested against a data set of 5000 words. The system performs well and gives a high precision and recall. Three different sets of test data are tagged and evaluated. These sets are from different domains given below.

Set 1 – Recipes
Set 2 – Veterinary Diseases
Set 3 – Historical Events

Three different types of data are taken because the constituents in a sentence are different in different domains. In recipes, multiple proper nouns occur continuously, separated by a punctuation mark. They are written in a step-by-step manner and most of the sentences are imperative. Whereas in veterinary diseases, there are considerable number of foreign words, transliterated medicine names etc and the format in which the text is written, mostly resembles textbooks. In historical domain, the paragraphs are highly related to each other and it is written like a story. Here, we come across many classical words from ancient Tamil. There are many compound words. Three sets of data containing 1565, 1810 and 1616 words are taken from the above domains for evaluation and the precision and recall is calculated. The precision for the 3 sets is 98.43% and the recall is 98.43%. The evaluation results are given in Table 1.

Table 3. Evaluation Results

Title	No. Of Words	Words tagged	Correctly Tagged	Precision	Recall
Set 1	1565	1565	1544	98.7%	98.7%
Set 2	1810	1810	1779	98.3%	98.3%
Set 3	1616	1616	1589	98.3%	98.3%

An advantage in using viterbi algorithm is the system is not influenced by small errors. For example,

enna – "what"

Here, the actual tag for this is "INT" but in the manually tagged corpus, it was wrongly tagged as "IND" at one place. But that is not carried to the output. Even if there is a small typological error in the tagged corpus, which changes the tag itself, the machine tagged output is correct. Even if there is an unseen word, the system tries to find out the tag for that word depending on the context. This does not need an exhaustive set of rules. As the training corpus increases, the system becomes robust.

As Tamil is an agglutinative language, each word may have different combinations of tags. Compound words are also used very often. The training corpus is tagged with the combination of basic tags and tags for inflection of the word. So, the tagset

increases to 350, as the combinations become high. The basic tag set including the inflection is 53.

If any unseen word (which does not occur in the training corpus) occurs in the test corpus, the emission probability of the word becomes zero. But when viterbi algorithm is implemented, it tries to assume the tag for the unseen word depending on the previous sequence of words. It explores all possible states for emitting the word. If no tags are occurring after a particular tag, transition probability of that particular tag becomes zero and there are no probable states after that. In that case, the whole sentence is not tagged because the total probability becomes zero. When there is a very small probability especially for special characters, the transition probability becomes insignificant and it becomes zero, leading to the total probability to be zero. When the insignificant output symbols such as ";", "!", etc. occurs in between a sentence, the above-mentioned problem is faced.

Some of the sentences are not tagged because at some point for an unseen word, no transition may be possible from the current state. So, no tags are assigned. Some words are assigned wrong tags, because transition or emission probability is wrongly calculated. This is because some forms of words do not occur in the training corpus and the test corpus has those forms. So, there is a high probability of assigning a wrong tag to those particular words. For example,

a. the word *" eRiya"* which means, "to throw" is tagged as "ADJ" by the machine. But the actual tag for this is "ADV".

b. the word *irukkum* which means "will be there" is tagged as V+FUT+3SN. But the actual tag for this in that particular sentence is "V+AJP".

5 Conclusion

In this paper we have discussed a statistical part of speech tagger for Tamil which is developed as a module in the Tamil- English Machine Translation system. In arriving at the POS tagger we use HMM and Viterbi algorithm for tagging a new corpus. The system performs well with high precision and recall. HMM emission probability for an unknown word is calculated depending on the word order. As Tamil is a relatively free word order language and morphologically rich, there is always a high probability for an unseen word to occur. So the precision may vary according to the training corpus. The system can be made more reliable and robust when trained with a bigger corpus, which contains all types of words and its combinations equally distributed. A high precision morphological analyzer is not needed in HMM; instead a compound word analyzer will improve the performance considerably. Another advantage in this system is even if there is any man made mistake in the training corpus, that is not carried to the output. This POS tagging task for Tamil is useful and it plays a vital role in language processing activities like information extraction, machine translation etc. from Tamil to other languages.

References

1. Eric Brill: Some Advances in transformation Based Part of Speech Tagging, In proceedings of the Twelfth International Conference on Artificial Intelligence (AAAI-94), Seattle, WA, (1994)
2. Eric Brill: A Simple Rule-Based Parts of Speech Tagger, University of Pennsylvania,(1992)
3. Arulmozhi. P, Sobha. L, Kumara Shanmugam. B.: Parts of Speech Tagger for Tamil, Symposium on Indian Morphology, Phonology & Language Engineering, Indian Institute of Technology, Kharagpur (2004), 55-57
4. Eric Brill: Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part of Speech Tagging, Association of Computational Linguistics, (1995)
5. Eric Brill: Transformation-Based Part of Speech Tagger (V 1.14), http://www.cs.jhu.edu/~brill/code.html
6. Radu Florian and Grace Ngai: Fast Transformation-Based Learning Toolkit, http://nlp.cs.jhu.edu/~rflorian/fntbl/ Johns Hopkins University,
7. Torbjörn Lager: The μ-TBL system, Paper presented at the Third International Workshop on Computational Natural Language Learning (CoNLL'99), Bergen, (1999)
8. Eugene Charniak: Statistical Language Learning,. MIT Press, Cambridge, London (1997)
9. Jun'ichi Kazama, Yusuke Miyao and Jun'ichi Tsujii "A Maximum Entropy Tagger with Unsupervised Hidden Markov Models". In Proceedings of the Sixth Natural Language Processing Pacific Rim Symposium NLPRS, (2001), 333—340
10. Shereen KHOJA: APT: Arabic Part-of-speech Tagger, Proceedings of the Student Workshop at the Second Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL2001), Carnegie Mellon University, Pittsburgh, Pennsylvania, (2001)
11. Chao-Huang Chang and Cheng-Der Chen: HMM-based Part-of-Speech Tagging for Chinese Corpora, Industrial Technology Research Institute Chutung, Hsinchu 31015, Taiwan, R.O.C
12. Kristina Toutanova *et al.*: Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network, HLT-NAACL, (2003), 252-259
13. Thomas Lehmann: A Grammar of Modern Tamil, Pondicherry institute of Linguistics and culture, Pondicherry, India, (1993)
14. Wolfgang Lezius: Morphy - German Morphology, Part-of-Speech Tagging and Applications, ; Stefan Evert; Egbert Lehmann and Christian Rohrer, editors, Proceedings of the 9th EURALEX International Congress, (2000), 619-623
15. Trevor Cohen, Franklin Din, Karina Tulipano: Hidden Markov Models – Methods in Biomedical Informatics. A tutorial on Hidden Markov Models and Selected Applications in Speech Recognition", Proceedings of the IEEE, Vol. 77, No 2. (1989)