Spelling Corrector for Indian Languages

K.V.N. Sunitha and A. Sharada

CSE Dept, G.Narayanamma Institute of Technology and Science,
Shaikpet, Hyderabad, India
k.v.n.sunitha@gmail.com,
sharada.nirmal@gmail.com

Abstract. With the advancements in computational linguistic processing, the paper work is replaced by documents in the form of soft copies. Though there are many software and keyboards available to produce such documents, the accuracy is not always acceptable. The chance of getting errors is more. This paper proposes a model which can be used to correct spellings of Indian languages in general and Telugu in particular. This paper also discusses the experimental setup and results of implementation. There are few spell checkers and correctors which were developed earlier. The spelling corrector proposed in this paper is different from that in terms of the approach used. Main claim of the paper is implementation of the correction algorithm and proposal of architecture.

Keywords: Indian Languages, Spelling Corrector, Edit distance, Validation, Samdhi Formation.

1 Introduction

Documents prepared in local language reach large number of people. A lot of research is carried out throughout India over the decade and most of the documents are being prepared in local language through software available for the purpose. Though there are many software's and keyboards available to produce such documents, the accuracy is not always acceptable, the chance of getting errors is more.' particularly for Indian language most of which are highly inflecting.

Indian language has close tie with Sanskrit and is characterized by a rich system of inflectional morphology and a productive system of derivation, saMdhi (conation of full words) and compounding. This means that the number of surface words will be very large and so will be the raw feature space, leading to data scarcity [1].

The main reason for richness in morphology of Indian languages is a significant part of grammar that is handled by syntax in English (and other similar languages) is handled within morphology. Phrases including several words in English would be mapped on to a single word in Telugu.

In highly inflecting language, such as Telugu, there may be thousands of word forms of the same root, which makes the construction of a fixed lexicon for any reasonable coverage hardly feasible. Also in compounding languages, complex concepts can be expressed in a single word, which considerably increases the number of possible word forms.

2 Literature Survey

There are few spell checkers and correctors which were developed earlier [8]. This spelling corrector is different from that in terms of the approach used. Many spell checkers store a list of valid words in the language. A given word is assumed to be free of spelling errors if that word is found in this stored list. But, as discussed in the above section, it is very difficult, if not impossible, to cover all the words of any Indian language, where each word may have thousands and millions of word forms. Such languages have a disadvantage with word-based approaches, thus leading to data scarcity problem in n-gram language modeling.

Factored language models have recently been proposed for incorporating morphological knowledge in the modeling of inflecting language. As suffix and compound words are the cause of the growth of the vocabulary in many languages, a logical idea is to split the words into shorter units [2].

2.1 Our Earlier Work in NLP

Details of our work in NLP can be found in papers [3] [4]. The necessity of designing this architecture can be found in [3] and details of ***notation used for transliteration*** and ***corpus*** on which we are working can be found in paper [4].

3 Proposed Model

The design of Spelling Corrector for Telugu, basically involves the interface architecture, Trie Creation, and Spelling Corrector module architecture. The user need to enter the input provided through the GUI. Then the input is syllabified and analyzed based on the rules to display the root word, inflection and correct word.

3.1 Trie Data Structure

In computer science, a trie, or prefix tree, is an ordered tree data structure that is used to store an associative array where the keys are usually strings. Unlike a binary search tree, no node in the tree stores the key associated with that node; A Trie is a multiway tree structure useful for storing strings over an alphabet.

Trie Creation

The *trie* considered here is different from standard trie in two ways:

1) A standard trie does not allow a word to be prefix of another, but the trie we used here allows a word to be prefix of another word. The node structure and search algorithm also is given according to this new property.

2) Each word in a standard trie ends at an external node, where as in our trie a word may end at either an external node, or the internal node. Irrespective of whether the word ends at internal node or external node, the node stores the index of the associated word in the occurrence list.

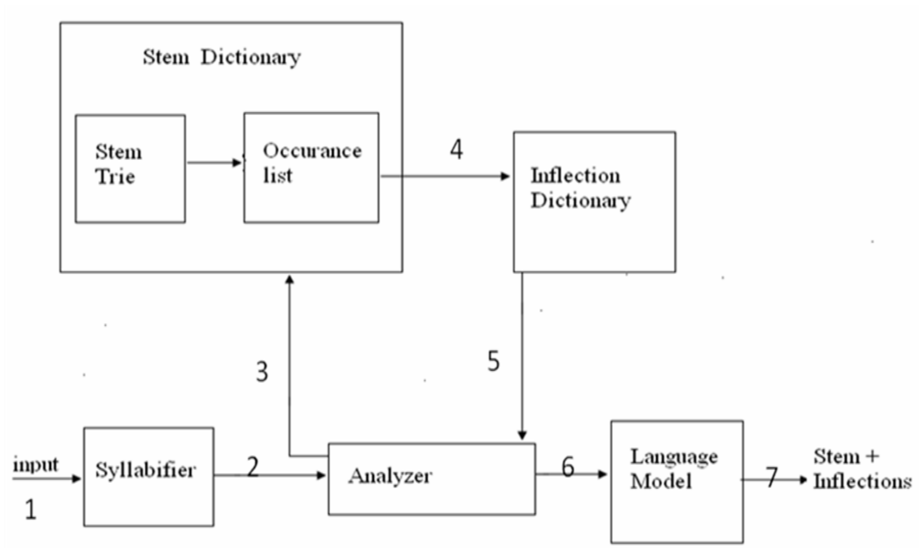


Fig. 1. Search process in a Trie

The node structure is changed such that, each node of the trie is represented by a triplet **<C,R,Ind>**.

C represents character stored at that node.

R represents whether the concatenation of characters from root till that node forms a meaningful stem word. Its value is 1, if characters from root node to that node form a stem, 0 otherwise.

Ind represents index of the occurrence list. Its value depends on the value of R. Its value is -1 (negative 1), if R=0, indicating it is not a valid stem. So no index of occurrence list matches with it. If R=1, its value is index of occurrence list of associated stem.

Advantages Relative to Binary Search Tree

The following are the main advantages of tries over binary search trees (BSTs):

- Looking up keys is faster. Looking up a key of length m takes worst case O(m) time. A BST performs O(log(n)) comparisons of keys, where n is the number of elements in the tree, because lookups depend on the depth of the tree, which is logarithmic in the number of keys if the tree is balanced. Hence in the worst case, a BST takes O(m log n) time. Moreover, in the worst case log(n) will approach m. Also, the simple operations tries use during lookup, such as array indexing using a character, are fast on real machines.
- Tries can require less space when they contain a large number of short strings, because the keys are not stored explicitly and nodes are shared between keys with common initial subsequences.
- Tries facilitating longest-prefix matching, helping to find the key sharing the longest possible prefix of characters all unique.

3.2 Architecture of the Proposed System

3.2.1 Dictionary Design

Dictionary design here involves creating stem words dictionary and inflections dictionary. Main claim of this work is the design of stem word dictionary which is implemented as an **Inverted Index** for better efficiency. The Inverted index will have the following 2 data structures in it:

a) *Occurrence list*, which is an array of pairs, <stem word, list of inflection indexes>
b) Variation of *trie* consisting of stem words

Occurrence list is constructed based on the grammar of the language, where each entry of the list contains the pair <stem word, index list of possible inflections>

3.2.2 Stem Dictionary

A Stem dictionary is maintained, which contains all the root words. As shown in the Table 1, each stem also contains a list of indices of Inflection dictionary indicating inflections possible with that word.

Table 1. Stem Dictionary

Word	Inflection Indices
amma (అమ్మ)	1, 2, 4, 6,7,8
anubhUti (అనుభూతి)	1,7
ataDu (అతడు)	….
Ame (ఆమె)	
AlOcana (ఆలోచన)	
AlApana (ఆలాపన)	
…..	

Table 2. Inflection dictionary

Sl.No	Inflection
1	ki (కి)
2	tO (తో)
3	lO (లో)
4	guriMci (గురించి)
5	lu (లు)
6	yokka (యొక్క)
7	ni (ని)
8	valana (వలన)
….	…….

Inflections Dictionary

All the possible inflections of the language are stored in the Inflection dictionary. Table 2 gives structure of Inflection Dictionary.

4 Spelling Corrector

A spelling corrector detects the spelling errors and gives the correct form of word. When a mis-spelled word is detected, a quantitative measure of the closeness of spelling is used to select the probable words list.

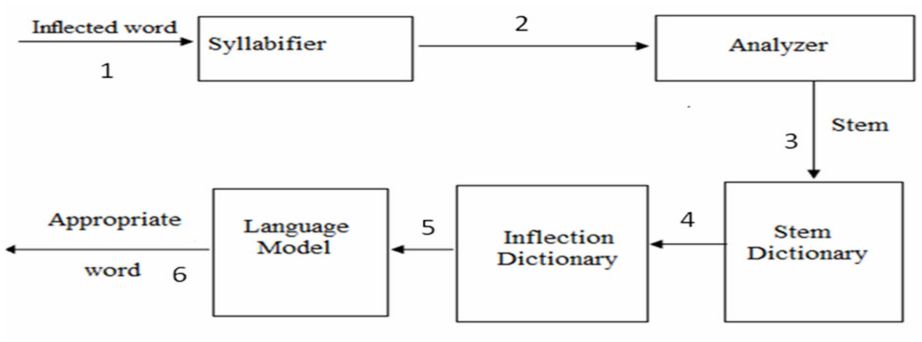


Fig. 2. Block Diagram for Spelling Corrector

4.1 Block Diagram for Spelling Corrector

Input the spelling corrector is an Inflected word. Syllabifier takes this word and divides the word into syllables and identifies if the letter is a vowel or a consonant. After applying the rules syllabified form of the input will be obtained. Once the process of syllabification is done, this will be taken up by the analyzer. Analyzer separates the stem and inflection part of the given word. This stem word will be validated by comparing it with the stem words present in stem dictionary. If the stem word is present, then the inflection of the input word will be compared with the inflections present in inflection dictionary of the given stem word. If both the inflections get matched then it will directly displays the output otherwise it takes the appropriate inflection(s) through comparison and then displays.

4.2 Steps for Spelling Correction

- Receiving the inflected word as an input from the user.
- Syllabifying the input
- Analyzing the input and validating the stem word.
- Identifying the appropriate inflection for the given stem word by comparing the inflection of given word with the inflections present in inflection dictionary of the stem word.
- Displaying the appropriate inflected word.

4.3 Rules for Implementing Spelling Corrector

Telugu is a syllabic language. Similar to most languages of India, each symbol in Telugu script represents a complete syllable. Syllabification is the separation of the words into syllables, where syllables are considered as phonological building blocks of words. It is dividing the word in the way of our pronunciation. The separation is marked by hyphen.

In the morphological analyzer, the main objective is to divide the root word and the inflection. For this, we divide the given input word into syllables and we compare the syllables with the root words and inflections to get the root word and appropriate inflection.

The Roman transliteration format has been considered for the representation of Vowels and Consonants in Telugu in this software.

The user input, the inflected word is taken as the input to the syllabification module and it divides the word into lexemes and decides whether each lexeme is a vowel(V) or a consonant(C), type of the lexeme is stored in a different array and it is processed by applying the Syllabification rules defined below in this section.

For example, considering the word **"nAnnacAriku"** (నాన్నచారికు) which is misspelled form of **"nAnnagariki"** (నాన్నగారికి) meaning **"to father",** the input is given the user in Roman transliteration format. This input is basically divided into lexemes as:

nAnnacAriku

⇩

n – A – n – n – a – c – A – r – i – k – u

⇩

c – v – c – c – v – c – v – c – v – c – v

Now, the array is processed which gives the type of lexeme by applying the rules of syllabification one by one.

- **Applying Rule 1**

" No two vowels come together in Telugu literature."

The given user input does not have two vowels together. Hence this rule is satisfied by the given user input. The output after applying this rule is same as above. If the rule is not satisfied, an error message is displayed that the given input is incorrect. Now the array is:

- **Applying Rule 2**

" Initial and final consonants in a word go with the first and last vowel respectively."

Telugu literature rarely has the words which end up with a consonant. Mostly all the Telugu words end with a vowel. So this rule does not mean the consonant that ends up with the string, but it means the last consonant in string. The application of this rule2 changes the array as following:

c – v – c – c– v – c – v – c – v – c – v

$$\Downarrow$$

cv – c – c – v – c – v – c – v – cv

This generated output is further processed by applying the other rules.

- **Applying Rule 3**

“ **VCV**: The C goes with the right vowel.”

The string wherever has the form of VCV, then this rule is applied by dividing it as V – CV. In the above rule the consonant is combined with the vowel, but here in this rule the consonant is combined with the right vowel and separated from the left vowel. To the output generated by the application of rule2, this rule is applied and the output will be as:

cv – c – c – v – c – v – c – v – cv

$$\Downarrow$$

cv – c – c – v – cv – cv – cv

This output is not yet completely syllabified, one more rule is to be applied which finishes the syllabification of the given user input word.

- **Applying Rule 4**

“ Two or more Cs between Vs - First C goes to the left and the rest to right.”

It is the string which is in the form of VCCC*V, then according to this rule it is split as VC – CC*V. In the above output VCCV in the string can be syllabified as VC – CV. Then the output becomes as:

cv – c – c – v – cv – cv – cv

$$\Downarrow$$

cvc– cv – cv – cv – cv

Now that this output is converted to the respective consonants and vowels. Thus giving the complete syllabified form of the given user input.

nAn – na –cA – ri – ku

$$\Downarrow$$

cvc – cv – cv – cv – cv

Hence, for the given user input, **“nAnnacAriku”,** the generated syllabified form is,

“nAn – na – cA – ri – ku”.

4.4 Spelling Correction Process

Using the rules the possible set of root words are combined with possible set of inflections and the obtained results are compared with the given user input and the nearest possible root word and inflection are displayed if the given input is *correct*.

If the given input is *not correct* then the inflection part of the given input word is compared with the inflections of that particular root word and identifies the nearest possible inflection and combines the root word with those identified inflections, applies sandhi rules and displays the output.

The user input is syllabified and this would be the input to the analyzer module. Matching the syllabified input from starting with the root words stored in dictionary module a possible set of root words is obtained.

For eg: ***nAnnacAriku*** ← user input

⇩ (After syllabification)

nAn-na-cA-ri-ku

Now, scanning the syllabified input from starting and matching it with the set of root words stored in dictionary module.

⟹ ***nAn-na-cA-ri-ku*** (matching from starting)

This process will identify the possible set of root words from the Stem dictionary using the procedure mentioned in [3].

…….
nAnna (నాన్న)
nANemu (నాణెము)
………

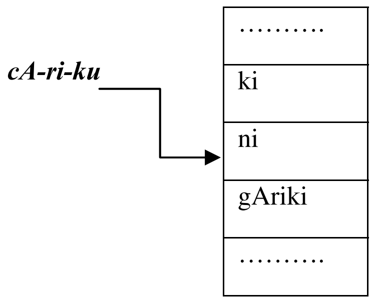
Once possible root words identified the given word is segmented into two parts, first being the root word and second part inflection. Now the inflection part is compared in the reverse direction for a match in the inflection dictionary. It will consider only the inflections that are mentioned against the possible root words, thus reducing the search space and making the algorithm faster.

nAn-na-cA-ri-ku ⟸ (matching from reverse)

Possible set of inflections in inflections dictionary

ki (కి)
ni (ని)
gAriki (గారికి)

After getting the possible set of root words and possible set of inflections they are combined with the help of SaMdhi formation rules. Here in this example ***cA-ri-ku*** is compared with the inflections of the root word ***nAnna***

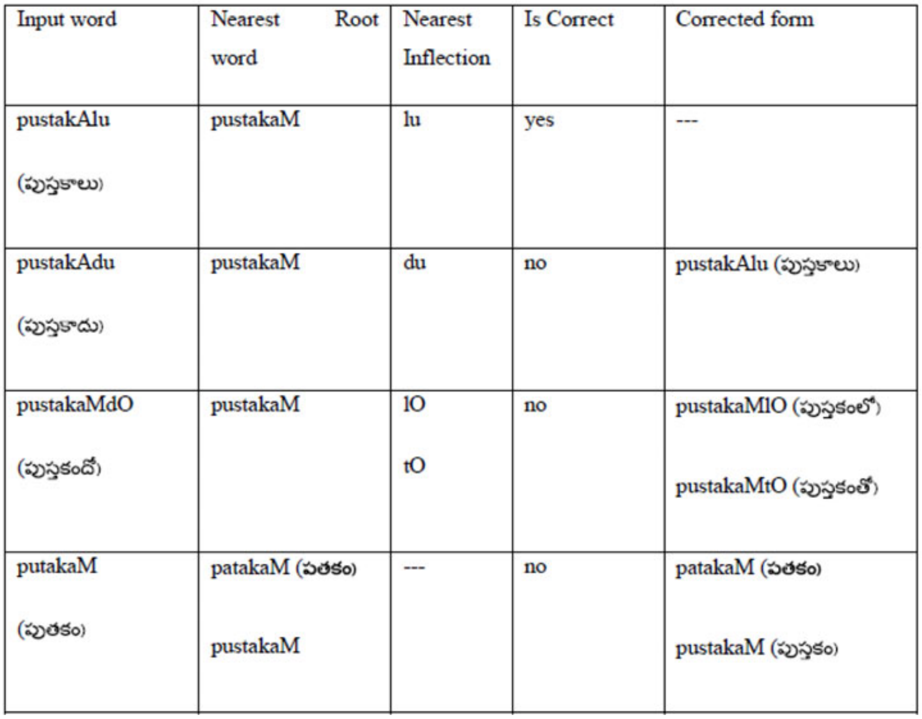


After comparing it identifies ***gAriki*** as the nearest possible inflection and combines the root word with the inflection and displays the output as ***"nAnnagAriki".***

5 Results and Experimental Setup

This model is implemented using PERL. We have used a corpus of 5 Million words, including 2 Million words GNITS Corpus developed in our institute and CIIL corpus.

Table 3. Sample Results



Input word	Nearest Root word	Nearest Inflection	Is Correct	Corrected form
pustakAlu (పుస్తకాలు)	pustakaM	lu	yes	---
pustakAdu (పుస్తకాడు)	pustakaM	du	no	pustakAlu (పుస్తకాలు)
pustakaMdO (పుస్తకందో)	pustakaM	lO tO	no	pustakaMlO (పుస్తకంలో) pustakaMtO (పుస్తకంతో)
putakaM (పుతకం)	patakaM (పతకం) pustakaM	---	no	patakaM (పతకం) pustakaM (పుస్తకం)

When there is more than one root word or more than one inflection has minimum edit distance then the model will display all the possible options. User can choose the correct one from that. E.g., when the given word is ***pustakaMdO*** (పుస్తకండో), the inflections ***tO*** making it ***pustakaMtO*** (పుస్తకంతో) meaning 'with the book' and ***lO*** making it ***pustakaMlO*** (పుస్తకంలో) meaning 'in the book') mis are possible. Present work will list both the words and user is given the option. We are working on improving this by selecting the appropriate word based on the context. Sample results are shown in Table 3.

6 Conclusion

The model proposed is useful for the spelling correction of native language documents and it could be a base work for accelerating the researches on speech recognition software and NLP. This tool will be a big boon for the researchers working on Natural Language Processing for Telugu. This tool also helps in knowing all the words that can be derived from a single root. The tool will help the beginners to learn new words and the specialists to create new terminology.

With the advancements in computational linguistic processing, although the paper work or the manual effort is replaced by documents and soft copies the chance of getting errors is more. Hope our *Spelling Corrector* improves this situation.

References

1. Uma Maheshwara Rao, G.: Morphological complexity of Telugu. In: ICOSAL-2 (2000)
2. Lovins, J.: Development of stemming algorithm. Journal of mechanical translation and computational linguistics 11, 22–31 (1968)
3. Sunitha, K.V.N., Sharada, A.: Building an Efficient Language Model based on Morphology for Telugu ASR. In: KSE-1, CIIL, Mysore (March 2010)
4. Sunitha, K.V.N., Sharada, A.: Telugu Text Corpora Analysis for Creating Speech Database. IJEIT 1(2) (December 2009), ISSN 0975-5292
5. Paice, C., Husk, G.: Another Stemmer. ACM SIGIR Forum 24(3), 566 (1990)
6. Porter, M.F.: An algorithm for suffix stripping. In: Readings in Information Retrieval, pp. 313–316. Morgan Kaufmann Publishers Inc., San Francisco (1997)
7. Xu, J., Croft, W.B.: Corpus based stemming using co-occurrence of word variants. ACM Trans. Inf. Syst. 16(1), 61–81 (1998)
8. Dawson, J.L.: Suffix removal for word conflation. Bulletin of the Association for Literary and Linguistic Computing 2(3), 33–46 (1974)
9. Krovetz, R.: Viewing morphology as an inference process. In: Proceedings of Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 191–203 (1993)
10. vishwabharat@tdil
11. Emerald Group Publishing Ltd., Issues in Indian languages computing in particular reference to search and retrieval in Telugu Language
12. LANGUAGE IN INDIA Strength for Today and Bright Hope for Tomorrow, August 2006, vol. 6 (2006)