CHAPTER 4

ENGLISH TO KANNADATRANSLITERATION

Language transliteration is one of important area in NLP. Machine Transliteration is the conversion of a character or word from one language to another without losing its phonological characteristics. In other word we can say machine translitaration is an orthographical and phonetic convertingprocess. Therefore, both grapheme and phoneme information shouldbe considered. The transliteration model must be designed in such a way that the phonetic structure of words should be preserved as closely as possible. Accurate transliteration of named entities plays an important role in the performance of MT and CLIR processes.

The main purpose of transliteration is to translate the named entities like Person, Location, Affiliation, Organization and Technical terms from source to target language according to the alphabetical system of the target language. The example for Person, Location, Affiliation, Organization and Technical terms is as follows:

Named Entity	:	Example
Person	:	Arathi
Location	:	Mysore
Affiliation	:	Secretary
Organization	:	Indian Public Service
Technical terms	:	modem

The table 4.1 below shows the NE classification with example for each. Location can be split into city, state, country etc. and a Person can be an entertainer, a politician, a scientist etc.

Table 4.1: NE classification with Examples

Named Entity Type	Generic Term	Examples
Person	first, middle and last names of the people, animals and fictional charactersaliases	
Organization	companies	companies press agencies,studios,banks,stock markets.manufacturers,cooperatives
	subdivisions of companies	newsrooms
	brands	-
	political movements	political parties,terrorist organizations
	government bodies	ministries,councils,courts,political unions of countries(e.g.the U.N)
	publications	Magazines,newspapers.journals
	musical companies	banda,choirs,opera companies,orchestras
	other collections of people	sports clubs,sports teams
	sports clubs	associations, theatres, companies,religious orders,youth organizations
Location	roads	streets, motorways
	trajectories	-
	regions	villages, towns, cities, provinces, countries, conti-nents, dioceses, parishes
	structures	bridges, ports, dams
	natural locations	mountains, mountain ranges, woods, rivers, wells,fields, valleys, gardens, nature reserves, allotments,beaches, national parks
	public places	squares, opera houses, museums, schools, mar-kets, airports, stations, swimming pools, hospitals,sports facilities, youth centers, parks, town halls,theaters, cinemas, galleries, camping grounds,NASA launch pads, club houses, universities, li-braries, churches, medical centers, parking lots,playgrounds, cemeteries

But in reality each English (source) named entity word can be translated into more than one possible target words. The transliteration aim to output the exact target word based on the pronunciation of the target language. For example the person name “Akash” can be translated into different target Kannada words as shown below.

AkAsh (ಆಕಾಶ್) →Correct target word according to Kannada pronunciation

akAsh (ಅಕಾಶ್)

Akash (ಆಕಶ್)

akash (ಅಕಶ್)

The transliteration model is built to capture the knowledge of bilingual phonetic association and subsequently it is applied to the transliteration process. To build the knowledge base, machine learning or rule-based algorithms are adopted. Most of the reported works utilize a phonetic clue to resolve the transliteration through a multiple step mapping rules and algorithms, such as dictionary lookup, statistical approach, rule-based and machine learning-based approaches have been used.

The transliteration model may be generative or discriminative. Generative model, builds a data model based on conditional probability density function.Discriminative Model, models the dependence of an unobserved variable ‘y’ on an observed variable ‘x’. The generative model is a full probability model of all variables, whereas a discriminative model provides a model only of the target variable(s) conditional on the observed variables. Thus a generative model can be used, to generate values of any variable in the model, whereas a discriminative model allows only sampling of the target variables conditional on the observed quantities.

In order to handle the transliteration problem from English to Kannada language, transliteration models were built by reformulating the transliteration task as sequence labelling and classification. English to Kannada transliteration systems were modelled using two different methods. The first transliteration model is based on a rule based approach where as the other transliteration model is based on statistical approach. In the first method rules were generated automatically using WEKA’s C4.5 decision tree classifier with features extracted from a parallel corpus. The second statistical transliteration model was developed using a publicly available structured output SVM algorithms. The parameters of the model were automatically learned from a bilingual proper name list by resolving different combinations of alignments and unit mappings. The systems have been designed and developed to resolve the complexities involved in

English to Kannada transliteration and to generate all possible phonetically equivalent transliterations. The performance of the models were evaluated and compared.

4.1 PROBLEMS IN TRANSLITERATION

Transliteration usually depends on context. For example:

- English grapheme ‘a’ can be transliterated into Kannada graphemes on the basis of its context, like ‘a’, ’aa’, ‘ei’ etc.
- Similarly ‘i’ can be transliterated either ‘i’ or ‘ai’ on the basis of its context.
- Also on the basis of its context, consonants like ‘c’, ’d’, ’l’, or ‘n’, has multiple transliterations in Kannada language.

The main reason of context dependency is that, vowels in English may correspond to long vowels or short vowels or some time combination of vowels in Kannada during transliteration. A transliteration system should be designed while considering all these barriers.

4.2 FORMULATING TRANSLITERATION AS SEQUENCE LABELLING AND CLASSIFICATION

Sequence labelling approach aims to assigning a label for each element in a sequence of observations. The main idea behind transliteration using sequence labelling is, mapping the letters from source script to the letters of the target script. This is a two step process in which the first step performs the segmentation of the source string into transliteration units. The second step involves the comparison of source language transliteration units with the target language units and resolve different combinations of alignments and unit mappings [150].

4.2.1 Problem Description

The problem can be stated formally as a sequence labelling problem from one language alphabet to other [151]. In the proposed problem the source language is English and the target language is Kannada. Consider a source language word x_1 x_2... x_i... x_N, where each x_i is treated as a word in the observation sequence. Let the equivalent target language orthography of the same word be y_1 y_2... y_i... y_N, where each y_i is treated as a label in the label sequence. The task here is to generate a valid target language word (label sequence) for the source language word (observation sequence). Each x_i is aligned with its phonetically equivalent y_i. Here the valid target language alphabet (y_i) for a source language alphabet (x_i) in the input source language word may depend on various factors like:

- The source language alphabet in the input word.
- The context alphabets surrounding source language alphabet (x_i) in the input word.
- The context alphabets surrounding target language alphabet (y_i) in the desired output word.

These generated features are used to generate rules using C4.5 decision tree algorithm or train the model using support vector machine. The generated rules and trained models are used to predict a target language word (Kannada) for new source language word (English).

4.3 TRANSLITERATION MODEL CREATION

In the proposed work, the English to Kannada transliteration problem was modelled as classification problem using two different approaches. The first transliteration model was based on a rule based approach using WEKA's C4.5 Decision tree classifier with features extracted from a parallel corpus. The second model was based on statistical approach using SVM. The model was trained with the same aligned parallel corpus which consists of 40,000 words containing names of various places in India.

4.3.1 Corpus Creation

The performance of the transliteration model hugely depends on the data with which it is trained. Hence it is important to have a large corpus with many examples that highlight the intricacies of the language. This will help the transliteration model to study the features of the target language so as to produce an accurate transliteration. A parallel corpus consisting of 40,000 Indian place names was created, from which the features were extracted. During the preprocessing phase, the source language names were segmented and aligned with the corresponding segmented target language names. The sequence of steps in preprocessing is as follows:

4.3.1.1 Romanization

WEKA’s C4.5 Decision tree classifier and SVM support only Roman (ASCII) character code but Dravidian language like Kannada does not support this code format and support only Unicode character. Unicode or officially called the Unicode Worldwide Character Standard is an entirely new idea in setting up binary codes for text or script characters. Unicode is an industry standard whose goal is to provide the means by which text of all forms and languages can be encoded for use by computers. So in order to map training and testing target data from Unicode to Roman and vice versa, mapping files were created. Using the mapping rules that defines English alphabet for each Kannada alphabet, Romanizes all the Kannada words. The Table 4.2 below shows the example for Romanization.

Table 4.2: Romanization

English place names	Kannada	Romanized Kannada
Karnataka	ಕರ್ನಾಟಕ	karnATaka
samathpur	ಸಮತ್ಪುರ್	samatpur
mumbai	ಮುಂಬೈ	muMbai

4.3.1.2 Segmentation

An important phase in machine transliteration process is segmentation and alignment. Efficiency of the transliteration model mainly depends on segmentation of source language and target language words into transliteration units (n-grams) and aligning the source language n-grams with corresponding target language n-grams. So before training the transliteration model, the transliteration units are obtained by segmenting the source and the target language words.

The rules for segmentation have been derived to suit phonetic reproduction of English names into Kannada.The English Names are segmented based on vowels, consonants, digraphs and trigraphs into English transliteration units. The segments or units can be synonymously called as English n-grams.

Vowels : a, e, i, o, u

Consonants : b, c, d, f, g, h, j, k, l, m, n, p, q, r, s, t, v, w, x, y, z

Digraphs: bh, ch, dh, gh, kh, ph, rh, sh, th, wh, zh, ng, nj

Trigraphs: ksh

When more than one vowel occur together, they are combined like aa, ae, ai, ao, au, ia, ie, io, etc., to form a single unit.

Similarly Romanized Kannada names are segmented based on vowels, consonants, digraphs and trigraphs into Kannada transliteration units. The segments or units can be synonymously called as Kannada n-grams. Table 4.3 shows example for segmentation.

Table 4.3: Segmentation

English	**Romanized Kannada**
k a r n a t a k a	k a R n A T a k a
s a m a t h p u r	s a m a t p u r
m u m b a i	m u M b a i

4.3.1.3 Alignment

Alignment is the final step in the preprocessing phase. Alignment is a most important phase in the transliteration process in which the one to one mapping between English language n-grams and the Kannada language n-grams is performed. Proper alignment of source language n-grams with phonetically equivalent target language n-grams is required to generate an efficient transliteration model. Alignment is based on the number of transliteration units in the segmented English and Romanized Kannada place names. The corresponding transliteration units in English and Romanized Kannada words are aligned if the number of units in the corresponding English and Romanized Kannada words are equal. Otherwise inserting an empty symbol '^' or combining the adjacent units in the

Romanized Kannada words, the units in the source place name are properly align to the unit in the target place name. Examples below shows, how the alignments of source and target words take place under different situation. Where ‘S’ and ‘T’ denotes source and target language words respectively.

Case 1: When the number of units are same:

Before alignment	After alignment
k a r n a t a k a (S)	*k | a | r | n | a | t | a | k | a*
(9 units)	(9 units)
k a R n A T a k a (T)	*k | a | R | n | A | T | a | k | a*
(9 units)	(9 units)

Case 2: Alignment of words by combining adjacent units:

Before alignment	After alignment
s a m a t h p u r (S)	*s | a | m |a | th | p | u | r*
(9 units)	(8 units)
s a m a t p u r (T)	*s | a | m | a | t | p | u | r*
(8 units)	(8 units)

Case 3: Alignment of words by inserting empty symbol:

Before alignment	After alignment
b o m b a y (S)	*b | o | m | b | a | y*
(6 units)	(6 units)
b A M b e (T)	*b | A | M | b | e |* ^
(5 units)	(6 units)

Convert these aligned source and target names in a column format based on the sequence labelling approach and SVM training input data format. The token is expected to be the first column of the line. The tag to predict takes the second column in the output. The column separator is the blank space. A sequence of tokens forms a word and each word is marked with boundary as shown below. The features required for training are defined with a window size of 5 elements and the core being the third position.

```
k k
a a
r R
n n
a A
t T
a a
k k
a a
. .
b b
o A
m M
b b
a e
y ^
. .
```

4.3.1.4 Mapping Analysis

From the results of segmentation and alignment, it is noted that an English n-gram can be mapped into one or more Kannada n-grams. A dictionary consisting of all English n-grams and their corresponding mapping Kannada n-grams (Class labels) was created from the training corpus. The frequency of an each English n-gram, i.e., number of occurrences of an English n-gram in the training corpus, along with the corresponding mapping label frequency is also maintained in the dictionary. The dictionary is referred during the training and the prediction process of transliteration.

4.4 TRANSLITERATION METHODOLOGIES USED

In the proposed work, English to Kannada transliteration task was modelled as classification problem using sequence labelling approach in two different ways. The first transliteration model was based on a rule based approach where as other one is based on statistical approaches. A well aligned parallel corpus consists of 40,000 words containing Indian place names were used to extract mapping features and dictionary. These generated features were used to generate rules using C4.5 decision tree algorithm. On the other hand, the parallel corpus features were used to train the model using support vector machine. The generated rules and trained models are used to predict a target language word (Kannada) for new source language word (English).

4.4.1 Transliteration Using WEKA

Fig. 4.1 shows the architecture of the proposed transliteration model. The transliteration model is divided into the following four phases: i) Preprocessing phase ii) Feature extraction and Dictionary creation Phase iii) Rule Generation Phase and iv)Transliteration phase.

4.4.1.1 Preprocessing

During the preprocessing phase, the source language names are romanized, segmented and aligned with the corresponding segmented target language names as explained in the section 4.3.

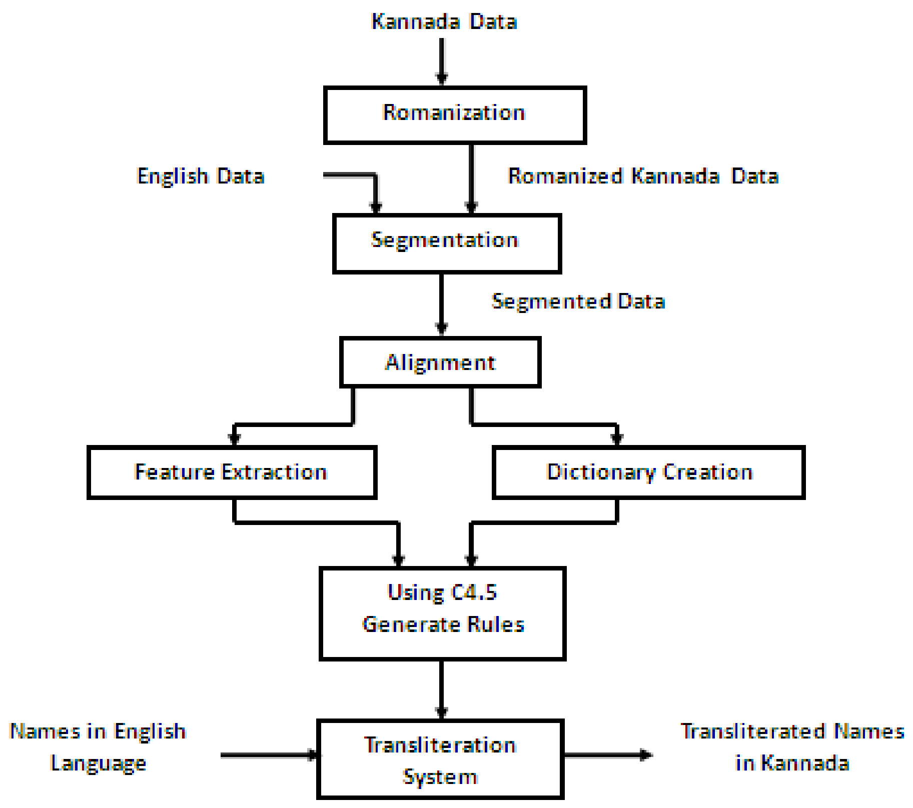


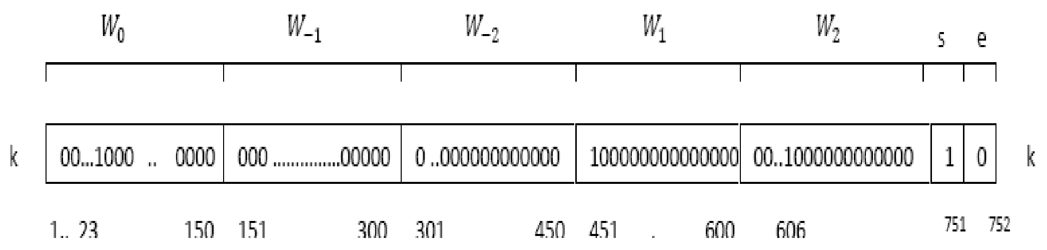
Fig. 4.1:Rule Based Transliteration System

4.4.1.2 Feature Extraction and Dictionary Creation

The feature extraction and dictionary creation was based on the aligned corpus. It was identified to consist of 150 unique English n-grams and 255 unique labels in the aligned corpus. A dictionary containing English n-grams with their mapping labels and the mapping label frequency was generated from the parallel corpus. For feature generation, the proposed system considered a centered window of five tokens, from which basic and n-gram patterns were evaluated to form binary features. The valid target language n-gram (yi) for a source language n-gram (xi) in the given source language input word is decided by considering the source language context features such as source language n-gram (xi), two left context n-grams (xi-2, xi-1) and two right context n-grams (xi+1, xi+2). The features specifying the start and end of words are also extracted and encoded as binary features. Thus for each ‘xi’, a vector of size 752binary features were created.The feature patterns corresponding to each source language n-gram ‘xi’ were generated to form a multi- class training set. For example the binary feature for n-gram ‘k’ in “Karnataka” can be identified as follows:

Source Name: k a r n a t a k a

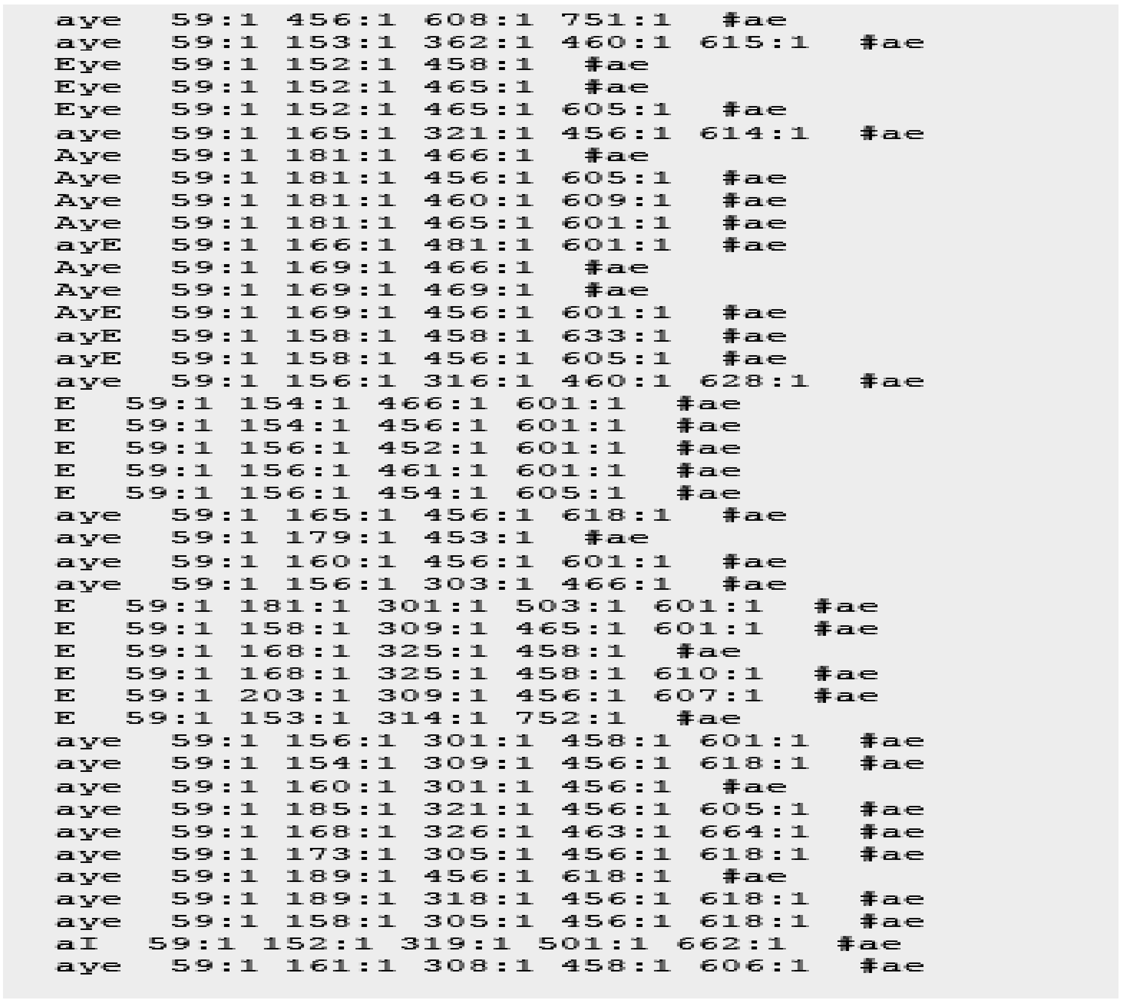
Target Name: k a R n A T a k a



For all 150 English n-grams, features are generated. For example the feature generated for n-gram ‘ae’ in sparse representation is as shown in Fig. 4.2. The left most column indicates the different class values associated with the same n-gram ‘ae’.

4.4.1.3 Generate Rules Using C4.5 Decision Tree Classifier

WEKA is a collection of machine learning algorithms implemented in Java [152]. WEKA consists of a large number of learning schemes for classification and regression numeric prediction like decision trees, support vector machines, instance-based classifiers, Bayes decision schemes, neural networks and clustering etc. It also provides Meta classifiers like bagging and boosting, evaluation methods like cross-validation and bootstrapping, numerous attribute selection methods and pre-processing techniques. A graphical user interface provides loading of data, applying machine learning algorithms and visualizing the built models. A Java interface available to all algorithms enables embedding them in any user’s program.



```
aye   59:1 456:1 608:1 751:1   #ae
aye   59:1 153:1 362:1 460:1 615:1   #ae
Eye   59:1 152:1 458:1   #ae
Eye   59:1 152:1 465:1   #ae
Eye   59:1 152:1 465:1 605:1   #ae
aye   59:1 165:1 321:1 456:1 614:1   #ae
Aye   59:1 181:1 466:1   #ae
Aye   59:1 181:1 456:1 605:1   #ae
Aye   59:1 181:1 460:1 609:1   #ae
Aye   59:1 181:1 465:1 601:1   #ae
ayE   59:1 166:1 481:1 601:1   #ae
Aye   59:1 169:1 466:1   #ae
Aye   59:1 169:1 469:1   #ae
AyE   59:1 169:1 456:1 601:1   #ae
ayE   59:1 158:1 458:1 633:1   #ae
ayE   59:1 158:1 456:1 605:1   #ae
aye   59:1 156:1 316:1 460:1 628:1   #ae
E   59:1 154:1 466:1 601:1   #ae
E   59:1 154:1 456:1 601:1   #ae
E   59:1 156:1 452:1 601:1   #ae
E   59:1 156:1 461:1 601:1   #ae
E   59:1 156:1 454:1 605:1   #ae
aye   59:1 165:1 456:1 618:1   #ae
aye   59:1 179:1 453:1   #ae
aye   59:1 160:1 456:1 601:1   #ae
aye   59:1 156:1 303:1 466:1   #ae
E   59:1 181:1 301:1 503:1 601:1   #ae
E   59:1 158:1 309:1 465:1 601:1   #ae
E   59:1 168:1 325:1 458:1   #ae
E   59:1 168:1 325:1 458:1 610:1   #ae
E   59:1 203:1 309:1 456:1 607:1   #ae
E   59:1 153:1 314:1 752:1   #ae
aye   59:1 156:1 301:1 458:1 601:1   #ae
aye   59:1 154:1 309:1 456:1 618:1   #ae
aye   59:1 160:1 301:1 456:1   #ae
aye   59:1 185:1 321:1 456:1 605:1   #ae
aye   59:1 168:1 326:1 463:1 664:1   #ae
aye   59:1 173:1 305:1 456:1 618:1   #ae
aye   59:1 189:1 456:1 618:1   #ae
aye   59:1 189:1 318:1 456:1 618:1   #ae
aye   59:1 158:1 305:1 456:1 618:1   #ae
aI   59:1 152:1 319:1 501:1 662:1   #ae
aye   59:1 161:1 308:1 458:1 606:1   #ae
```

Fig. 4.2: Features Generated for n-gram 'ae'

WEKA uses C4.5 algorithm to build decision trees from a set of training data using the concept of information entropy [152,153]. The training data is a set $S = s_1, s_2, \ldots$ of already classified samples. Each sample $s_i = x_1, x_2, \ldots$ is a vector where $x_1, x_2, \ldots$ represent attributes or features of the sample. The training data is augmented with a vector $C = c_1, c_2, \ldots$, where $c_1, c_2, \ldots$ represent the class to which each sample belongs. At each node of the tree, C4.5 chooses one attribute of the data that most effectively splits its set of samples into subsets enriched in one class or the other. Its criterion is the normalized information gain (difference in entropy) that results from choosing an attribute for splitting the data. The attribute with the highest normalized information gain is chosen to make the decision. The C4.5 algorithm then recurses on the smaller sublists. This algorithm has a few base cases as follows:

i) All the samples in the list belong to the same class. When this happens, it simply creates a leaf node for the decision tree saying to choose that class.

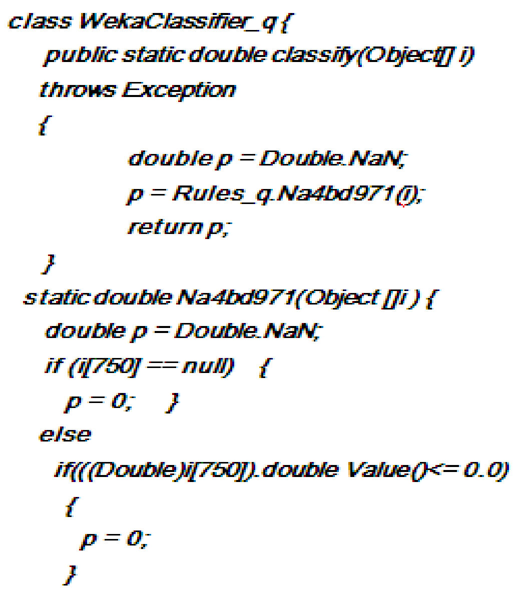
ii) None of the features provide any information gain. In this case, C4.5 creates a decision node higher up the tree using the expected value of the class and

iii) Instance of previously-unseen class encountered. In this case C4.5 creates a decision node again that higher up the tree using the expected value.

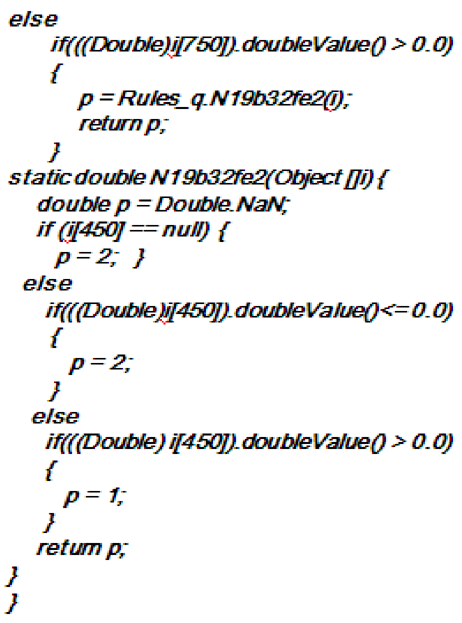
C4.5 Decision tree classification algorithm uses features and extract mapping rules for each and every n-gram and builds decision trees from these rules using the concept of information entropy. Each branch node in a decision tree represents a choice between a number of alternatives, and each leaf node represents a classification or decision. A leaf node attribute produces a homogeneous result (all in one class), which does not require additional classification testing. The attribute with the highest normalized information gain is chosen to make the decision. The information entropy is calculated as

$$\text{Entropy}(s) = -p_1 \log_{2}(p_1) - p_2 \log_{2}(p_2) - \ldots\ldots$$

For example the rulesgenerated for the n-gram 'q' from the feature pattern for the proposed transliteration system is as shown in Fig. 4.3.



```
class WekaClassifier_q {
   public static double classify(Object[] i)
   throws Exception
   {
            double p = Double.NaN;
            p = Rules_q.Na4bd971(i);
            return p;
   }
 static double Na4bd971(Object []i ) {
   double p = Double.NaN;
   if (i[750] == null)   {
     p = 0;    }
   else
    if(((Double)i[750]).double Value()<= 0.0)
     {
      p = 0;
     }
```



```
else
     if(((Double)i[750]).doubleValue() > 0.0)
     {
        p = Rules_q.N19b32fe2(i);
        return p;
     }
static double N19b32fe2(Object []i) {
   double p = Double.NaN;
   if (i[450] == null) {
      p = 2;  }
  else
     if(((Double)i[450]).doubleValue()<= 0.0)
     {
       p = 2;
     }
    else
     if(((Double) i[450]).doubleValue() > 0.0)
     {
       p = 1;
     }
    return p;
}
}
```

Fig. 4.3: RulesGenerated for n-gram ‘q’

4.4.1.4 Generate Transliteration System

The set of all generated rules were used to develop the transliteration model. During Transliteration phase, each word that is to be transliterated is first segment into a sequence of English n-grams and extracts the feature vector for each English n-gram. Using the transliteration model, predict the corresponding feature vector (class labels) for each n-gram in English test word. The sequence of predicted class labels forms a transliterated word for the given English word. The sequence of predicted class labels for each English word is converted into Unicode to form a transliterated Kannada word.

4.4.2 Transliteration Using Support Vector Machine Tool

The second transliteration model was based on statistical approach developed using publicly available structured output SVM algorithms [154]. The transliteration model is shown in Fig. 4.3.The whole model has three important phases: i) preprocessing phase, ii) training phase using SVM and iii) transliteration phase to generate Kannada transliterations for a given English Name.

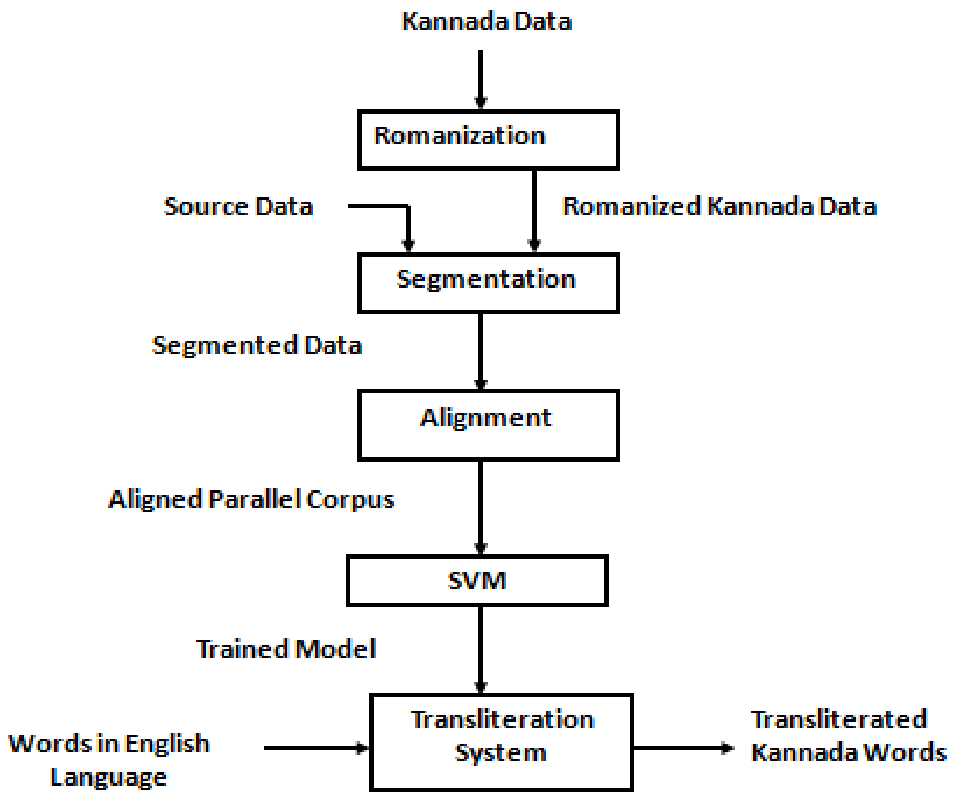


Fig. 4.4: Statistical Based Transliteration Model

4.4.2.1 Data Pre-processing Phase

During the pre-processing phase, the source language (English) names were romanized, segmented and aligned with the corresponding segmented target language (Kannada) names and a well fledged parallel corpus was created. The aligned source language and target language names were given as input sequence X=x1,x2, .., xn and label sequence Y=y1,y2,…,yn respectively in the two column format, as required by SVMTool for training the transliteration model. Each row, called as a token, contains an English n-gram and the corresponding aligned Kannada n-gram in two columns. The two columns were separated by space (s). It is important to note that, each token should have equal number of columns. A sequence of tokens forms a word and each word is marked with boundary. In the proposed system, the features required for training were defined with a window size of 5 elements and the core being the third position. A sample of five names in the SVMTool column format is given below:

sh S

i i

v v

a a

n n

. .

r r

a A

k k

e E

sh S

. .

r r

a a

m m

e E

sh S

. .

h h

a a

r r

i i

. .

v v

i i

sh S

a A

l l

. .

4.4.2.2 Training Phase

The preprocessing phase converts the corpus into SVM input file format in which aligned source language and target language names were given as input sequence and label sequence respectively for training. During training, features were extracted automatically by the SVMlight. A dictionary was generated from the aligned training corpus, with all possible class labels for each n-gram in the source language word. When considering the occurrence of an n-gram $x_i$labelled as y_i, this example is used as a positive example for class y_i and a negative example for all other classes. During the training phase the model was trained for every class in order to distinguish between examples of this class and all the rest. Also SVM generate a dictionary which consists of all possible class labels for each n-gram in the source language name. This dictionary avoids the excessive negative examples while training the model and training becomes faster.

SVMlight is an implementation of Vapnik's SVMs (Vapnik, 1995) in C, developed by Thorsten Joachims. SVM learning uses linear kernel and the learning time remains linear with respect to the number of examples. Training was performed using SVMTlearn component of SVMlight. The SVMTlearn algorithm extracts all feature patterns and other relevant information from the training corpus in the form different model files.The *config.svmt* file consists of a path to the 'Training Corpus' and a 'Model Name' that should be set before SVMTlearn algorithm. SVMTlearn behavior is easily adjusted through a configuration file. The usage of SVMTlearn is as shown below.

Usage: SVMTlearn [options] <config-file>

Options:

- V verbose 0: none verbose

1: low verbose [default]

2: medium verbose

3: high verbose

Example: SVMTlearn -V 2 config.svmt

4.4.2.3 Transliteration Phase

SVMlight uses SVMTagger module for testing the transliteration system. The names to be translated are first converted into a single coumn format with word boundary. The file name containing the ‘Test Data’ and the ‘Model Name’, which refer the models generated by the SVMTlearn are given as input arguments to the SVMTagger as shown below. The SVMTagger classifier predicts all possible class labels for a given sequence of source language alphabets and selects only the most probable class labels.

```
SVMTagger -T 0 KAN_TRANS <input.txt > output.txt
```

Where ‘KAN_TRANS’ is the model name, ‘inpu.txt’ and ‘output.txt’ indicates input file output files respectively. The predicted label sequence for each English word is converted into Unicode to obtain the transliterated Kannada word.

4.4.3 Evaluation and Results

Given a SVMTool predicted tagging output and the corresponding gold-standard, SVMTeval of SVMlight evaluates the performance of the English to Kannada transliteration system. Based on the dictionary created during training time, results are presented for different sets of n-gram such as known n-grams vs. unknown n-grams, ambiguous n-grams vs. unambiguous n-grams. A different view of these same results can be seen from the class of ambiguity perspective i.e. n-grams sharing the same kind of ambiguity may be considered together. N-grams sharing the same degree of disambiguation complexity, determined by the size of their ambiguity classes, can also be grouped. The usage of SVMTeval is as shown below.

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, in this case KAN_TRANS

gold: correct tagging file

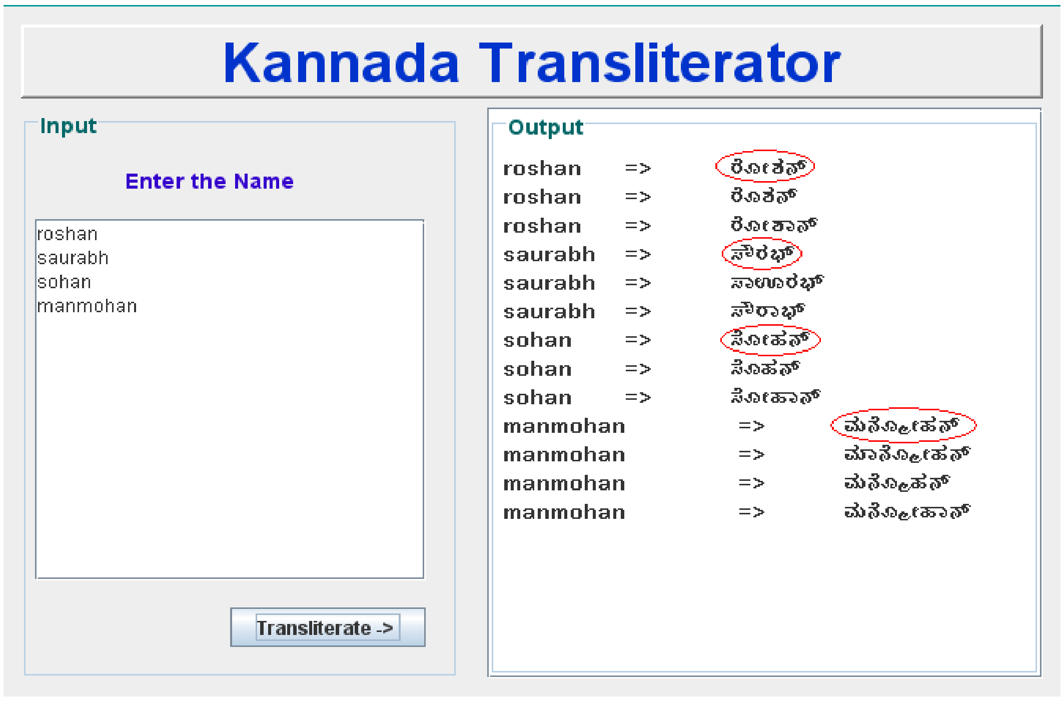
pred: predicted tagging file

A collection of 3000 names that were out of corpus was used for computing the efficiency of the proposed model. The model was evaluated by considering top 5 transliterations. Scores were calculated for individual transliterated segments, by comparing them with the set of good quality reference transliteration. The scores were then averaged over the whole corpus to find an estimate of the transliteration's overall quality.

Table 4.4: Transliteration Accuracy

Output	**Accuracy**	
	WEKA	**SVM**
Top1	72.4%	81.25%
Top2	76.2%	85.88%
Top3	78.1%	87.64%
Top4	80.5%	89.11%
Top5	83.38%	91.32%

The transliteration is considered correct, only if it exactly matches the one in the gold-standard. The transliteration system also generates a list of possible transliterations for any given name. The evaluation results of the transliteration models in terms of Top-1 i.e. the correct transliteration is the top candidate, Top2- the correct transliteration is the second candidate in the list, Top3- the correct transliteration is the third, Top4 and Top5 are presented. The number of possible transliterations can be increased by considering the next best class labels which in turn increase the accuracy of the transliteration model. The transliteration accuracy of the proposed models are shown in Table 4.4. The Fig. 4.5 shows the sample graphical user interface screen shot of the proposed English to Kannada transliteration system.



```
roshan
saurabh
sohan
manmohan
```

```
roshan     =>      ರೋಶನ್
roshan     =>      ರೊಶನ್
roshan     =>      ರೋಶಾನ್
saurabh    =>      ಸೌರಭ್
saurabh    =>      ಸಾಊರಭ್
saurabh    =>      ಸೌರಾಭ್
sohan      =>      ಸೋಹನ್
sohan      =>      ಸೊಹನ್
sohan      =>      ಸೋಹಾನ್
manmohan           =>      ಮನ್ಮೋಹನ್
manmohan           =>      ಮಾನ್ಮೋಹನ್
manmohan           =>      ಮನ್ಮೊಹನ್
manmohan           =>      ಮನ್ಮೋಹಾನ್
```

Fig. 4.5:GUI of English to Kannada Transliteration system

4.5 SUMMARY

In this research work, there are two different English to Kannada transliteration systems were developed to handle named entity translation using rule based as well as statistical approach. The performance of both of these systems depends on the size and scalability of the aligned corpus and the corpus creation is the key and most time consuming task in this work. In the rule based system, using WEKA's C4.5 Decision Tree algorithm, rules were generated automatically based on the features extracted from an aligned bilingual parallel corpus. These generated rules are then used to develop the transliteration model. Identifying the different source n-grams and target class labels in the corpus, extracting features for each source n-gram based on sequence labelling approach and generating rules for each source n-gram based on these features are the main objectives of the rule based transliteration system. The statistical based approach is simpler than the rule based approach. Once we a well organized, large sized aligned parallel corpus, and then we can easily learn the features and other peculiarities of the language using an SVM machine learning algorithm. The learned model is then used as a transliteration system.

The experiments showed that, the statistical based approach using SVM performbetterthan the rule based approach using WEKA. The performance of the proposed transliteration models can be improved by increasing the corpus size to cover more named entity words, which in turn extracts more feature information. The proposed SVM based transliteration model was successfully adapted in the proposed rule based English to Kannada MT system for translating named entities from English to Kannada language.

4.6 PUBLICATIONS

1. Antony P J, Ajith V P and Soman K P: "Feature Extraction Based English to Kannada Transliteration", Third International Conference on Semantic E-business and Enterprise Computing (SEEC-2010).

2. Antony P J, Ajith V P and Soman K P: "Kernel Method for English to Kannada Transliteration", International Conference on-Recent Trends in Information, Telecommunication and Computing (ITC 2010), Paper is archived in the IEEE Xplore and IEEE CS Digital Library.